université PARIS DIDEROT PARIS 7

Mémoire scientifique pour l'obtention d'une

HABILITATION À DIRIGER DES RECHERCHES

présentée à

L'UNIVERSITÉ PARIS DIDEROT – PARIS 7

par

Alexandre Miquel

Titre :

De la formalisation des preuves à l'extraction de programmes

Soutenu le mercredi 9 décembre 2009 devant le jury composé de

Rapporteurs :	Stefano Berardi	Università di Torino
	Thierry Coquand	Chalmers Institute, Göteborg
	Thomas Ehrhard	CNRS & Université Paris-Diderot
Examinateurs :	Peter Aczel	University of Manchester
	Jean-Louis Krivine	Université Paris-Diderot
	Benjamin Werner	INRIA & École Polytechnique

Chapitre 1

Introduction

Bien que traitant de sujets assez différents, les parties qui composent ce mémoire sont toutes reliées par une idée à la fois simple et très féconde, qui est que toute *preuve* (ou *démonstration*) mathématique[1] peut être vue comme un *programme*[2], lequel réalise un type de données déduit de la formule démontrée. Ce qu'on appelle la *correspondance de Curry-Howard* [44] est en réalité une correspondance multiple entre les concepts de la théorie de la démonstration[3] et ceux de la programmation fonctionnelle, puisqu'elle relie non seulement les *formules mathématiques* (côté logique) aux *types de données* (côté informatique) et les *preuves* aux *programmes*, mais aussi les *règles de déduction* aux *règles de typage*, l'*élimination des coupures* (ou la *normalisation*) à la *β-réduction*, le raisonnement par *récurrence* aux programmes définis par *récursion*, ainsi qu'à plus haut niveau, la structuration des mathématiques en *théories* et la programmation *modulaire*. Le dictionnaire ci-dessous donnera au lecteur une petite idée de la richesse de cette correspondance sans pour autant l'épuiser :

Théorie de la démonstration	**Programmation fonctionnelle**
Formule (proposition)	Type de données
Preuve (démonstration)	Programme (fonctionnel)
Conjonction $A \wedge B$	Produit cartésien $A \times B$
Implication $A \Rightarrow B$	Espace de fonctions $A \to B$
Règle de déduction	Règle de typage
Normalisation	β-réduction (calcul)
Vérification de preuve	Vérification de type
Preuve par récurrence	Programme récursif
Raisonnement classique	Programmation par continuations
Théorie (énoncés + preuves)	Module (interface + implémentation)
...	...

[1] Ici, les mots *preuve* et *démonstration* (que nous utiliserons dans ce mémoire comme des synonymes) doivent être compris dans leur sens formel — c'est-à-dire au sens de structures arborescentes finies construites suivant les règles définies par un langage formel donné.

[2] Exprimé le plus souvent dans un langage de programmation fonctionnelle.

[3] La théorie de la démonstration est la branche de la logique qui s'intéresse à la structure fine des démonstrations, et pas simplement aux questions de prouvabilité, de validité ou de cohérence, qui sont au cœur de la logique plus « traditionnelle ».

S'il est difficile de dater précisément la découverte de la correspondance de Curry-Howard — elle est en fait le produit d'un long processus de gestation au sein duquel les travaux de Curry et de Howard ne constituent que des étapes significatives[4] — il est indubitable que celle-ci a profondément bouleversé la théorie de la démonstration et la théorie de la programmation fonctionnelle, à la fois sur le plan théorique et sur le plan pratique.

Sur le plan théorique, la correspondance de Curry-Howard a renouvelé de fond en comble les outils et méthodes de la théorie de la démonstration, grâce au nouvel éclairage porté sur dynamique calculatoire des démonstrations. (Cet aspect dynamique des démonstrations-programmes, complètement évident côté programmation, était déjà connu côté logique sous le nom d'*élimination des coupures*, mais tenait alors une place assez marginale.)

Parallèlement à cette correspondance se sont développés de nombreux outils théoriques permettant d'analyser le comportement des programmes fonctionnels (et donc des preuves), notamment la sémantique dénotationnelle, la théorie des catégories (en lien avec le λ-calcul), la sémantique opérationnelle et plus récemment la sémantique des jeux. Les deux sous-produits les plus remarquables de cette maturation sont la théorie des types et les formalismes dérivés (tels que le calcul des constructions) ainsi que la logique linéaire, qui a beaucoup contribué à restructurer la théorie de la démonstration.

Sur le plan pratique, la correspondance de Curry-Howard permet de réduire le problème épineux de la *vérification de preuve* (qui consiste à déterminer si un texte donné constitue une preuve correcte de la formule voulue) au problème de la *vérification de type* (qui consiste à déterminer si un texte donné constitue un programme du type voulu) — un problème qu'on sait bien traiter depuis longtemps avec des algorithmes raisonnables. Il n'est donc pas étonnant que le développement de la correspondance preuves-programmes se soit accompagné d'un

[4]Pour donner une estimation assez large, on peut dire que la correspondance de Curry-Howard apparaît de manière embryonnaire dans les idées développées par Brouwer autour de l'intuitionnisme [17, 48] (1908), et qu'elle trouve sa forme définitive (du moins en ce qui concerne la logique intuitionniste) avec l'interprétation des preuves de l'arithmétique du second ordre dans le système F (Girard, 1970) [41, 44] et avec l'invention concomitante de la théorie des types (Martin-Löf, 1971) [67, 68]. Brouwer voyait déjà en une preuve de $A \Rightarrow B$ un « procédé » (*a method*, *a rule*) permettant de transformer une preuve de A en une preuve de B, et expliquait la transitivité de l'implication (« de $A \Rightarrow B$ et $B \Rightarrow C$, déduire $A \Rightarrow C$ ») par la composition du procédé conduisant de A à B avec le procédé conduisant de B à C. Au-delà de la méfiance viscérale qu'il entretenait vis-à-vis du formalisme en logique, Brouwer était objectivement handicapé par l'absence d'une définition formelle pour la notion de « procédé » au sens où il l'entendait. Celle-ci allait venir quasiment trente ans plus tard avec l'émergence de la théorie de la *calculabilité*, d'abord avec les travaux de Gödel, puis surtout avec les travaux de Church [20, 19] et de Turing (1936) [101] autour de l'*Entscheidungsproblem*. Il n'est donc pas étonnant que ce soit Kleene — étudiant de Church et condisciple de Turing à Princeton — qui le premier ait eu l'idée de définir formellement une transformation de preuves en programmes à travers ce qui allait devenir la théorie de la réalisabilité (1945) [54]. Cependant, la représentation malencontreuse des programmes par des entiers (sous l'influence de Gödel) plutôt que par des λ-termes (ainsi que Kleene l'avait initialement envisagé) allait maintenir le voile sur l'autre versant de la correspondance — entre les formules et les types — pendant encore quelques années. Ce fut le regain d'intérêt pour les interprétations fonctionnelles de l'arithmétique — notamment la *no counter-example interpretation* de Kreisel (1951) [55, 56] puis l'interprétation *Dialectica* de Gödel (1958) [45] — en lien avec le développement du λ-calcul typé (le système T de Gödel) qui permit finalement à Howard (1969) [52] d'expliciter la correspondance (et dans ce cas précis : un isomorphisme) entre la déduction naturelle intuitionniste et le λ-calcul simplement typé. Celui-ci avait été précédé par Curry et Feys (1958) [29] qui avaient dégagé une correspondance similaire entre la logique combinatoire typée et la déduction à la Hilbert.

foisonnement d'assistants à la démonstration (c'est-à-dire, essentiellement, des vérificateurs de preuve) basés sur ce principe : Coq [98, 15] (Rocquencourt), Lego (Edimbourg), Alfa/Agda (Chalmers), NuPrl (Cornell), Plastic (Durham), etc. Dans ces assistants, les démonstrations sont représentées par des programmes (de manière plus ou moins transparente vis-à-vis de l'utilisateur), et leur correction est vérifiée avec des algorithmes de typage très proches de ceux qu'on trouve dans les compilateurs de langages fonctionnels.

Formellement, la réduction de la vérification de preuve à la vérification de type n'est possible que si le système de types utilisé pour représenter les formules (et le langage de programmation utilisé pour représenter les preuves) est suffisamment expressif pour pouvoir traiter toutes les formules du langage mathématique — et pas seulement celles du calcul propositionnel. C'est pourquoi les divers assistants à la preuve construits sur ce principe sont tous basés sur la *théorie des types* de Martin-Löf — un langage de programmation fonctionnel dont le système de *types dépendants* permet de représenter toutes les formules et les démonstrations des mathématiques constructives — ou sur l'une de ses nombreuses extensions. Parmi ces extensions de la théorie des types, la plus utilisée est sans conteste le *calcul des constructions* (Coquand-Huet, 1985) qui combine la théorie de Martin-Löf avec le polymorphisme imprédicatif du système F de Girard, et qui constitue le noyau du langage utilisé dans le système Coq, mais aussi des langages utilisés dans les systèmes Lego et Plastic (son successeur) développés outre-Manche.

Une autre application pratique de la correspondance de Curry-Howard est l'*extraction de programme*, qui permet (entre autres) d'extraire un programme réalisant une spécification à partir d'une preuve d'existence constructive[5] — typiquement une fonction f de type $\mathbb{N} \to \mathbb{N}$ réalisant la spécification $A(x, f(x))$ (pour tout $x \in \mathbb{N}$) à partir d'une preuve constructive de la formule

$$\forall x \in \mathbb{N} \;\; \exists y \in \mathbb{N} \;\; A(x, y)\,.$$

Pendant longtemps, le mécanisme d'extraction de programmes en Coq a été limité aux preuves constructives, et ce n'est que récemment qu'il a été étendu aux preuves classiques, à l'aide de la théorie de la réalisabilité classique de Krivine. (J'aurai l'occasion de revenir sur ce point en détail au chapitre 4.)

1.1 Le choix du formalisme

Ayant effectué ma thèse au sein du projet Coq/LogiCal à l'INRIA Rocquencourt (dont le successeur est le projet TypiCal à Saclay), j'ai été amené très tôt à m'intéresser aux différents langages susceptibles de formaliser de manière effective (i.e. sur machine) l'ensemble des mathématiques.[6]

[5] Il serait plus correct de parler d'existence « intuitionniste », puisque le mécanisme d'extraction de programme mentionné ici fonctionne en logique intuitionniste comme en logique constructive. Rappelons qu'une logique constructive est une logique qui est à la fois intuitionniste *et* prédicative, en ce sens qu'elle n'autorise pas de définition de type circulaire. Dans la suite de cette introduction, on utilisera le mot « constructif » dans le sens d'« intuitionniste » (qui est beaucoup moins parlant), même s'il est clair que pour un mathématicien constructiviste, aucun des systèmes que nous allons étudier dans la suite de ce mémoire n'est constructif — pas même ceux du chapitre 2, pourtant tous intuitionnistes.

[6] Au cours de ma thèse, j'ai défini et étudié une variante du calcul des constructions — le *calcul des constructions implicite* — dont l'étude a été depuis reprise par Barras et Bernardo, lesquels ont même développé un prototype basé sur mon calcul.

S'il existe une pléthore de systèmes formels en logique, les systèmes qui sont à même d'exprimer de larges pans des mathématiques (disons au-delà de l'arithmétique) sont en nombre beaucoup plus restreint. On peut à ce jour les classer en deux familles : les théories des ensembles et les théories des types.[7]

1.1.1 La théorie des ensembles

Historiquement, la théorie des ensembles a été introduite (sous diverses formes) comme une théorie unificatrice des mathématiques, c'est-à-dire comme une théorie permettant d'exprimer dans un même formalisme toutes les branches des mathématiques : la géométrie, l'algèbre, l'arithmétique, l'analyse, la topologie, etc. Issue des travaux pionniers de Cantor (motivés initialement par des questions pointues d'analyse), elle a connu plusieurs définitions formelles, d'abord par Frege (qui produisit malheureusement un système incohérent) puis par Zermelo (à qui l'on doit notamment l'axiome du choix), avant de trouver sa forme définitive dans ce qu'on appelle aujourd'hui la *théorie des ensembles de Zermelo-Fraenkel* (ZF) — grâce à l'introduction du schéma de remplacement, découvert indépendamment par Fraenkel et par Skolem.[8]

Conceptuellement, la théorie des ensembles est bâtie toute entière autour de la seule notion d'ensemble ; les autres notions mathématiques pourtant plus familières — les nombres, les matrices, les fonctions — y sont d'emblée bannies. Comme les ensembles sont les seuls objets de la théorie, les éléments des ensembles sont eux-mêmes des ensembles, dont les éléments sont encore des ensembles, qui contiennent d'autres ensembles d'ensembles... lesquels au final ne contiennent que le vide. Paradoxalement, c'est cette pauvreté conceptuelle poussée à l'extrême qui fait toute la force de la théorie. Car à l'usage, la notion d'ensemble apparaît tellement informe et malléable qu'il est possible d'y façonner à peu près tout ce qui existe dans l'univers des mathématiques — et en réalité infiniment plus d'objets, pour la plupart complètement inutiles. Ainsi à partir des ensembles on reconstruira les entiers naturels (suivant le codage de von Neumann), mais aussi les entiers relatifs, les entiers rationnels, et finalement les nombres réels (suivant la construction de Dedekind ou de Cauchy), les nombres complexes, les fonctions et les espaces de fonctions, ainsi que tous les objets dignes d'intérêt en mathématiques.

Cependant, l'utilisation de la théorie des ensembles en vue d'une formalisation *effective* des mathématiques[9] — par opposition à la formalisation *potentielle* entrevue et mise en œuvre par Bourbaki — fait vite apparaître les insuffisances du formalisme proprement dit.

Le premier problème vient du langage lui-même. Dans sa formulation standard (au premier ordre), la théorie des ensembles ne comporte ni symbole de

[7] Les diverses variantes de l'arithmétique de Peano — au second ordre ou à l'ordre supérieur — appartiennent incontestablement à la deuxième famille, dans la mesure où le système de déduction de chacune de ces théories peut être présenté sous la forme d'un système de types purs (PTS) étendu avec des entiers primitifs (AF2, AFω, etc.)

[8] Il existe également d'autres formalismes similaires — comme la théorie de von Neumann-Bernays-Gödel (NBG) — ou plus exotiques — comme les surprenantes *New Foundations* (NF) de Quine et ses diverses déclinaisons (NFU, etc.) — dont nous ne parlerons pas ici.

[9] C'est-à-dire une formalisation sur ordinateur, où les théories sont stockées (après vérification par la machine) dans des bases de données. On n'insistera jamais assez sur le fait que la correction d'un raisonnement mathématique (pour peu que toutes ses étapes aient été explicitées) est une propriété *objective* au sens premier du terme, c'est-à-dire une propriété testable par un *objet* — ici l'ordinateur — sans avoir besoin de référer à un quelconque sujet.

constante ni symbole de fonction, et ses seuls termes sont les variables : on ne peut donc parler des ensembles qu'à travers des énoncés existentiels. Pour pouvoir désigner les objets, il faut sortir du formalisme initial et introduire des notations pour représenter les notions courantes. En ce qui concerne des notions simples telle que les couples ou l'ensemble des parties, leur désignation s'effectue en étendant le formalisme avec des symboles de Skolem (et les axiomes qui les accompagnent), extension dont on sait qu'elle est conservative. Le problème devient beaucoup plus délicat dès qu'il s'agit d'introduire des symboles avec des lieurs, par exemple pour désigner des ensembles définis par compréhension, des fonctions, ou différentes formes de sommation. (C'est un problème sur lequel nous aurons l'occasion de revenir en détail au chapitre 2, section 2.3.4.)

Le second problème vient du fait que la théorie des ensembles n'accorde aucune place au calcul, ou plutôt aucune place spécifique, puisqu'elle traite le calcul uniquement à travers le raisonnement (en décomposant chaque étape de calcul en un grand nombre d'étapes de raisonnement). En théorie des ensembles, une proposition aussi élémentaire que « $2 + 2 = 4$ » nécessite une démonstration — fort longue, mais bien moins que la démonstration de « $21 + 21 = 42$ » — ce qui est un comble quand on pense qu'une machine peut vérifier très rapidement ce type d'assertion sans qu'on ait à lui fournir la moindre justification. Ici, c'est bien la notion primitive qui est en cause, cette notion d'ensemble qui se prête tellement moins facilement au calcul que la notion de fonction...[10]

Malgré ses défauts, la théorie des ensembles est à la base d'un des plus anciens assistants à la démonstration : le système Mizar (1973) [100] qui fait encore aujourd'hui l'objet d'un développement actif, et dont les librairies mathématiques surpassent par leur richesse celles qui sont fournies avec la plupart des assistants à la preuve conçus sur des formalismes plus modernes.

1.1.2 La théorie des types

L'idée de stratifier les objets mathématiques autour d'un *système de types* est une idée ancienne qui remonte aux *Principia mathematica* (1910–1913) [104] de Russell et Whitehead. (La notion de type ramifié ayant été introduite par Russell [95] pour échapper aux antinomies de la théorie des ensembles naissante.) Mais c'est incontestablement avec Martin-Löf [68, 69, 83] qu'apparaît la théorie des types sous sa forme moderne, dans un manuscrit non publié qui allait faire date dans le domaine : *A theory of types* (1971) [67]. Malgré l'incohérence du formalisme qui y est présenté (découverte peu de temps après par Girard [41]), on y trouve la plupart des ingrédients de la théorie des types moderne : le produit dépendant, les jugements de typage, la règle de conversion et même les techniques de preuve de normalisation forte pour les types dépendants.[11]

Contrairement aux systèmes de types antérieurs [95, 104, 21], la théorie des types de Martin-Löf tire parti de la correspondance de Curry-Howard pour in-

[10] Évidemment, on ne saurait reprocher aux concepteurs de la théorie des ensembles d'avoir négligé le calcul : du temps de Zermelo, Fraenkel et Skolem, on était bien loin d'imaginer l'importance pratique que le calcul allait revêtir à la fin du 20e siècle. Tout au plus commençait-on à se demander si les mathématiques étaient « automatisables », sans avoir encore la moindre définition mathématique à proposer pour ce mot.

[11] Rappelons que la preuve de normalisation forte présentée dans [67] n'est pas *formellement* fausse, mais qu'elle est effectuée dans un système incohérent. De fait, les techniques de réductibilité qui y sont présentées (adaptées de la technique de Girard pour le système F) restent valables dans des formalismes tels que le calcul des constructions.

tégrer complètement le système de déduction à l'intérieur du système de types. Dans un tel formalisme, les formules sont représentées par des types d'une forme particulière — les types propositionnels — et les preuves d'une formule donnée sont tout simplement identifiés avec les termes (ou programmes) qui habitent le type par lequel la formule est représentée. Par exemple, la formule $A \Rightarrow A$ est représentée par le type $A \to A$ des fonctions de A de A — la formule A étant à présent vue comme le type de ses preuves — tandis que la preuve canonique de cette implication est représentée par le λ-terme $\lambda x : A \,.\, x$ (ici avec une annotation de type « $: A$ » sur la variable x). L'intérêt d'un tel formalisme est très clair du point de vue de l'implémentation, puisque non seulement les formules et les preuves n'ont plus besoin d'être définies séparément, mais surtout, la tâche épineuse qui consiste à vérifier si un texte donné constitue effectivement une preuve d'une formule donnée peut être entièrement dévolue au système de vérification de type. (On conçoit bien que dans un tel cadre, la vérification de type se doit de rester décidable, ce qui justifie amplement la présence d'annotations supplémentaires dans les programmes-preuves.)

Bien entendu, une telle façon de procéder n'est envisageable que dans la mesure où le système de types est suffisamment riche pour pouvoir exprimer suffisamment de formules — c'est-à-dire au moins toutes les formules du calcul des prédicats et de l'arithmétique — ainsi que toutes les règles de déduction qui vont avec (du moins en ce qui concerne le fragment intuitionniste). En pratique, ceci est rendu possible par l'introduction de deux nouvelles constructions de types, à savoir le produit dépendant $\Pi x : A \,.\, B(x)$ et la somme dépendante $\Sigma x : A \,.\, B(x)$, qui correspondant respectivement à la quantification universelle $\forall x : A \,.\, B(x)$ et à la quantification existentielle $\exists x : A \,.\, B(x)$ à travers la correspondance de Curry-Howard. Ce qui est encore plus remarquable, c'est que ces constructions peuvent être vues comme des généralisations de deux constructions déjà connues auparavant — à savoir le type flèche $A \to B$ et le type produit cartésien $A \times B$ — et ne nécessitent donc aucune extension de la syntaxe des termes (sauf en ce qui concerne les annotations de type).

Un autre trait distinctif de la théorie des types tient à la place occupée par le calcul dans le formalisme. Par le jeu des dépendances, les formules-types sont amenées à contenir des expressions susceptibles de faire l'objet d'un calcul, comme par exemple la formule $2+2 < 5+7$ dont les deux membres se réduisent respectivement sur les expressions 4 et 12. Pour traiter cette forme d'équivalence calculatoire entre deux formules — ici entre la formule $2+2 < 5+7$ et la formule $4 < 12$ — on introduit dans le système de types (qui, rappelons-le, contient aussi le système de déduction) une règle de *conversion*

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M : A'} \; {\scriptstyle A \cong A'}$$

qui exprime que tout terme M de type A (ou toute preuve M de la formule A) est aussi un terme de type A' dès lors que les types A et A' sont *convertibles*, c'est-à-dire équivalents modulo un nombre fini de transformations purement calculatoires (c'est ici ce qu'exprime la condition de bord $A \cong A'$).

Là encore, cette règle permet d'intégrer directement les étapes de calcul au raisonnement de façon transparente pour les preuves. Ainsi une preuve M de l'égalité $4 = 4$ (qui découle directement de la réflexivité de l'égalité) constitue-t-elle aussi une preuve de la formule $2 + 2 = 4$. Les étapes de raisonnement purement calculatoires qui prenaient tant de place dans les preuves formelles

en théorie des ensembles disparaissent donc purement et simplement dans les termes de preuves en théorie des types. En réalité, on a surtout troqué de l'espace mémoire contre du temps de calcul, car l'équivalence calculatoire entre deux formules A et A', bien qu'invisible au niveau des preuves, doit faire l'objet d'une vérification — c'est-à-dire, précisément, d'un calcul — au moment de la vérification de type (ou de la vérification de preuve). Une vérification qui sera d'autant plus longue que le calcul correspondant est complexe.[12]

1.1.3 D'un paradigme à l'autre

L'existence de deux grand paradigmes pour unifier les mathématiques — qui se déclinent eux-mêmes en un grand nombre de formalismes différents — pose un certain nombre de problèmes pratiques et théoriques.

En pratique, la diversité des formalismes « sur le papier » se retrouve naturellement dans le cœur des assistants à la preuve, ce qui suscite des problèmes d'interopérabilité évidents. Aujourd'hui encore, on trouve très peu d'outils permettant de transcrire un développement formalisé dans un système donné dans la syntaxe d'un autre système, et d'autant moins lorsque les systèmes sont basés sur des paradigmes différents.[13] Le problème se pose également lorsqu'il s'agit de formaliser un texte mathématique rédigé en style ensembliste[14] dans un assistant à la preuve basé sur la théorie des types. Quiconque a déjà formalisé un morceau de mathématiques en Coq sait qu'un tel travail est loin d'être purement mécanique, et requiert même une certaine dose d'imagination créatrice.

Ces considérations m'ont amené au cours de ma dernière année de thèse à m'intéresser de près au problème de la traduction de la théorie des ensembles en théorie des types, et vice-versa.

Rappelons qu'une traduction d'un système S_1 vers un système S_2 consiste essentiellement en la donnée de deux transformations syntaxiques de S_1 vers S_2 : une transformation $A \mapsto A^*$ sur les formules et une transformation $\pi \mapsto \pi^*$ sur les preuves, avec la contrainte que si π est une preuve de A dans S_1, alors π^* doit être une preuve de A^* dans S_2. (Le schéma est sensiblement plus complexe dans le cas où S_1 est une théorie des types, mais l'idée reste la même.) Si de plus on impose que la traduction de la formule $\bot_1$ représentant le faux dans S_1 est équivalente à la formule $\bot_2$ représentant le faux dans S_2 (i.e. $S_2 \vdash \bot_1^* \Leftrightarrow \bot_2$), alors la traduction nous permet d'établir la *cohérence relative* du système S_1 par rapport au système S_2, un résultat qu'on abrège souvent en $S_1 \leq S_2$.[15]

[12] Il est intéressant de remarquer que la règle de conversion tenait au départ une place relativement marginale dans la théorie des types de Martin-Löf. Ce n'est qu'avec le développement des assistants à la preuve qu'on a pris conscience de l'importance pratique de cette règle, au point d'en faire une méthodologie de preuve : la *preuve par réflexion*, qui consiste à systématiser l'utilisation des arguments purement calculatoires dans les preuves (au détriment des autres formes de raisonnement) pour en réduire la taille. La démonstration du théorème des quatre couleurs est un exemple typique de cette méthodologie de démonstration.

[13] Il n'y a à ma connaissance aucun outil permettant de transformer une preuve Mizar en une preuve Coq, et vice-versa. Au-delà des difficultés techniques liées à la présence de nombreuses constructions idiomatiques dans les deux langages, le problème réside surtout dans le fait que les deux formalismes ne sont pas basés sur le même paradigme.

[14] Ce qui est le cas de l'immense majorité des textes mathématiques.

[15] Pour avoir une preuve de cohérence relative réellement satisfaisante, il faut en plus ajouter une troisième contrainte qui est que la preuve de correction de la traduction $A/\pi \mapsto A^*/\pi^*$ doit être effectuée dans un système formel plus faible que S_2. Car il ne faut jamais oublier qu'à chaque fois qu'on établit l'inégalité $S_1 \leq S_2$ (au sens défini plus haut) dans un système

À ce stade, il est important de souligner que les deux sens de traduction — de la théorie des types vers la théorie des ensembles ou de la théorie des ensembles vers la théorie des types — n'offrent pas du tout le même niveau de difficulté.

Définir une traduction de la théorie des types vers la théorie des ensembles est un exercice relativement facile, puisqu'il s'agit de représenter une structure très contrainte par le typage dans une structure ensembliste qui l'est beaucoup moins. Comme la plupart des concepts de la théorie des types ont de fait un équivalent immédiat en théorie des ensembles, la définition d'une telle traduction se résume essentiellement à un exercice de reformulation dont la difficulté technique ne concerne réellement que les traits les plus avancés du formalisme source. (Par exemple : les définitions inductives et/ou récursives.)

Définir une traduction de la théorie des ensembles vers la théorie des types est en revanche un problème plus délicat, puisqu'il s'agit de représenter une structure ensembliste peu contrainte dans une structure très contrainte par le typage, sans oublier personne au passage. Pour cela, il est nécessaire de dégager dans le formalisme cible une structure suffisamment malléable pour supporter toutes les opérations ensemblistes (à travers un codage approprié), y compris celles qui n'auraient manifestement aucun sens dans un cadre typé.[16]

Il n'existe aujourd'hui (en tout cas à ma connaissance) que deux méthodes pour traduire une théorie des ensembles vers une théorie des types.

La première méthode, introduite par Aczel [2], consiste à représenter les ensembles par les habitants d'un type d'arbres bien fondés à branchement fortement infinitaire — ce qui requiert l'existence (dans le formalisme cible) d'un type d'arbres avec des possibilités de branchement suffisamment grandes pour permettre la représentation d'ensembles de cardinal arbitrairement élevé. Pour cette raison, cette méthode est particulièrement adaptée aux théories des types disposant d'un mécanisme de définitions inductives généralisées comme la théorie des types de Martin-Löf ou le calcul des constructions inductives.

La seconde méthode, que j'ai introduite dans la troisième partie de ma thèse [74], consiste à représenter les ensembles par des graphes pointés. (Cette méthode est elle-même inspirée de travaux plus anciens de Forti, Honsell [35] et Aczel [1] en théorie des ensembles, autour de l'axiome d'antifondation.) Le principal avantage de cette méthode de représentation (sans doute un peu plus délicate à manipuler que la précédente) est de nécessiter beaucoup moins de structure dans le formalisme cible. En pratique, on peut l'utiliser dans les systèmes de types purs (PTS) [10, 11] sans aucun type inductif, ce qui m'a permis dans ma thèse de démontrer l'existence d'un plongement de la théorie des ensembles de Zermelo intuitionniste dans le système $F\omega.3$, un sous-PTS strict du calcul des constructions avec univers.

L'exploration des possibilités offertes par cette seconde méthode de représentation des ensembles en théorie des types a été mon principal sujet de recherche dans les premières années qui ont suivi ma thèse, et les résultats que j'ai obtenus dans cette direction font l'objet du chapitre 2 de ce mémoire. Je me suis plus particulièrement attaché à deux problèmes, à savoir :

---

formel S_3, ce n'est en réalité pas la cohérence de S_1 relativement à S_2 qu'on a démontrée, mais la cohérence de S_1 relativement à S_2 *et* à S_3. En pratique, cette troisième contrainte est facilement satisfaite dans la mesure où la plupart des preuves de ce type sont purement combinatoires, et peuvent être aisément formalisées dans un système tel que PRA (arithmétique primitive récursive), sinon PA.

[16] Par exemple : l'opération « $\cos \cup \sqrt{2}$ », parfaitement légale en théorie des ensembles.

1. La définition d'un système de types capable d'interpréter tout le fragment intuitionniste de la théorie des ensembles de Zermelo-Fraenkel (ma thèse n'ayant abordé que la traduction de la théorie de Zermelo).
2. La définition d'un système de types capturant exactement la force de la théorie des ensembles de Zermelo (puisque le système présenté dans ma thèse n'avait permis de donner qu'une majoration stricte).

J'ai ainsi proposé deux solutions au premier problème : d'abord un système de types majorant strictement la force de la théorie des ensembles de Zermelo-Fraenkel [75] (présenté à la section 2.2), puis un système de types plus fin capturant exactement la force de cette théorie (présenté à la section 2.5). La solution du second problème [76] (que j'avais conjecturée dans ma thèse) est quant à elle présentée à la section 2.3 ; et une variante de cette solution sous la forme d'une théorie des graphes pointés formulée en déduction modulo [31] (en collaboration avec Gilles Dowek) est esquissée dans la section 2.4.

1.1.4 Types inductifs et filtrage

Mon intérêt pour les diverses déclinaisons de la théorie des types m'a naturellement conduit à m'intéresser aux mécanismes de définitions inductives généralisées [102, 86] qu'on trouve dans la théorie des types de Martin-Löf ou dans le calcul des constructions inductives. J'ai toujours été frappé par la grande complexité des règles d'inférence associées aux constructions inductives, surtout en regard de la simplicité des règles qui définissent le PTS [11] sous-jacent.

Il m'est alors apparu que la complexité de ces règles d'inférence venait en grande partie de la façon dont le filtrage lui-même est présenté dans les langages fonctionnels, avec des règles de réduction monolithiques de la forme :

$$\begin{array}{l}\mathsf{match}\ \ c_i(N_1,\cdots,N_{k_i})\ \ \mathsf{with}\\ \mid\ \ c_1(x_{1,1},\ldots,x_{1,k_1})\ \mapsto\ M_1\\ \quad\ \vdots\\ \mid\ \ c_n(x_{n,1},\ldots,x_{n,k_n})\ \mapsto\ M_n\end{array}\quad\longrightarrow\quad M_i\{x_{i,1}:=N_1;\ldots;x_{i,k_i}:=N_{k_i}\}$$

Ceci m'a conduit à m'intéresser de plus près au filtrage algébrique implémenté dans les langages fonctionnels de la famille ML, et à chercher un formalisme où ce filtrage serait présenté d'une manière plus atomique. De ce projet est né le λ-calcul avec constructeurs [4, 5] auquel est consacré le chapitre 3 de ce mémoire. Dans ce chapitre je présenterai le formalisme et ses principales propriétés (démontrées en collaboration avec Ariel Arbiser et Alejandro Ríos, de l'Université de Buenos Aires), à savoir : la propriété de Church-Rosser modulaire (section 3.3) et le théorème de séparation (section 3.4).

1.2 Extraction de programmes

L'autre grande application de la correspondance de Curry-Howard est l'extraction de programmes. Depuis Gentzen [39] et Prawitz [89] on sait que la logique intuitionniste vérifie deux propriétés structurelles fondamentales, à savoir la *propriété de la disjonction* et la *propriété du témoin* (qui sont toutes les deux fausses en logique classique[17]).

[17] Sauf dans les théories finitaires et dans certaines théories infinitaires moins expressives que l'arithmétique de Peano, telles que l'arithmétique de Presburger. On peut toujours forcer la

La propriété de la disjonction exprime que si une disjonction $A \vee B$ est prouvable sans hypothèse, alors l'un (au moins) de ses deux membres A ou B est prouvable (sans hypothèse). La propriété du témoin exprime quant à elle que si une formule existentielle $\exists x\, A(x)$ est prouvable sans hypothèse, alors une de ses instances $A(t_0)$ est prouvable (sans hypothèse) pour un certain terme t_0 du langage. Ces deux propriétés sont vraies dans le calcul des prédicats intuitionniste « pur » (i.e. sans axiome), mais aussi dans les diverses déclinaisons de l'arithmétique de Heyting et de la théorie des types.

À travers la correspondance de Curry-Howard (du moins dans un cadre constructif) la propriété du témoin se comprend très bien : une preuve de la formule $\exists x\, A(x)$ n'est rien d'autre qu'un programme p de type $\Sigma x : T\,.\,A(x)$, où $\Sigma x : T\,.\,A(x)$ désigne le type des paires (t_0, p_0) formées d'un objet $t_0 : T$ et d'un programme $p_0 : A(t_0)$ correspondant à une preuve de $A(t_0)$. (En supposant que la quantification existentielle porte sur les objets de type T.)

Si le système de types utilisé est correct — c'est-à-dire si les programmes d'un certain type se réduisent effectivement vers des valeurs qui ont la forme prescrite par ce type — alors le programme p devra se réduire sur le couple témoin/justification (t_0, p_0) recherché. (On reviendra un peu plus loin sur les outils qui permettent d'établir cette propriété de correction.)

Plus généralement, une preuve constructive d'une formule de la forme

$$\forall x\ \exists y\ A(x, y)$$

correspond à un programme p de type

$$\Pi x : T\,.\,\Sigma y : U\,.\,A(x, y)\,,$$

c'est-à-dire à un programme p qui, lorsqu'on l'applique à un argument $t_0 : T$ arbitraire, retourne un couple (u_0, p_0) où $u_0 : U$ est le témoin recherché et où $p_0 : A(t_0, u_0)$ désigne la justification qui l'accompagne. Comme en pratique c'est le témoin u_0 qui nous intéresse et non la justification p_0, on construit à partir du programme p la fonction

$$\lambda x\,.\,\mathsf{fst}(p\ x) \quad : \quad T \to U$$

qui à tout élément $t : T$ associe un objet $u : U$ tel que $A(t, u)$ — débarrassé de la justification désormais inutile.

1.2.1 Extraction en logique intuitionniste

Le mécanisme qui vient d'être décrit constitue la base de l'extraction de programmes en logique intuitionniste. Sur le plan théorique, il s'agit donc de lire une preuve constructive comme un programme pour obtenir le programme recherché — à quelques modifications près. À travers la correspondance de Curry-Howard, la fonction qui extrait un programme à partir d'une preuve est donc « quasiment » la fonction identité.

Évidemment, toute la difficulté dans la mise en œuvre concrète d'un tel mécanisme d'extraction réside dans ce « quasiment ». Car le calcul de la justification qui accompagne le témoin peut être très coûteux — il l'est en général

propriété de la disjonction et la propriété du témoin dans une théorie classique en la saturant de témoins de Henkin et en la complétant (dans cet ordre). Mais la théorie qu'on obtient alors perd tout caractère récursif, et devient précisément ce qu'on appelle un modèle.

plus que le calcul du témoin — et il ne suffit pas (même dans une stratégie en appel par nom) d'effacer cette justification en fin de chaîne pour effacer le calcul nécessité par la production de cette justification.

Il existe plusieurs stratégies pour éviter que la production de la justification ne prenne trop de temps de calcul.

Une première stratégie consiste à identifier dans le langage des formules une classe particulière de formules, les *formules de Harrop*, dont on peut à l'avance prédire le comportement calculatoire des termes de preuves. L'intérêt de ces formules est qu'on peut purement et simplement ignorer leurs preuves au moment de l'extraction, dans la mesure où les programmes qu'elles engendrent ont un contenu calculatoire « vide ». Cette stratégie, qui est à la base du mécanisme d'extraction de l'assistant à la preuve Minlog [96], est d'autant plus efficace que le corps de la formule existentielle $\exists x\, A(x)$ dont on cherche à extraire un témoin (à partir d'une preuve de la formule) est proche d'une formule de Harrop. Ce qui est par définition le cas si la formule en question est Σ^0_1. Le principal inconvénient de cette stratégie est qu'elle ne fonctionne vraiment bien que pour des théories assez faibles — du moins au regard des systèmes que nous allons considérer dans ce mémoire — à savoir l'arithmétique de Heyting et certaines de ses extensions (par exemple avec le principe de récursion barrée).

Une autre stratégie, qui est mise en œuvre dans le schéma d'extraction constructive de Coq [65], consiste à scinder l'espace des types en deux catégories : les types *informatifs* et les types *non informatifs*, sachant que seuls les termes habitant les types informatifs seront pris en considération au cours de l'extraction, les autres étant ignorées. Dans Coq, les types qui vivent dans les sortes `Set` et `Type` sont considérés comme informatifs, tandis que les types vivant dans la sorte `Prop` sont considérés comme non informatifs. Cependant, cette stratégie a l'inconvénient d'imposer un certain nombre de restrictions sur le système de types (et notamment sur ses schémas d'élimination) afin d'interdire toute possibilité de construire un objet informatif à partir d'un objet non informatif, qui aura disparu au moment de l'extraction.[18] On notera qu'avec cette approche, c'est à l'utilisateur de choisir les sortes dans lesquelles il définit ses objets et ses spécifications, en faisant attention à respecter les contraintes imposées par les schémas d'élimination de Coq.

1.2.2 Typage et réalisabilité

À ce stade de la discussion, il est important de souligner que l'extraction de programme ne repose pas exactement sur les mêmes outils théoriques que la formalisation ou la recherche de preuve. La recherche de preuve se situe en effet en amont de la preuve formelle — la preuve est son produit — tandis que l'extraction de programme se situe en aval — la preuve est sa matière première. La première est orientée vers la construction d'une dérivation alors que la seconde s'en éloigne déjà pour aller vers le calcul. Sur le plan théorique, ces deux activités sont basées sur deux façons différentes d'envisager le lien entre un terme et un type : le *typage* (par la déduction) et la *réalisabilité* (par le calcul). Pour illustrer la différence entre ces deux approches, on a récapitulé

[18] Pour être honnête, les restrictions dans les schémas d'élimination de Coq sont tout autant motivées par un souci de cohérence logique, ou du moins, par un souci de compatibilité avec les mathématiques « habituelles ». Voir à ce sujet la discussion dans [18].

dans la Fig. 1.1 la définition du typage et de la réalisabilité pour un même système de types : le système T de Gödel.

Types $\quad A, B \;::=\; \mathsf{nat} \;\mid\; A \to B$

Termes $\quad M, N \;::=\; x \;\mid\; \lambda x\,.\,M \;\mid\; M\,N \;\mid\; \mathsf{0} \;\mid\; \mathsf{s} \;\mid\; \mathsf{rec}$

Réduction

$$\begin{aligned} (\lambda x\,.\,M)\,N &\succ M\{x := N\} \\ \mathsf{rec}\,M_0\,M_1\,\mathsf{0} &\succ M_0 \\ \mathsf{rec}\,M_0\,M_1\,(\mathsf{s}\,N) &\succ M_1\,N\,(\mathsf{rec}\,M_0\,M_1\,N) \end{aligned}$$

Définition du typage (M, N ouverts)

$$\frac{}{\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x\,.\,M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$$

$$\frac{}{\Gamma \vdash \mathsf{0} : \mathsf{nat}} \qquad \frac{}{\Gamma \vdash \mathsf{s} : \mathsf{nat} \to \mathsf{nat}} \qquad \frac{}{\Gamma \vdash \mathsf{rec} : A \to (\mathsf{nat} \to A \to A) \to \mathsf{nat} \to A}$$

Définition de la réalisabilité (M, N clos)

$$\begin{aligned} M \Vdash \mathsf{nat} &\equiv \exists n \in \mathbb{N}\ (M \succ^* \mathsf{s}^n\mathsf{0}) \\ M \Vdash A \to B &\equiv \forall N\ (N \Vdash A \;\Rightarrow\; M\,N \Vdash B) \end{aligned}$$

Lemme d'adéquation — *Si $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$, alors pour tous termes clos $N_1, \ldots, N_n$ tels que $N_1 \Vdash A_1$, …, $N_n \Vdash A_n$ on a $M\{x_1 := N_1; \ldots; x_n := N_n\} \Vdash B$.*

Fig. 1.1 – Typage et réalisabilité dans le système T

La relation de typage (notée $M : A$) procède d'une analyse récursive de la structure du terme M qu'on est en train de « typer ». Chaque construction syntaxique a sa (ou ses) règle(s) de typage, et tout changement de la syntaxe du calcul, si mineur soit-il, nécessite de modifier les règles de typage correspondantes. Afin de pouvoir traverser les λ-abstractions, la relation de typage a besoin de travailler avec des termes ouverts ; c'est là la raison de la présence des contextes de typage, qui associent des types à ces termes indéterminés que sont les variables. Le principal intérêt de la relation de typage $M : A$ réside dans le fait qu'elle admet une justification simple : la dérivation de typage, qui dans les systèmes les plus simples peut même être inférée à partir du terme M et du type A, voire à partir du seul terme M.

Une chose essentielle est pourtant totalement absente de la définition : le calcul, qui pas une seule fois n'est mentionné dans les règles[19]. De fait, le lien

[19] C'est vrai même pour les systèmes de types avec une règle de conversion : la notion de calcul qui intervient dans cette règle (qu'on peut tout aussi bien remplacer par un jugement

entre le typage et le calcul est un lien indirect, qui ne peut être établi qu'en ayant recours à cette autre forme de typage qu'est la réalisabilité[20]

Contrairement au typage, la relation de réalisabilité (notée $M \Vdash A$) est définie par récurrence sur la structure du type A (voir Fig. 1.1). Ici, le terme M n'est observé qu'à travers son comportement calculatoire, qui lui est imposé par le type A. La syntaxe est quasiment traitée comme une boîte noire : il n'est jamais nécessaire de descendre dans le corps des abstractions, et la réalisabilité peut être définie uniquement sur les termes clos.

La réalisabilité a cependant un défaut par rapport au typage : elle ne dispose pas d'une notion de « certificat » similaire aux dérivations de typage. De fait, même pour un exemple aussi simple que le système T, la relation de réalisabilité n'est ni récursive ni même semi-récursive. Pour justifier qu'un terme M réalise un type A, on devra donc utiliser un argument externe, emprunté à la logique « ambiante » ou formalisé dans un système de déduction quelconque[21].

L'opposition entre typage et réalisabilité n'est pas nouvelle en logique, et correspond très exactement à la vieille opposition entre prouvabilité (au sens d'un système formel) et validité (au sens des modèles. Typage et réalisabilité entretiennent d'ailleurs la même relation que prouvabilité et validité : la relation de typage $M : A$ implique la relation de réalisabilité $M \Vdash A$ à travers un *lemme d'adéquation* (Fig. 1.1) dont la réciproque est bien évidemment fausse. Et c'est grâce à ce lemme d'adéquation qu'il est possible d'établir que le système de types est correct vis-à-vis du calcul. (C'est-à-dire dans le cas du système T : que tout terme clos de type nat se réduit effectivement sur un entier naturel.)

Dans la perspective de l'extraction de programme à partir d'une preuve (en logique intuitionniste ou en logique classique), le lemme d'adéquation permet de faire des combinaisons tout à fait inédites entre des termes bien typés et des réalisateurs. Supposons par exemple qu'on dispose d'une preuve formelle M d'un théorème B sous une certaine hypothèse A, c'est-à-dire d'un terme ouvert M tel que $x : A \vdash M : B$. Le lemme d'adéquation nous dit alors que pour tout réalisateur $N \Vdash A$ le terme $M\{x := N\}$ constitue un réalisateur du théorème B. Bien entendu, on peut prendre pour N n'importe quelle preuve de l'hypothèse A (auquel cas le terme $M\{x := N\}$ correspond lui-aussi à une preuve). Mais dans le cas où A est une formule très simple (typiquement : un énoncé Π^0_1, comme par exemple la commutativité de l'addition dans $\mathbb{N}$, le dernier théorème de

d'égalité) est une notion de calcul sur les types qui n'a rien à voir avec le calcul susceptible d'être effectué par le terme qu'on est en train de typer (qui n'est pas affecté par la règle).

[20] Dans la communauté de la programmation fonctionnelle et de la théorie des types, on a coutume d'établir le lien entre le typage et le calcul selon un schéma un peu différent. Dans un premier temps on démontre la propriété de préservation du typage par réduction (*subject reduction*), puis on vérifie que les formes normales closes de type nat correspondent bien aux entiers naturels. Enfin, on prouve un théorème de normalisation forte, qui garantit que tous les calculs terminent indépendamment de la stratégie adoptée. Ce n'est que la combinaison des trois résultats qui permet d'établir que tout programme clos de type nat se réduit effectivement sur un entier naturel. Dans cette démonstration en trois morceaux, la réalisabilité est en réalité cachée dans le modèle de normalisation, qui n'est rien d'autre qu'un modèle de réalisabilité trafiqué pour prendre en compte les différentes stratégies de réduction possibles.

[21] Ici le système naturel pour justifier la relation $M \Vdash A$ serait l'arithmétique de Heyting (ou n'importe quel système plus puissant). Historiquement, la réalisabilité a été définie [54] comme une transformation sur les formules de l'arithmétique de Heyting, et elle est encore très souvent présentée de cette façon. Ici comme dans la suite de ce mémoire, j'adopterai le point de vue de la théorie des modèles, en considérant la réalisabilité comme une sémantique qui n'a pas besoin d'autres outils pour être justifiée que ceux que nous fournit la logique ambiante. (Seule exception : la preuve de normalisation forte « relative » présentée à la section 2.5.6.)

Fermat ou même la conjecture de Goldbach) on aura tout intérêt à choisir pour l'hypothèse A un réalisateur N construit à la main sans qu'il soit nécessaire de formaliser la preuve de A[22] (en supposant même qu'on en connaisse une). Ce qui nous donnera en bout de chaîne un réalisateur de B beaucoup plus efficace que celui qui nous aurait été donné par une simple combinaison des preuves.

Nous aurons l'occasion de revenir plus en détail sur cette opposition entre typage et réalisabilité au chapitre 4, mais dans un cadre — la réalisabilité classique de Krivine — nettement plus subtil que celui du système T.

1.2.3 Curry-Howard en logique classique

On a longtemps pensé que la correspondance preuves-programmes n'avait de sens que dans un cadre intuitionniste[23], et que le contenu calculatoire des preuves classiques ne pouvait être analysé que de manière indirecte, à travers des traductions vers un système intuitionniste : les A-traductions de Friedman, les diverses traductions de la logique classique vers la logique linéaire, ou encore la *no-counter-example interpretation* de Kreisel [55].

Cette idée a été battue en brèche au début des années 1990 quand Felleisen et Griffin [46] ont découvert que l'opérateur de contrôle « *call/cc* » pouvait être typé avec la loi de Peirce $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$, qui implique en logique intuitionniste[24] le principe du tiers-exclu. Cette découverte mettait en évidence un lien profond entre les principes de raisonnement de la logique classique et les *opérateurs de contrôle* en programmation fonctionnelle, c'est-à-dire les opérateurs qui permettent à un programme de manipuler les continuations, et notamment de capturer la continuation courante pour pouvoir la restaurer ultérieurement.

De nombreux λ-calculs classiques ont été proposés à la suite de ce travail pionnier : des calculs déterministes comme le $\lambda\mu$-calcul de Parigot [85], le λ_c-calcul de Krivine [62], le $\lambda\bar{\lambda}\mu\tilde{\mu}$-calcul de Curien et Herbelin [28], ou encore des calculs non déterministes comme le λ-calcul symétrique de Barbanera et Berardi [8] et ses variantes [64]. Il est à noter que tous les λ-calculs classiques déterministes se traduisent dans le λ-calcul pur à l'aide de transformations-CPS appropriées, qui correspondent au niveau des formules à des traductions négatives de la logique classique vers la logique intuitionniste.

C'est dans ce contexte que Jean-Louis Krivine a développé à partir du λ_c-calcul une théorie de la réalisabilité classique [62]. La principale originalité de ce cadre (outre le fait qu'il s'agisse de réalisabilité plutôt que de typage) vient du fait qu'il est capable de traiter non seulement tous les principes de raisonnement classiques, mais aussi certaines formes de l'axiome du choix [61]. Le cadre est par ailleurs suffisamment malléable pour pouvoir être étendu à la théorie des ensembles de Zermelo-Fraenkel [60] et au *forcing* [63].

Cela fait maintenant plus de deux ans que l'essentiel de mon activité de recherche porte sur la théorie de la réalisabilité classique, à laquelle est consacrée le chapitre 4 de ce mémoire. Ce chapitre est organisé autour d'un fil conducteur, qui est la problématique de l'extraction de témoin (existentiel) en logique

[22] En réalisabilité intuitionniste comme en réalisabilité classique, toute formule $\forall^{\mathsf{N}} x\, f(x) = 0$ (i.e. Π^0_1) qui est vraie dans le modèle standard admet pour réalisateur le λ-terme $\lambda nz\,.\,z$. En réalisabilité classique, ce résultat se généralise même à toutes les formules de l'arithmétique du premier ordre [62] (théorème 6 p. 89), avec des réalisateurs plus complexes.

[23] Ou dans le cadre de la logique linéaire, qu'il est possible de voir comme une version « symétrisée » de la logique intuitionniste [43].

[24] En présence du principe du *ex falso quod libet*.

classique — sans doute l'aspect de la réalisabilité classique où celle-ci se démarque le plus nettement de la réalisabilité intuitionniste. Je montrerai aussi comment les principaux résultats de la théorie peuvent être étendus au calcul des constructions avec univers à l'aide d'une extension appropriée du modèle de réalisabilité sous-jacent, avant d'expliquer comment ces techniques sont mises en œuvre dans le module d'extraction classique de Coq.

Je conclurai ce mémoire (au chapitre 5) en discutant des perspectives ouvertes par la théorie de la réalisabilité classique, tant sur le plan des applications pratiques que sur le plan théorique, en présentant des développements plus récents qui suggèrent la possibilité d'une extension de la correspondance preuves-programmes aux effets de bord en lien avec la théorie du *forcing*.

Chapitre 2

Types et ensembles

Ce chapitre est consacré à quatre résultats de cohérence relative entre des théories des ensembles et des systèmes de types à l'aide de traductions basées sur le paradigme « ensemble = graphe pointé » introduit dans [35, 1] et [74]. Le premier résultat [75] (section 2.2) est la définition d'un premier système de types permettant de traduire toute la théorie des ensembles de Zermelo-Fraenkel intuitionniste, et d'obtenir ainsi un résultat de normalisation forte. Le second résultat [76] (section 2.3) est la définition d'un système de types purs à quatre sortes qui capture très exactement la force de la théorie des ensembles de Zermelo. Le troisième résultat [31] (section 2.4), issu d'un travail en commun avec Gilles Dowek, est une variante du résultat précédent dans le cadre de la déduction modulo. Enfin, le dernier résultat (section 2.5) effectue une synthèse des travaux précédents pour définir un système de types qui capture exactement la force de la théorie des ensembles de Zermelo-Fraenkel.

2.1 Ensembles et graphes pointés

2.1.1 Graphes pointés et antifondation

Historiquement, la représentation des ensembles sous forme de graphes pointés a été introduite en théorie des ensembles par M. Forti et F. Honsell [35] puis par P. Aczel [1][1] pour interpréter l'axiome d'antifondation. Cette technique consiste à représenter la structure récursive d'un ensemble (c'est-à-dire : lui-même, ses éléments, les éléments de ses éléments, etc.) sous la forme d'un *graphe pointé*, c'est-à-dire un triplet (X, A, a) où :

– X est ensemble quelconque — le *support* — dont les éléments constituent les *sommets* du graphe pointé ;

[1] J'ai moi-même redécouvert ces idées de manière indépendante en classe de mathématiques supérieures (1990–1991) et ai rédigé un mémoire d'une cinquantaine de pages intitulé *Théorie axiomatique des ensembles abyssaux.* (L'axiome d'antifondation y est formulé de manière identique sous le nom d'*axiome de la profondeur infinie.*) Ce mémoire contient également une construction de modèle (à vrai dire très maladroite) basée sur le paradigme « ensemble = graphe pointé » et sur la notion de *modèle intérieur* décrite dans la première édition de [59] (datant de 1971). Cet ouvrage m'avait été prêté par mon professeur de mathématiques de l'époque, Pierre Delezoide, qui m'avait également suggéré de considérer les problèmes de non-fondation dans les preuves en calcul des séquents — une notion que j'ignorais totalement.

– A est une relation binaire sur X — la *relation d'adjacence* — qui décrit la structure du graphe sous-jacent ;
– $a \in X$ est un sommet distingué — la *racine* — qui constitue intuitivement le « point d'entrée » dans le graphe pointé et détermine sa lecture sous forme d'ensemble.

Le lien entre un graphe pointé donné et le ou les ensembles qu'il est censé représenter est défini au moyen des notions de *décoration* et de *réification*. Formellement, une décoration d'un graphe pointé (X, A, a) est une fonction ϕ de domaine X (à valeurs dans les ensembles) telle que pour tout $x \in X$ on a

$$\phi(x) \;=\; \{\phi(x') \;:\; x' \in X \;\text{ et }\; x'\,A\,x\}$$

(Cette notion ne dépend en fait que du graphe (X, A) sous-jacent.) On dit alors qu'un ensemble E est une *réification* du graphe pointé (X, A, a) s'il existe une décoration ϕ du graphe sous-jacent (X, A) telle que $\phi(a) = E$.

Tout ensemble E est la réification d'un graphe pointé : il suffit pour cela de considérer le graphe pointé dont le support est la clôture transitive [59] du singleton $\{E\}$, dont la relation d'adjacence est la relation d'appartenance sur cet ensemble, et dont la racine est le sommet E lui-même. Mais, réciproquement, tout graphe pointé admet-il une réification ? Si oui, qu'en est-il de l'unicité ?

Il est facile de démontrer avec les axiomes de Zermelo-Fraenkel que tout graphe pointé (X, A, a) dont la relation d'adjacence A est bien fondée admet une et une seule décoration ϕ (construite par induction sur A) et, par suite, une et une seule réification. Dans le cas où la relation A n'est pas bien fondée sur X, l'existence et l'unicité d'une réification du graphe pointé (X, A, a) sont toutes les deux indécidables dans ZF[2].

Si on suppose l'axiome de la fondation (qui exprime que la relation d'appartenance est bien fondée sur l'univers ensembliste), il est facile de démontrer qu'un graphe pointé (X, A, a) dont la relation d'adjacence A n'est pas bien fondée (sur X) n'admet aucune décoration — et *a fortiori* aucune réification.

L'idée de Forti, Honsell et Aczel est de réfuter l'axiome de la fondation en le remplaçant par un axiome d'*antifondation* (AFA) dont l'énoncé est le suivant :

(AFA) *Tout graphe pointé admet une et une seule réification.*

L'intérêt de l'axiome d'antifondation est double. D'abord il permet de construire des ensembles « mal fondés », comme par exemple un ensemble x qui est son unique élément (i.e. $x = \{x\}$), simplement en réifiant le graphe pointé à un sommet (la racine) et à un arc reliant la racine à elle-même. Mais surtout, l'axiome d'antifondation permet de démontrer des égalités entre deux ensembles mal-fondés. Par exemple, si x et y sont tels que $x = \{x\}$ et $y = \{y\}$, alors $x = y$, puisque x et y sont des réifications du même graphe pointé.

Mieux encore : considérons à présent deux ensembles u et v tels que $u = \{v\}$ et $v = \{u\}$. Il est clair que u est la réification d'un graphe pointé à deux sommets et deux arcs (reliant chaque sommet à l'autre), à travers une décoration associant u à la racine et v à l'autre sommet. Mais pour des raisons de symétrie évidentes, l'ensemble v est aussi une réification du même graphe pointé... d'où il ressort (par unicité) que $u = v = \{u\} = \{v\}$, et finalement : $u = v = x = y$.

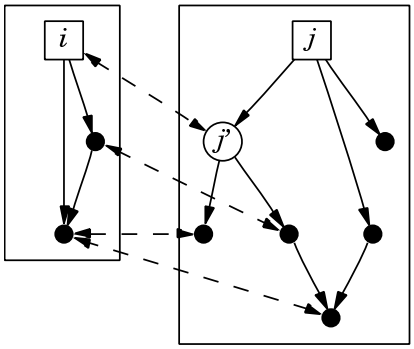
[2]En adoptant la convention que l'axiome de la fondation (ou *axiome de régularité*) ne fait pas partie des axiomes de ZF (suivant [59]).

Le modèle Pour montrer la cohérence relative de ZF + AFA par rapport à ZF, les auteurs construisent un modèle dans lequel les ensembles sont interprétés précisément par des graphes pointés (pris dans un modèle quelconque de ZF). Dans ce cadre, la relation d'égalité (extensionnelle) entre deux ensembles est interprétée par la relation de bissimilarité entre les graphes pointés (X, A, a) et (Y, B, b) correspondants, c'est-à-dire comme l'existence d'une relation binaire $R \subseteq X \times Y$ (qu'on appelle une *bissimulation*) telle que :

1. *Simulation de* (X, A, a) *dans* (Y, B, b) : Pour tous $x, x' \in X$ et $y \in Y$ tels que $x' \, A \, x$ et $x \, R \, y$, il existe $y' \in Y$ tel que $y' \, B \, y$ et $x' \, R \, y'$;
2. *Simulation de* (Y, B, b) *dans* (Y, A, a) *:* Pour tous $y, y' \in Y$ et $x \in X$ tels que $y' \, B \, y$ et $x \, R \, y$, il existe $x' \in X$ tel que $x' \, A \, x$ et $x' \, R \, y'$;
3. *Connection des racines :* $a \, R \, b$.

La relation d'appartenance entre deux ensembles représentés par des graphes pointés (X, A, a) et (Y, B, b) est quant à elle interprétée par l'existence d'un sommet $b' \in Y$ tel que $b' \, B \, b$ et tel que les graphes pointés (X, A, a) et (Y, B, b') sont bissimilaires. (Ce que dans la suite de ce chapitre j'appellerai la relation de *bissimilarité décalée*.)

À titre d'exemple, on trouvera ci-dessous deux graphes pointés (figurés chacun dans un rectangle) qui représentent de gauche à droite les entiers de von Neumann $2 = \{\varnothing; \{\varnothing\}\}$ et $3 = \{\varnothing; \{\varnothing\}; \{\varnothing; \{\varnothing\}\}\}$ avec la bissimulation décalée (en tireté) exprimant l'appartenance du premier ensemble au second :



(La relation d'adjacence est représentée par des flèches qui pointent des parents vers les fils, et les racines sont représentées par des carrés.)

2.1.2 Graphes pointés en théorie des types

Au début de ma dernière année de thèse, il m'est clairement apparu que toute la construction de Forti, Honsell et Aczel pouvait être reformulée dans de nombreux systèmes de types, offrant ainsi une nouvelle façon de traduire la théorie des ensembles en théorie des types.

En théorie des types, un graphe pointé est constitué d'un type X (dont la sorte reste à préciser), d'un terme A de type $X \to X \to *$ (où $*$ désigne la sorte des propositions), et d'un terme a de type X. On notera qu'ici, le *triplet* constitué par ces trois objets n'a pas besoin d'être exprimé dans la théorie cible : on peut tout aussi bien travailler avec ses trois composantes séparément du moment qu'on ne les mélange pas avec les composantes qui participent à la représentation d'autres ensembles.

Mise en œuvre dans un PTS Pour mettre en œuvre une telle construction dans un système de types basé sur un PTS $(\mathscr{S}, \mathscr{A}, \mathscr{R})$ [11], il faut faire un certain nombre d'hypothèses sur la structure du PTS sous-jacent. On doit d'abord supposer qu'il contient (au moins) les sortes et axiomes suivants :

1. Une sorte $* \in \mathscr{S}$ des *propositions* ;
2. Une sorte $*' \in \mathscr{S}$ (type de $*$) avec l'axiome $(*, *') \in \mathscr{A}$;
3. Une sorte $\Box \in \mathscr{S}$ des *supports de graphes pointés* ;
4. Une sorte $\Box' \in \mathscr{S}$ (type de $\Box$) avec l'axiome $(\Box, \Box') \in \mathscr{A}$.

On doit également supposer que le PTS sous-jacent contient les règles suivantes :

5. Deux règles $(*, *, *) \in \mathscr{R}$ et $(*', *, *) \in \mathscr{R}$ pour former respectivement l'implication logique et la quantification universelle sur toutes les propositions. Cette seconde règle permet notamment de définir les codages au second ordre des connecteurs logiques $\neg$, $\wedge$ et $\vee$. (On notera que ces deux règles définissent très exactement le système F, qui est imprédicatif.)
6. Deux règles $(\Box, *, *) \in \mathscr{R}$ et $(\Box', *, *) \in \mathscr{R}$ pour former respectivement la quantification universelle sur les sommets d'un graphe pointé et la quantification universelle sur tous les supports de graphes pointés. Combinées avec les deux règles précédentes, ces deux règles permettent également de définir au second ordre les quantifications existentielles correspondantes.
7. Deux règles $(\Box, \Box, \Box) \in \mathscr{R}$ et $(\Box, *', \Box) \in \mathscr{R}$ pour former les supports de graphes pointés correspondant aux constructions usuelles de la théorie des ensembles telles que la paire ordonnée, l'union, les ensembles définis par compréhension, ainsi que les ensembles de parties. Par exemple, la paire formée par deux ensembles représentés par des graphes pointés (X, A, a) et (Y, B, b) peut elle-même être représentée par un graphe pointé dont le support est le type $(X \to *) \to (Y \to *) \to *$ [74], un type dont la formation repose sur la présence simultanée des deux règles.

Enfin, il est nécessaire de supposer que

8. La sorte $\Box$ est habitée par un type non vide (i.e. un type habité par un terme clos), afin de pouvoir définir au moins une structure de graphe pointé. Même si elle est très souvent réalisée, cette dernière condition n'est pas automatiquement impliquée par les conditions précédentes.

De très nombreux PTS satisfont les conditions 1 à 8 données ci-dessus. Tout d'abord les systèmes incohérents, tels que le système $* : *$ de Martin-Löf ou le système U de Girard, qui permettent à travers la représentation par des graphes pointés d'interpréter la théorie des ensembles naïve (et incohérente) introduite par Cantor et formalisée par Frege. (Une construction similaire est possible dans le système U^-, mais il faut aménager un peu la traduction [74].) Du côté des systèmes cohérents (ou, du moins, supposés tels), on peut citer le système $F\omega.2$ [74, 76] (en posant $*' = \Box = \Box_1$ et $\Box' = \Box_2$) et bien entendu tous les sur-systèmes (tels que le calcul des constructions avec univers par exemple).

Définition de la traduction Formellement, la traduction des formules de la théorie des ensembles dans un système de types satisfaisant les conditions 1 à 8 données ci-dessus est définie de la manière suivante. À chaque variable x de la théorie des ensembles on associe trois variables de la théorie des types $\bar{x} : \Box$, $\tilde{x} : \bar{x} \to \bar{x} \to *$ et $\dot{x} : \bar{x}$. (Ces trois variables viennent donc avec le morceau

de contexte correspondant, où les types des variables $\tilde{x}$ et $\dot{x}$ dépendent de la variable $\bar{x}$.) Chaque formule ϕ de la théorie des ensembles est traduite en un type $\phi^* : *$ (dans le contexte induit par ses variables libres) en posant :

$$\begin{array}{rcl}
(x = y)^* & \equiv & (\bar{x}, \tilde{x}, \dot{x}) \approx (\bar{y}, \tilde{y}, \dot{y}) \\
(x \in y)^* & \equiv & \exists z : \bar{y} \,.\, (\tilde{y}\, z\, \dot{y} \wedge (\bar{x}, \tilde{x}, \dot{x}) \approx (\bar{y}, \tilde{y}, z)) \\
(\forall x\, \phi)^* & \equiv & \forall \bar{x} : \Box \,.\, \forall \tilde{x} : (\bar{x} \to \bar{x} \to *) \,.\, \forall \dot{x} : \bar{x} \,.\, \phi^* \\
(\exists x\, \phi)^* & \equiv & \exists \bar{x} : \Box \,.\, \exists \tilde{x} : (\bar{x} \to \bar{x} \to *) \,.\, \exists \dot{x} : \bar{x} \,.\, \phi^* \\
(\neg \phi)^* & \equiv & \neg \phi^* \\
(\phi \Rightarrow \psi)^* & \equiv & \phi^* \Rightarrow \psi^* \\
(\phi \wedge \psi)^* & \equiv & \phi^* \wedge \psi^* \\
(\phi \vee \psi)^* & \equiv & \phi^* \vee \psi^*
\end{array}$$

où la proposition $(\bar{x}, \tilde{x}, x_0) \approx (\bar{y}, \tilde{y}, y_0)$ est une abréviation pour

$$\begin{array}{l}
\exists R : (\bar{x} \to \bar{y} \to *) \,. \\
\quad (\forall \alpha, \alpha' : \bar{x} \,.\, \forall \beta : \bar{y} \,.\, \tilde{x}\, \alpha'\, \alpha \Rightarrow R\, \alpha\, \beta \Rightarrow \exists \beta' : \bar{y} \,.\, \tilde{y}\, \beta'\, \beta \wedge R\, \alpha'\, \beta') \quad \wedge \\
\quad (\forall \beta, \beta' : \bar{y} \,.\, \forall \alpha : \bar{x} \,.\, \tilde{y}\, \beta'\, \beta \Rightarrow R\, \alpha\, \beta \Rightarrow \exists \alpha' : \bar{x} \,.\, \tilde{x}\, \alpha'\, \alpha \wedge R\, \alpha'\, \beta') \quad \wedge \\
\quad R\, x_0\, y_0
\end{array}$$

et où les connecteurs $\neg$, $\wedge$, $\vee$ et les différentes formes du quantificateur existentiel sont définis en théorie des types par les codages habituels au second ordre.

On démontre alors le résultat suivant :

Proposition 1 (Correction) — *Si le système de types cible vérifie les conditions 1 à 8, alors tous les axiomes « finitaires » de la théorie des ensembles de Zermelo sont prouvables (i.e. admettent des termes de preuves clos) à travers la traduction $\phi \mapsto \phi^*$, à savoir : les axiomes d'égalité (réflexivité, symétrie, transitivité, compatibilité à gauche et à droite de la relation d'appartenance vis-à-vis de l'égalité), l'axiome d'extensionnalité, l'axiome de la paire, toutes les instances du schéma de compréhension, l'axiome de la réunion et l'axiome des parties. Sont prouvables également (à travers la traduction) l'axiome de la clôture transitive (TC)*[3] *et l'axiome d'antifondation (AFA).*

Preuve. On utilise les mêmes constructions que celles qui sont décrites dans [74] (pour les axiomes de Zermelo finitaires) et dans [76] (pour TC et AFA). □

L'axiome de l'infini Les conditions 1 à 8 ne suffisent pas à dériver l'axiome de l'infini à travers la traduction $\phi \mapsto \phi^*$. Pour cela, il faut que le système de types cible permette de construire un ensemble prouvablement infini dans la sorte $\Box$, à partir duquel on peut reconstruire un graphe pointé représentant l'ensemble des entiers de von Neumann.

Il y a essentiellement deux façons de procéder.

La première consiste à supposer que le système de types contient un type des entiers primitifs dans la sorte $\Box$ (i.e. satisfaisant les axiomes de Peano). C'est le choix qui est retenu dans [75], ce qui explique pourquoi le système considéré

[3] Rappelons que l'axiome de la clôture transitive exprime simplement que « tout ensemble admet une clôture transitive ». Cet axiome est inutile dans ZF puisque l'existence de la clôture transitive est prouvable à l'aide du remplacement. En revanche, il n'est pas démontrable dans la théorie de Zermelo (il existe des modèles de Z qui ne le satisfont pas [59]). On peut toutefois souligner que les théories Z et Z + TC + AFA demeurent équiconsistantes [33].

($F\omega.2^{++}$) n'a que deux univers prédicatifs. Cette solution est très simple, et son seul inconvénient est de sortir du cadre des PTS.

La deuxième façon de procéder consiste à introduire une nouvelle sorte $\Box_0$ au-dessous de la sorte $\Box$, avec l'axiome $(\Box_0 : \Box) \in \mathscr{A}$ et les règles de formation de produit dépendant $(\Box_0, \Box_0, \Box_0) \in \mathscr{R}$, $(\Box, \Box_0, \Box) \in \mathscr{R}$ et $(\Box_0, *, *) \in \mathscr{R}$. Grâce à l'introduction de cette sorte supplémentaire, il est possible de définir dans la sorte $\Box$ un type *prédicatif* des entiers de Church

$$\mathsf{Nat} \quad \equiv \quad \Pi X : \Box_0 \,.\, (X \to (X \to X) \to X) \quad : \quad \Box$$

dont j'ai démontré dans ma thèse qu'il vient avec suffisamment de structure pour dériver les axiomes de Peano. C'est cette deuxième solution qui est retenue dans [76] (afin de ne pas sortir du cadre des PTS), ce qui explique la présence de trois sortes prédicatives dans le système présenté (λZ) au lieu de deux.

On vérifie alors que :

Proposition 2 (Axiome de l'infini) — *Si le système de types cible vérifie les conditions 1 à 8 et si la sorte $\Box$ contient un type des entiers naturels, primitif ou défini à l'aide d'une sorte supplémentaire, alors l'axiome de l'infini est prouvable à travers la traduction $\phi \mapsto \phi^*$.*

2.2 Un théorème de normalisation forte pour IZF

Il s'agit à présent de définir un système de types dans lequel il est possible de dériver non seulement tous les axiomes de Zermelo (à travers la traduction $\phi \mapsto \phi^*$), mais aussi toutes les instances du schéma de remplacement de la théorie de Zermelo-Fraenkel. Bien entendu, l'intérêt d'une telle construction est de conférer aux preuves de la théorie des ensembles de Zermelo-Fraenkel (du moins son fragment intuitionniste) une interprétation calculatoire dans le cadre de la correspondance de Curry-Howard. Si de plus les termes de preuves du système de types cible sont fortement normalisables, on obtient alors un théorème de normalisation forte pour IZF (toujours à travers la traduction).

Au moment où j'ai établi ce résultat (à la fin de l'année 2002), il n'y avait à ma connaissance parmi toutes les théories des types disponibles alors — du moins parmi celles qui satisfont la propriété de normalisation forte — aucune dans laquelle on sache plonger le fragment intuitionniste de la théorie de Zermelo-Fraenkel tout entier (sans ajouter d'axiome non calculatoire). Aujourd'hui encore, on ne sait toujours pas comment la plus puissante d'entre elles — le calcul des constructions inductives de Coq — se compare vis-à-vis de la théorie de Zermelo-Fraenkel en termes de force théorique.[4] Ce résultat pionnier a toutefois un prix, qui est que le système de types utilisé, le système $F\omega.2^{++}$, a été conçu dans le seul but de « faire passer » le schéma de remplacement et est franchement inesthétique[5]. Ce système dispose en outre de mauvaises propriétés sur lesquelles j'aurai l'occasion de revenir par la suite.

Cependant, mon travail n'est pas le premier à avoir défini une correspondance de Curry-Howard pour ZF, puisqu'il avait été précédé quelques années auparavant par le modèle de réalisabilité classique de Krivine pour la théorie

[4] Mes diverses tentatives pour résoudre ce problème ont échoué jusqu'ici.
[5] Le nom $F\omega.2^{++}$ est censé refléter le caractère *ad hoc* du système.

de Zermelo-Fraenkel [60], dans lequel on trouve également un système de types pour ZF (classique) — même si ce dernier ne satisfait pas la propriété de normalisation forte, qui n'est d'ailleurs pas très utile dans le cadre de la réalisabilité classique (j'aurai l'occasion de revenir sur ce point au chapitre 4).

Depuis, Moczydłowski [81] a proposé une autre approche, qui se situe davantage dans la lignée des travaux de Myhill [82], Friedman [37] et McCarthy [70] autour de la réalisabilité (à la Kleene) en théorie des ensembles intuitionniste. D'un point de vue technique, l'auteur explicite le modèle de réalisabilité décrit dans [70] en remplaçant les entiers de Gödel par un vrai calcul de λ-termes intuitionnistes, avec des constructeurs et des destructeurs pour chaque axiome, dans un esprit très proche de la théorie de Martin-Löf.[6] Ce modèle de réalisabilité permet alors de justifier un système de types correspondant très précisément à IZF_R (un fragment intuitionniste de la théorie de Zermelo-Fraenkel). De fait, la méthodologie adoptée se rapproche très sensiblement de celle de Krivine, tout en restant sur un terrain purement intuitionniste.

Le système de Moczydłowski est sur de nombreux points bien plus satisfaisant que celui que je vais présenter ici. Il a cependant deux défauts. Le premier, plutôt mineur, est que son modèle de réalisabilité ne permet de démontrer qu'un résultat de normalisation faible, et non pas un résultat de normalisation forte comme je le fais dans [75]. Le second, sans doute plus significatif d'un point de vue logique, est que son système n'est en mesure de traiter que le schéma de remplacement intuitionniste. Or depuis [38] on sait que le schéma de remplacement intuitionniste est plus faible que le schéma de remplacement classique. Pour pouvoir bénéficier à travers la traduction négative de Gödel-Friedman [36] de toute la force du schéma de remplacement classique, il faut se donner dans le cadre intuitionniste un schéma plus fort : le schéma de collection.

2.2.1 Les différentes formes de remplacement

Pour saisir les subtilités d'une interprétation calculatoire des preuves de ZF en logique intuitionniste, il est important de bien distinguer les deux formulations du schéma de remplacement (c'est-à-dire : le schéma de remplacement et le schéma de collection) ainsi que les différences entre ces deux formulations, lesquelles sont surtout sensibles en logique intuitionniste, mais aussi à travers la traduction négative des preuves de ZF classique [36, 38].

Le *schéma de remplacement* qui permet de passer de la théorie de Zermelo à la théorie de Zermelo-Fraenkel [59] est constitué des axiomes de la forme

$$\text{(Remplacement)} \qquad \forall a\ \big(\forall x \in a\ \exists! y\ \phi(x,y)\ \Rightarrow\ \exists b\ \forall x \in a\ \exists y \in b\ \phi(x,y)\big)$$

où $\phi(x,y)$ est n'importe quelle formule dans laquelle la variable b n'a pas d'occurrence libre. (Comme d'habitude, ce n'est pas la formule ci-dessus qu'on pose en axiome, mais sa clôture universelle par rapport à toutes les variables libres de la formule ϕ autres que a, x et y.)

Intuitivement, l'axiome ci-dessus exprime que si une relation binaire $\phi(x,y)$ est fonctionnelle par rapport à son premier argument sur un ensemble a donné, alors il existe un ensemble b constitué par toutes les images des éléments de cet ensemble à travers la relation ϕ. D'après la contrainte d'unicité de y (dans

[6] À cette différence près que le système de types proposé par Moczydłowski [81] est évidemment très fortement imprédicatif.

la prémisse), chaque élément de a a une et une seule image dans b par ϕ. On pourrait donc remplacer dans la conclusion le « $\exists y \in b$ » par un « $\exists! y \in b$ » sans changer pour autant le sens de l'énoncé.

On peut cependant renforcer le schéma de remplacement en supprimant la contrainte d'unicité dans la prémisse de l'implication. On obtient alors le *schéma de collection* dont l'énoncé est :

$$\text{(Collection)} \qquad \forall a \ \big(\forall x \in a \ \exists y \ \phi(x,y) \ \Rightarrow \ \exists b \ \forall x \in a \ \exists y \in b \ \phi(x,y)\big)$$

On notera que sous cette formulation, un même élément de a peut avoir une ou plusieurs images dans b par ϕ. Ceci dit, dans le cas où l'image par ϕ d'un élément de a est une classe propre, le schéma devra bien effectuer un « choix » parmi les images disponibles de façon à ce que les images retenues dans b forment un ensemble. Le schéma de collection introduit donc une certaine forme de choix, dont la forme est assez différente de celle de l'axiome du choix usuel (AC) [59].

Le schéma de collection entraîne clairement le schéma de remplacement en logique intuitionniste et *a fortiori* en logique classique.

En logique classique et en présence de l'axiome de la fondation, le schéma de remplacement implique à son tour le schéma de collection, et les deux formulations sont équivalentes. Pour démontrer l'axiome de collection pour un ensemble a et une formule $\phi(x,y)$ donnée (pas nécessairement fonctionnelle en son premier argument) à partir du schéma de remplacement et de l'axiome de la fondation, il suffit d'appliquer le schéma de remplacement à la formule

$$\phi'(x,z) \quad \equiv \quad z \text{ est l'ensemble des } y \text{ de rang minimal tels que } \phi(x,y).$$

(L'existence et l'unicité de z en fonction de x découle de la notion de rang [59], qui n'a de sens qu'en présence de l'axiome de la fondation.) Si b' est l'image de l'ensemble a par la relation fonctionnelle $\phi'(x,z)$, alors l'ensemble $b = \bigcup b'$ satisfait la conclusion du schéma de collection avec la formule $\phi(x,y)$. (Cette preuve « par rang minimum » repose fortement sur l'utilisation du tiers-exclu.)

Même si dans l'absolu le remplacement n'est pas équivalent à la collection, les deux systèmes classiques qu'on obtient sont équiconsistants. Pour cela il suffit de montrer que la théorie des ensembles — formulée avec le schéma de remplacement ou le schéma de collection — est équiconsistante avec la même théorie étendue avec l'axiome de la fondation. La preuve (par relativisation à la classe propre V) est standard quand il s'agit du schéma de remplacement [59], et s'adapte sans difficulté au cas où on utilise le schéma de collection.

La situation est très différente en logique intuitionniste car les deux schémas ne sont plus équivalents, même en présence de l'axiome de la fondation.[7] On doit donc distinguer deux versions intuitionnistes de la théorie des ensembles de Zermelo-Fraenkel : le système IZF_R (avec schéma de remplacement) et le système IZF_C (avec schéma de collection), sachant qu'on a trivialement l'inclusion $\mathrm{IZF}_R \subset \mathrm{IZF}_C$. Friedman et Ščedrov ont démontré [38] que ces deux formalismes intuitionnistes ne satisfont pas les mêmes formes de la propriété du témoin (cf section 2.2.6) et en déduisent une formule qui est prouvable de manière intuitionniste avec le schéma de collection, mais pas avec le schéma de remplacement (sous réserve que ce dernier est cohérent). L'inclusion entre les

[7] Rappelons qu'en logique intuitionniste, l'axiome de la fondation doit être reformulé sous la forme d'un principe d'induction nœthérienne sur la relation d'appartenance. Cependant, cette formulation demeure classiquement équivalente à la formulation usuelle [59].

deux formalismes IZF_R et IZF_C est donc stricte.[8] De plus, la classe des fonctions récursives prouvablement totales dans IZF_R est strictement plus petite que la classe des fonctions récursives prouvablement totales dans IZF_C [38].

Par ailleurs, Friedman a également démontré [36] qu'à travers la traduction négative de Gödel, les axiomes de ZF ne « tombent » pas dans IZF_R, mais dans IZF_C. (La difficulté est liée à des problèmes profonds liés à l'axiome d'extensionnalité). Par conséquent, le système ZF classique est équiconsistant à IZF_C et constitue même une extension conservative de ce dernier sur la classe des énoncés Π^0_2. (Les deux systèmes permettent donc de prouver la totalité des mêmes fonctions récursives.) Pour toutes ces raisons, c'est donc le système IZF_C qu'il convient de considérer comme la « bonne » version intuitionniste de la théorie des ensembles de Zermelo-Fraenkel.[9]

2.2.2 Du système $F\omega.2$ au système $F\omega.2^+$

Le système $F\omega.2^{++}$ est construit sur le PTS $F\omega.2$ [74] dont les sortes, les axiomes et les règles sont donnés par

$$\begin{array}{rcl} \mathscr{S} & = & \{*;\ \Box_1;\ \Box_2\} \\ \mathscr{A} & = & \{(* : \Box_1);\ (\Box_1 : \Box_2)\} \\ \mathscr{R} & = & \{(*,*,*);\ (\Box_i,*,*)\ :\ i \in \{1;2\}\} \quad \cup \\ & & \{(\Box_i,\Box_j,\Box_{\max(i,j)})\ :\ i \in \{1;2\}\} \end{array}$$

Comme le système $F\omega.2$ est un PTS non-dépendant [25], on peut stratifier sa syntaxe en distinguant d'une part les *termes-objets* (notation : M, N, T, U, etc.), c'est-à-dire les termes dont le type est de sorte $\Box_1$ ou $\Box_2$, et d'autre part les *termes de preuves* (notation : t, u, etc.), c'est-à-dire les termes dont le type est une *proposition* $A : *$ (notation : A, B, etc.)

Toujours pour la même raison, on peut scinder les contextes de typage en deux morceaux, suivant la sorte de la variable déclarée. On distingue ainsi les *signatures* (notation : Σ, etc.) destinées à déclarer les variables d'objets (notation : x, y, z, etc.) des *contextes logiques* (notation : Γ, etc.) destinés à déclarer les variables de preuve (notation : ξ, ξ', etc.) On notera que les propositions qui figurent dans les contextes logiques peuvent dépendre des variables déclarées dans les signatures (mais pas l'inverse), et c'est pourquoi le contexte complet s'écrit $\Sigma;\Gamma$ dans cet ordre.

La syntaxe stratifiée du système $F\omega.2^+$ est rappelée dans la Fig. 2.1.

Termes de preuves à la Curry Pour rendre plus claire la structure des termes de preuves, on supprime toutes les annotations de type (et de termes-objets) dans les termes de preuves, et on passe à une présentation à la Curry. À

[8] En réalité, le résultat de Friedman et Ščedrov ne porte pas sur les systèmes IZF_R et IZF_C tels que nous les avons définis ici, mais sur les systèmes $\mathrm{IZF}_R + \mathrm{Ind}$ et $\mathrm{IZF}_C + \mathrm{Ind}$, où Ind désigne la forme intuitionniste de l'axiome de la fondation. Ceci dit, il est très facile d'en déduire le résultat correspondant pour les systèmes IZF_R et IZF_C, simplement en relativisant la formule incriminée à la collection V des ensembles bien fondés. (Cette collection se définit très bien en logique intuitionniste, voir par exemple [70].)

[9] Cependant, on ne sait pas dans quelle mesure IZF_R est moins fort que IZF_C, et il se pourrait très bien que les deux systèmes fussent équiconsistants. La question demeure ouverte aujourd'hui.

Termes-objets	M, N, T, U, A, B	$::=$	$* \mid \Box_1 \mid \Box_2$
			$\mid x \mid \lambda x : T \,.\, M \mid M\,N$
			$\mid \Pi x : T \,.\, U \mid A \Rightarrow B$
			$\mid \forall x : T \,.\, A \mid \exists x : T \,.\, A$
Termes de preuves	t, u	$::=$	$\xi \mid \lambda \xi \,.\, t \mid tu$
Signatures	Σ	$::=$	$x_1 : T_1, \ldots, x_n : T_n$
Contextes logiques	Γ	$::=$	$\xi_1 : A_1, \ldots, \xi_m : A_m$

FIG. 2.1 – Syntaxe du système $F\omega.2^+$

ce stade, la syntaxe des termes de preuves devient identique à celle du λ-calcul pur. (Mais les systèmes de types sous-jacents restent équivalents.)

Le passage d'un système de preuves à la Church à un système de preuves à la Curry opère un changement de nature sur la quantification universelle $\forall x : T \,.\, A$, qui passe d'un statut de produit dépendant (au sens de Martin-Löf) à un statut de type intersection infinitaire. On tire alors parti de cette transformation pour ajouter une construction duale, à savoir une *quantification existentielle primitive* noté $\exists x : T \,.\, A$, qu'on interprétera comme une union infinitaire. (On notera que cette nouvelle construction propositionnelle au niveau des termes-objets n'amène aucune nouvelle construction au niveau des termes de preuve, où elle est traitée de manière purement locative [42].)

Si le quantificateur existentiel primitif ne pose pas de problème particulier vis-à-vis de la normalisation (on l'interprétera par une union d'ensembles saturés au sens de Tait), la formulation de sa règle d'élimination est plus délicate que pour la quantification universelle. On la formulera ici sous la forme de la règle de typage

$$\frac{\vdash t : \forall x : T \,.\, (A \Rightarrow B)}{\vdash t : (\exists x : T \,.\, A) \Rightarrow B}\ {\scriptstyle x \notin FV(B)}$$

Il serait sans doute plus naturel d'introduire cette règle sous la forme d'une règle de sous-typage

$$\forall x : T \,.\, (A \Rightarrow B) \quad \leq \quad (\exists x : T \,.\, A) \Rightarrow B \qquad\qquad (x \notin FV(B))$$

mais cela nécessiterait d'étendre le formalisme avec un jugement de sous-typage qui complexifierait notablement la présentation du système sans ajouter quoi que ce soit d'essentiel à son expressivité. On notera cependant que ce type de règle *ad hoc* fait en général mauvais ménage avec la propriété de *subject-reduction* pour les termes de preuves. De fait, le système $F\omega.2^{++}$ que nous allons présenter à la fin de cette section semble ne pas vérifier pas la propriété de *subject-reduction* (tout au moins au niveau des termes de preuve), et nous verrons qu'il y a sans doute des bonnes raisons à cela (cf sous-section 2.2.6).

Il faut également souligner que la quantification existentielle primitive est logiquement équivalente à la quantification existentielle codée au second ordre ; son intérêt réside avant tout dans le fait qu'elle permet de clarifier (en la simplifiant) la structure des termes de preuves, ce qui jouera un rôle non négligeable dans la mise en place de l'opérateur ε de Hilbert intuitionniste (cf section 2.2.5).

Jugements et règles de typage Formellement, le système de types $F\omega.2^+$ est structuré autour de quatre formes de jugement :

1. Un jugement de bonne formation de signature, noté $\vdash \Sigma$ sig
   (« Σ est une signature bien formée »)
2. Un jugement de typage sur les termes objets, noté $\Sigma \vdash M : T$
   (« Dans la signature Σ, le terme-objet M a pour type T ») ;
3. Un jugement de bonne formation de contexte logique $\Sigma \vdash \Gamma$ ctx
   (« Dans la signature Σ, le contexte logique Γ est bien formé ») ;
4. Un jugement de typage des termes de preuves, noté $\langle\Sigma\rangle\ \Gamma \vdash t : A$
   (« Dans la signature Σ et le contexte Γ, t est une preuve de A »).

Les règles d'inférence correspondantes sont données à la Fig. 2.2.

2.2.3 Interprétation de la théorie de Zermelo

L'interprétation de la théorie des ensembles de Zermelo s'effectue suivant la méthode de traduction décrite à la section 2.1.2, en basant tous les graphes pointés sur la sorte $\Box_1$ (la sorte $\Box_2$ ne servant qu'à typer $\Box_1$). Pour dériver (à travers la traduction $\phi \mapsto \phi^*$) l'axiome de l'infini, on ajoute au système $F\omega.2^+$ les constantes suivantes

$$\begin{array}{rcl} \mathsf{Nat} & : & \Box_1 \\ \mathsf{0} & : & \mathsf{Nat} \\ \mathsf{s} & : & \mathsf{Nat} \to \mathsf{Nat} \end{array} \qquad \begin{array}{rcl} \mathsf{pred} & : & \mathsf{Nat} \to \mathsf{Nat} \\ \mathsf{null} & : & \mathsf{Nat} \to * \end{array}$$

munies des règles de réduction

$$\begin{array}{rcl} \mathsf{pred}\,\mathsf{0} & \to & \mathsf{0} \\ \mathsf{pred}\,(\mathsf{s}\,N) & \to & N \end{array} \qquad \begin{array}{rcl} \mathsf{null}\,\mathsf{0} & \to & \forall x : * \,.\, x \Rightarrow x \\ \mathsf{null}\,(\mathsf{s}\,N) & \to & \forall x : * \,.\, x \end{array}$$

lesquelles sont incorporées dans les deux règles de conversion (typage des termes-objets et des termes de preuves) de la Fig. 2.2.

La constante pred sert à dériver l'injectivité du successeur (3e axiome de Peano) tandis que la constante null sert à dériver l'inégalité $\mathsf{s}\,N \neq \mathsf{0}$ (4e axiome de Peano). Il n'est pas nécessaire d'ajouter un principe de récurrence, dans la mesure où il est possible de définir au second ordre la plus petite sous-classe de Nat contenant $\mathsf{0}$ et close par successeur. On vérifie alors que :

Proposition 3 — *Si une formule close ϕ est dérivable dans la théorie des ensembles de Zermelo intuitionniste, alors la proposition ϕ^* admet un terme de preuve clos dans le système $F\omega.2^+$ avec entiers primitifs.*

2.2.4 Le modèle de normalisation

Le système $F\omega.2^+$ (avec entiers primitifs) ne nous intéresse ici que dans la mesure où il permet de donner un contenu calculatoire fortement normalisable aux preuves de la théorie des ensembles intuitionniste. On ne s'intéresse donc ni à la propriété de *subject reduction* (sans doute fausse dans $F\omega.2^{++}$) ni même aux propriétés de normalisation de la couche supérieure (les termes-objets).

La propriété de normalisation forte des termes de preuves s'établit à l'aide d'un modèle de normalisation $\mathcal{M}$ dont la construction est tout ce qu'il y a de

Formation des signatures : $\vdash \Sigma$ **sig**

$$\frac{}{\vdash \varnothing \text{ sig}} \qquad \frac{\Sigma \vdash T : \Box_i}{\vdash \Sigma, x : T \text{ sig}}\ {\scriptstyle i\in\{1;2\},\ x\notin \mathrm{dom}(\Sigma)}$$

Typage des termes-objets : $\Sigma \vdash M : T$

$$\frac{\vdash \Sigma \text{ sig}}{\Sigma \vdash x : T}\ {\scriptstyle (x:T)\in\Sigma} \qquad \frac{\vdash \Sigma \text{ sig}}{\Sigma \vdash * : \Box_1} \qquad \frac{\vdash \Sigma \text{ sig}}{\Sigma \vdash \Box_1 : \Box_2}$$

$$\frac{\Sigma \vdash T : \Box_i \qquad \Sigma, x : T \vdash U : \Box_j}{\Sigma \vdash \Pi x : T \,.\, U : \Box_{\max(i,j)}}\ {\scriptstyle i,j\in\{1;2\}} \qquad \frac{\Sigma \vdash M : T}{\Sigma \vdash M : T'}\ {\scriptstyle T=_\beta T'}$$

$$\frac{\Sigma, x : T \vdash M : U}{\Sigma \vdash \lambda x : T \,.\, M : \Pi x : T \,.\, U} \qquad \frac{\Sigma \vdash M : \Pi x : T \,.\, U \qquad \Sigma \vdash N : T}{\Sigma \vdash M\, N : U\{x := N\}}$$

$$\frac{\Sigma \vdash A : * \qquad \Sigma \vdash B : *}{\Sigma \vdash A \Rightarrow B : *} \qquad \frac{\Sigma, x : T \vdash A : *}{\Sigma \vdash Q x : T \,.\, A : *}\ {\scriptstyle Q\in\{\forall;\exists\}}$$

Formation des contextes logiques : $\Sigma \vdash \Gamma$ **ctx**

$$\frac{\vdash \Sigma \text{ sig}}{\Sigma \vdash \varnothing \text{ ctx}} \qquad \frac{\Sigma \vdash \Gamma \text{ ctx} \qquad \Sigma \vdash A : *}{\Sigma \vdash \Gamma, \xi : A \text{ ctx}}\ {\scriptstyle \xi\notin \mathrm{dom}(\Gamma)}$$

Typage des termes de preuves : $\langle\Sigma\rangle\ \Gamma \vdash t : A$

$$\frac{\Sigma \vdash \Gamma \text{ ctx}}{\langle\Sigma\rangle\ \Gamma \vdash \xi : A}\ {\scriptstyle (\xi:A)\in\Gamma} \qquad \frac{\langle\Sigma\rangle\ \Gamma \vdash t : A}{\langle\Sigma\rangle\ \Gamma \vdash t : A'}\ {\scriptstyle A=_\beta A'}$$

$$\frac{\langle\Sigma\rangle\ \Gamma, \xi : A \vdash t : B}{\langle\Sigma\rangle\ \Gamma \vdash \lambda \xi \,.\, t : A \Rightarrow B} \qquad \frac{\langle\Sigma\rangle\ \Gamma \vdash t : A \Rightarrow B \qquad \langle\Sigma\rangle\ \Gamma \vdash u : A}{\langle\Sigma\rangle\ \Gamma \vdash t u : B}$$

$$\frac{\langle\Sigma, x : T\rangle\ \Gamma \vdash t : A}{\langle\Sigma\rangle\ \Gamma \vdash t : \forall x : T \,.\, A}\ {\scriptstyle x\notin FV(\Gamma)} \qquad \frac{\Sigma \vdash N : T \qquad \langle\Sigma\rangle\ \Gamma \vdash t : \forall x : T \,.\, A}{\langle\Sigma\rangle\ \Gamma \vdash t : A\{x := N\}}$$

$$\frac{\Sigma, x : T \vdash A : * \qquad \Sigma \vdash N : T \qquad \langle\Sigma\rangle\ \Gamma \vdash t : A\{x := N\}}{\langle\Sigma\rangle\ \Gamma \vdash t : \exists x : T \,.\, A}$$

$$\frac{\Sigma, x : T \vdash A : * \qquad \Sigma \vdash B : * \qquad \langle\Sigma\rangle\ \Gamma \vdash t : \forall x : T \,.\, (A \Rightarrow B)}{\langle\Sigma\rangle\ \Gamma \vdash t : (\exists x : T \,.\, A) \Rightarrow B}$$

FIG. 2.2 – Règles de typage du système $F\omega.2^+$

plus standard. Les termes objets sont interprétés suivant un schéma purement ensembliste, et les termes de preuves (bien typés) sont interprétés par eux-mêmes (à une substitution près) en utilisant l'information de réductibilité du modèle.

Plus précisément :

- Les univers prédicatifs $\Box_1$ et $\Box_2$ sont interprétés par des univers ensemblistes construits à partir de deux cardinaux inaccessibles [59] μ_1 et μ_2 tels que $\mu_1 < \mu_2$. Comme pour n'importe quelle preuve de normalisation forte, on doit commencer par exclure le type vide dans le modèle (même au niveau des termes objets). On posera donc $[\![\Box_1]\!] = V_{\mu_1} \setminus \{\varnothing\}$ et $[\![\Box_2]\!] = V_{\mu_2} \setminus \{\varnothing\}$, où V_α désigne l'ensemble des ensembles de rang α dans la hiérarchie de Veblen [59].
- Le produit dépendant est interprété par le produit cartésien infinitaire de la théorie des ensembles, tandis que l'abstraction (annotée) et l'application sont interprétées par les constructions équivalentes en théorie des ensembles. Le type Nat est interprété par l'ensemble des entiers naturels, et les constantes $\mathsf{0}$, s, pred et null sont interprétées de manière évidente.
- La sorte $*$ des propositions est interprétée par l'ensemble des *ensembles saturés* au sens de Tait [74, 75]. (C'est-à-dire par un ensemble d'ensembles de λ-termes purs qui, rappelons-le, est à la fois clos par intersection et par union de familles non vides.) L'implication est interprétée par la flèche de la réalisabilité (intuitionniste) tandis que les quantifications universelles et existentielles sont interprétées respectivement par l'intersection et l'union des familles (non vides) d'ensembles saturés correspondants.

On montre alors deux résultats dont découle immédiatement le théorème de normalisation forte au niveau des termes de preuves.

D'abord un résultat de correction vis-à-vis du typage des termes-objets :

Proposition 4 — *Si $\Sigma \vdash M : T$ (dans $F\omega.2^+ + \mathsf{Nat}$), alors pour toute valuation $\rho \in [\![\Sigma]\!]$, on a $[\![M]\!]_\rho \in [\![T]\!]_\rho$.*

(On se reportera à [75] pour la définition de la notion de valuation et de l'interprétation des signatures.) Et enfin un résultat d'adéquation vis-à-vis de l'information de réductibilité incorporée dans le modèle :

Proposition 5 — *Si $\langle\Sigma\rangle\ \Gamma \vdash t : A$ (dans $F\omega.2^+ + \mathsf{Nat}$), alors pour toute valuation $\rho \in [\![\Sigma]\!]$ et pour toute substitution $\sigma \in [\![\Gamma]\!]_\rho$, on a $t[\sigma] \in\in [\![A]\!]_\rho$.*

(On se reportera à [76] pour la définition de la notion de substitution et de l'interprétation des contextes logiques.)

2.2.5 Opérateur ε intuitionniste et schéma de collection

Une idée très naturelle pour interpréter (à travers la traduction $\phi \mapsto \phi^*$) la forme de choix qui est à l'œuvre dans le schéma de collection

$$\text{(Collection)} \qquad \forall a\ \big(\forall x \in a\ \exists y\ \phi(x,y)\ \Rightarrow\ \exists b\ \forall x \in a\ \exists y \in b\ \phi(x,y)\big)$$

est d'enrichir la théorie des types cible avec un opérateur ε de Hilbert [50, 51] capable de choisir pour chaque $x \in a$ une structure de graphe pointé représentant un y tel que $\phi(x,y)$. Techniquement, un tel opérateur se présente dans la syntaxe des termes-objets sous la forme d'une nouvelle construction $\boldsymbol{\varepsilon} x : T\,.\,A$

qui à chaque type T et à chaque proposition A dépendant (éventuellement) de la variable $x : T$ associe un objet $x : T$ satisfaisant la proposition $A(x)$, ou n'importe quel objet de type T si la proposition $A(x)$ est uniformément fausse. (On notera que cela suppose qu'un type T qui vit dans la couche haute du formalisme n'est jamais vide, ce qui est déjà le cas dans le modèle de normalisation.)

Une forme intuitionniste de l'opérateur ε de Hilbert Malheureusement, il y a peu de chances pour qu'on puisse un jour interpréter un tel opérateur de choix (même non extensionnel) dans un cadre purement intuitionniste ! Pourtant, j'ai montré [75][10] qu'il est possible d'affaiblir la spécification d'un tel opérateur en supposant qu'il retourne non pas un unique objet de type T (ce qui est trop fort) mais une suite dénombrable d'objets de type T parmi lesquels figure au moins un élément satisfaisant la propriété désirée, sous réserve que cette propriété soit satisfaite par au moins un élément de T.

Formellement, on étend la syntaxe du formalisme avec la construction

$$\textbf{Termes-objets} \qquad M, N, T, U, A, B \quad ::= \quad \cdots \quad | \quad \boldsymbol{\varepsilon} x : T \,.\, A$$

(« opérateur ε de Hilbert intuitionniste ») dont la formation est assurée par une nouvelle de typage

$$\frac{\Sigma, x : T \vdash A : *}{\Sigma \vdash \boldsymbol{\varepsilon} x : T \,.\, A : \mathsf{Nat} \to A}$$

Dans le modèle de normalisation $\mathcal{M}$ (2.2.4), on interprète la construction $\boldsymbol{\varepsilon} x : T \,.\, A(x)$ par une suite $(u_n)_{n\in\mathbb{N}}$ d'éléments de $[\![T]\!]$ construite de telle sorte que pour chaque $n \in \mathbb{N}$, l'élément u_n est

– un élément de $[\![T]\!]$ tel que $t_n \in [\![A(u_n)]\!]$ s'il en existe ; ou bien
– n'importe quel élément de $[\![T]\!]$ sinon ;

où $n \mapsto t_n$ désigne une bijection entre les entiers naturels et les λ-termes purs (en considérant les termes ouverts) fixée à l'avance. Grâce à cette définition qui combine le tiers-exclu et l'axiome du choix (dans le modèle), on vérifie que

Proposition 6 — *Dans le modèle de normalisation $\mathcal{M}$, les deux propositions*

$$\exists x : T \,.\, A \qquad \textit{et} \qquad \exists n : \mathsf{Nat} \,.\, A\{x := (\boldsymbol{\varepsilon} x : T \,.\, A)\, n\}$$

sont dénotées par le même ensemble saturée (au sens de Tait).

Cette égalité dénotationnelle justifie alors l'introduction d'une nouvelle règle de typage au niveau des termes de preuves (en fait : une règle de sous-typage déguisée) qui est la suivante :

$$\frac{\langle\Sigma\rangle\ \Gamma \vdash t : \exists x : T \,.\, A}{\langle\Sigma\rangle\ \Gamma \vdash t : \exists n : \mathsf{Nat} \,.\, A\{x := (\boldsymbol{\varepsilon} x : T \,.\, A)\, n\}}$$

Dans ce qui suit, on désigne par $F\omega.2^{++}$ le système $F\omega.2^{+}$ étendu avec les entiers primitifs (décrits à la section 2.2.3) et l'opérateur ε de Hilbert intuitionniste (avec ses deux règles). D'après la proposition qui précède, les nouvelles règles sont correctes et adéquates vis-à-vis du modèle de normalisation, et la propriété de normalisation forte (des termes de preuves) reste vraie dans $F\omega.2^{++}$.

[10] En m'inspirant de l'astuce utilisée par Krivine pour réaliser l'axiome du choix dépendant [60], mais sans introduire une instruction supplémentaire, qui n'est ici pas nécessaire.

Interprétation du schéma de collection Le système $F\omega.2^{++}$ permet de démontrer l'implication

$$\exists x : T \,.\, A(x) \;\Rightarrow\; \exists n : \mathsf{Nat} \,.\, A\big((\boldsymbol{\varepsilon} x : T \,.\, A(x))\, n\big)$$

(avec le terme de preuve $\lambda\xi \,.\, \xi$) qui exprime qu'on peut toujours remplacer une quantification existentielle sur un type T quelconque (et éventuellement un « grand type », de type $\Box_2$) par une quantification existentielle dénombrable, c'est-à-dire sur un « petit type », de type $\Box_1$. C'est précisément cette propriété qui va nous permettre de dériver le schéma de collection.

Soit $\phi(y)$ une formule ensembliste dépendant d'une variable y, traduite dans le système $F\omega.2^{++}$ par une proposition $\phi^*(\bar{y}, \tilde{y}, \dot{y})$ dépendant de trois variables $\bar{y} : \Box_1$, $\tilde{y} : \bar{y} \to \bar{y} \to *$ et $\dot{y} : \bar{y}$. (À ce stade il n'est pas nécessaire de mentionner la variable x, qui joue un simple rôle de paramètre dans la construction.) À cette proposition, on associe une suite de types $Y_\phi : \mathsf{Nat} \to \Box_1$ définie par

$$Y_\phi \;\equiv\; \boldsymbol{\varepsilon} \bar{y} : \Box_1 \,.\, \exists \tilde{y} : (\bar{y} \to \bar{y} \to *) \,.\, \exists \dot{y} : \bar{y} \,.\, \phi^*(\bar{y}, \tilde{y}, \dot{y})$$

Cette suite vérifie la propriété que si la proposition $\phi(\bar{y}, \tilde{y}, \dot{y})$ est satisfaite par au moins un graphe pointé $(\bar{y}, \tilde{y}, \dot{y})$ dans l'univers, alors il existe $n : \mathsf{Nat}$ et un graphe pointé de la forme $(Y_\phi n, R, r)$ tel que $\phi^*(Y_\phi n, R, r)$.

Par ailleurs, il est facile (mais relativement technique) de construire, pour un type $Z : \Box_1$ fixé, un graphe pointé universel $U(Z)$ qui définit (intuitivement) l'ensemble de tous les ensembles susceptibles d'être représentés par un graphe pointé supporté par le type Z. (Il s'agit en fait d'une construction très proche de l'ensemble des parties, au moins dans l'esprit.)

Par conséquent, le graphe pointé $G_\phi = \bigcup_{n:\mathsf{Nat}} U(Y_\phi n)$ (en identifiant les graphes pointés avec les ensembles qu'ils représentent) contient au moins un élément y tel que $\phi(y)$, à supposer qu'il existe un tel élément dans l'univers. On notera que cette construction n'est possible que parce que le domaine de cette union, le type $\mathsf{Nat} : \Box_1$, reste un support possible de graphe pointé. (Ce qui ne serait pas le cas avec le type $\Box_1 : \Box_2$.)

Si maintenant on considère un ensemble a et une relation binaire $\phi(x, y)$ dépendant de deux variables x et y, il suffit de construire un graphe pointé représentant l'ensemble $b = \bigcup_{x \in a} G'_{\phi(x,\cdot)}$ et le tour est joué.

Toute cette construction dont nous n'avons donné qu'une description très informelle a été entièrement formalisée en Coq (étendu avec un symbole ε de Hilbert intuitionniste axiomatisé), et le développement correspondant (`IZF`) se trouve dans les contributions des utilisateurs de Coq à l'URL

`http ://coq.inria.fr/distrib/v8.2/contribs/IZF.html`

2.2.6 En guise de conclusion

La force théorique de $F\omega.2^{++}$ Le lecteur aura sans doute remarqué que la preuve de cohérence relative de IZF_C par rapport à $F\omega.2^{++}$ n'utilise qu'une petite fraction de ce système de types, notamment dans la sorte $\Box_2$ où le plus grand type rencontré est le type $\mathsf{Nat} \to \Box_1$ (qui est utilisé dans la preuve de la traduction du schéma de collection).

De fait, le système $F\omega.2^{++}$ est beaucoup plus fort que IZF_C : il suffit de penser au type $\Pi X : \Box_1 \,.\, (X \to X \to *) \to X \to *$ qui représente naturellement

le type des classes d'ensembles. Pour cette raison, il me semble à peu près clair que le système $F\omega.2^{++}$ a au moins la puissance théorique de la théorie des ensembles intuitionniste d'ordre supérieur avec schéma de collection.

Subject-reduction Je conjecture que le système de preuves de $F\omega.2^{++}$ ne satisfait pas la propriété de *subject reduction* (sans avoir de contre-exemple à proposer), notamment à cause de la règle de typage

$$\frac{\langle\Sigma\rangle\ \Gamma \vdash t : \exists x : T\,.\,A}{\langle\Sigma\rangle\ \Gamma \vdash t : \exists n : \mathsf{Nat}\,.\,A\{x := (\boldsymbol{\varepsilon} x : T\,.\,A)\,n\}}$$

qui fait intervenir la construction $\boldsymbol{\varepsilon} x : T\,.\,A$ dépourvue de toute règle de calcul.

Si c'est bien le cas, le calcul effectué par les termes de preuves ne peut pas être suivi pas à pas au niveau de la dérivation de typage. Ce divorce entre le calcul et la déduction (familier en réalisabilité intuitionniste et classique) est à mon avis à rapprocher du fait que dans IZF_C, la propriété du témoin est fausse en général, sauf en ce qui concerne les témoins numériques.

Propriétés du témoin dans IZF_R et IZF_C On peut en effet distinguer deux formes de propriété du témoin en théorie des ensembles intuitionniste : la propriété du témoin numérique, et la propriété du témoin généralisée.

La propriété du témoin numérique exprime que si une formule existentielle numérique close (c'est-à-dire de la forme $\exists n \in \omega\ \phi(n)$, où n est la seule variable libre de $\phi(n)$) est démontrable dans la théorie des ensembles intuitionniste considérée, alors la formule $\phi(n_0)$ est démontrable dans cette même théorie pour un certain entier naturel n_0. (En travaillant dans un langage étendu avec des symboles de Skolem pour le zéro et le successeur.)

La propriété du témoin généralisée exprime que si une formule existentielle close $\exists x\,\phi(x)$ (où x est la seule variable libre de la formule $\phi(x)$) est démontrable dans la théorie des ensembles intuitionniste considérée, alors on peut trouver une formule $\psi(x)$ telle que les formules

$$\exists! x\ \psi(x) \qquad \text{et} \qquad \exists x\ (\psi(x) \wedge \phi(x))$$

soient toutes les deux démontrables dans cette même théorie. On notera qu'ici le témoin est représenté par la formule $\psi(x)$ qui définit un ensemble et un seul (formule de gauche), vérifiant par ailleurs la propriété ϕ (formule de droite).

Myhill [82] et Friedman [37] ont démontré que les systèmes IZ, IZF_R et IZF_C vérifient tous les trois la propriété du témoin numérique (par un argument de réalisabilité à la Kleene), et que les systèmes IZ et IZF_R vérifient également la propriété du témoin généralisée. Quelques années plus tard, Friedman et Ščedrov [38] ont en revanche démontré que le système IZF_C ne vérifie pas la propriété du témoin généralisée, en exhibant un contre-exemple approprié. (C'est ce contre-exemple qui permet de déduire que l'inclusion $\mathrm{IZF}_R \subset \mathrm{IZF}_C$ est stricte.)

Là encore, on retrouve cette même idée d'un divorce entre le calcul et le système de déduction, et le système IZF_C semble être un système logique bien trop puissant pour que le système de déduction (basé sur les formules) arrive à suivre le calcul des réalisateurs ou des termes de preuves.

C'est ce qui m'amène à penser qu'il n'est sans doute pas possible de dériver le schéma de collection en théorie des types sans introduire une construction *ad hoc* (au sens du typage) qui casse le lien entre le calcul et la déduction.

D'où la conjecture :

Conjecture 1 — *Les systèmes de types qui satisfont la propriété de* subject-reduction *au niveau des termes de preuves (les PTS, les formalismes en déduction modulo [32], etc.) ne peuvent par capturer exactement la théorie* IZF_C.

Ce problème n'apparaît pas dans le système de Moczydłowski [81] qui ne traite que le schéma de remplacement intuitionniste. (Le formalisme vérifie la *subject-reduction* ainsi que les deux formes de la propriété du témoin.)

2.3 Théorie de Zermelo et système λZ

Le système $F\omega.2^{++}$ présenté à la section 2.2 cherchait à répondre à la question : « quelle extension de la théorie des types permet d'exprimer la force du schéma de collection de la théorie de Zermelo-Fraenkel ? ». Dans cette section, on s'intéresse maintenant à la question : « quel fragment de la théorie des types permet de capturer très exactement la théorie des ensembles de Zermelo ? ». Pour répondre à cette question, j'avais introduit dans ma thèse le système λZ, et conjecturé que ce système capture exactement la force de la théorie des ensembles de Zermelo. Dans [76] j'ai démontré cette conjecture à partir d'un résultat plus fort, à savoir que le système λZ est à travers la traduction $\phi \mapsto \phi^*$ une extension conservative du système IZ + AFA + TC, un système dont il est facile de se convaincre qu'il est équiconsistant à la théorie de Zermelo [33].

2.3.1 Le PTS λZ

Le système λZ est un PTS [11] à quatre sortes défini par

$$\begin{array}{rcl} \mathscr{S} & = & \{*;\ \Box_1;\ \Box_2;\ \Box_3\} \\ \mathscr{A} & = & \{(* : \Box_1);\ (\Box_1 : \Box_2);\ (\Box_2 : \Box_3)\} \\ \mathscr{R} & = & \{(*,*,*);\ (\Box_i,*,*)\ :\ i \in \{1;2;3\}\} \quad \cup \\ & & \{(\Box_i,\Box_j,\Box_{\max(i,j)})\ :\ i,j \in \{1;2\}\} \end{array}$$

Comme pour le système $F\omega.2^{++}$ (et comme pour tous les systèmes de la face gauche du cube de Barendregt), le PTS λZ admet une présentation stratifiée (Fig. 2.3) dans laquelle les termes-objets (de sorte $\Box_i$) constituent une catégorie syntaxique indépendante des termes de preuves (de sorte $*$).

Termes-objets	M, N, T, U, A, B	::=	$x \mid * \mid \Box_i \quad (i \in \{1;2;3\})$
			$\mid \lambda x : T \,.\, M \mid MN \mid \Pi x : T \,.\, U$
			$\mid A \Rightarrow B \mid \forall x : T \,.\, A$
Termes de preuves	t, u	::=	ξ
			$\mid \lambda \xi^A . t \mid tu$
			$\mid \lambda x : T \,.\, t \mid tM$

FIG. 2.3 – Présentation stratifiée du système λZ

Dans le cadre d'une correspondance avec la théorie des ensembles, les univers prédicatifs $\Box_i$ ($i \in \{1;2;3\}$) et les règles de formation qui leur sont associées peuvent se comprendre de la manière suivante :

1. Le premier univers $\Box_1$ est l'univers des types finis, dont le rôle est ici de permettre la construction d'un type prouvablement infini dans la sorte $\Box_2$ (cf section 2.1.2). La quantification universelle $\forall x : T \,.\, A(x)$ formée à l'aide de la règle $(\Box_1, *, *)$ correspond donc approximativement à la quantification de domaine fini $\forall x < t \;\; A(x)$ (avec $t \in \omega$) en théorie des ensembles. La règle $(\Box_1, \Box_1, \Box_1)$ permet quant à elle de définir toutes les constructions standard sur les ensembles finis. On l'utilisera ici uniquement pour construire un type infini $\mathsf{Nat} : \Box_2$.
2. L'univers $\Box_2$ est la sorte des supports des graphes pointés. Ainsi, la quantification universelle $\forall x : T \,.\, A(x)$ formée par la règle $(\Box_2, *, *)$ correspond à la quantification bornée $\forall x \in t \;\; A(x)$ de la théorie des ensembles (où t est un ensemble arbitraire). Les règles de formation $(\Box_2, \Box_2, \Box_2)$, $(\Box_1, \Box_2, \Box_2)$ et $(\Box_2, \Box_1, \Box_2)$ permettent de définir les constructions standard sur les ensembles (en faisant intervenir éventuellement un type fini).
3. Enfin, la sorte $\Box_3$ est la sorte maximale du système, dont l'unique habitant est la sorte $\Box_2$ (en raison de l'absence de règles de la forme $(s_1, s_2, \Box_3)$). La quantification universelle $\forall X : \Box_2 \,.\, A(X)$ formée par la règle $(\Box_3, *, *)$ correspond ainsi à la quantification universelle non bornée sur tous les ensembles.

2.3.2 Traduction de la théorie des ensembles

Les formules de la théorie des ensembles se plongent dans le système λZ à l'aide de la traduction $\phi \mapsto \phi^*$ définie à la section 2.1.2, en travaillant ici avec des graphes pointés dont le support est de type $\Box_2$.

D'après les propositions 1 et 2, il est clair que :

Proposition 7 (Correction) — *Si une formule (close) de la théorie des ensembles est prouvable dans* IZ + AFA + TC*, alors la proposition ϕ^* admet un terme de preuve (clos) dans le système λZ.*

À travers la traduction $\phi \mapsto \phi^*$, le système λZ peut donc être vu comme une extension du système IZ + AFA + TC. Nous allons maintenant démontrer qu'il s'agit en réalité d'une extension conservative, ce qui impliquera automatiquement l'équiconsistance des deux systèmes.

2.3.3 Traduction du système de types

Il s'agit de définir à présent une traduction en sens inverse, c'est-à-dire une traduction du système λZ vers la théorie des ensembles de Zermelo intuitionniste. La tâche est ici bien plus ardue que dans le cas du système $F\omega.2^{++}$ décrit à la section 2.2, car on ne s'autorise ici (dans la théorie des ensembles cible) ni les cardinaux inaccessibles ni même le schéma de remplacement. La définition d'une telle traduction va en pratique nous poser deux problèmes.

Le premier problème vient du fait que la théorie des ensembles de Zermelo dispose d'un langage de termes trop rudimentaire (car réduit aux seules variables) pour qu'on puisse l'utiliser comme langage cible dans le cadre d'une

traduction quelconque. Pour résoudre ce problème, on va donc devoir étendre la syntaxe de la théorie des ensembles de Zermelo avec des symboles de Skolem (pour désigner les paires, les unions, les ensembles des parties, etc.) mais aussi une construction $\{x \in t \ : \ \phi\}$ pour désigner les ensembles définis par compréhension — une construction qui n'est pas à proprement parler un symbole de Skolem dans la mesure où elle fait intervenir une formule. Ceci va nous amener à introduire une version skolémisée de la théorie de Zermelo — le système Z^{sk} — et à étudier ses propriétés, notamment la propriété de conservativité vis-à-vis de la théorie de Zermelo sous sa forme usuelle.

Le second problème vient du fait qu'en théorie des ensembles, on traduit le produit dépendant $\Pi x : T \,.\, U(x)$ par le produit cartésien infinitaire $\prod_{x \in T} U(x)$ dont la construction habituelle repose sur le schéma de remplacement[11]. Pour résoudre ce problème, nous devrons donc montrer que la forme particulière du schéma de remplacement qui est nécessaire pour définir cette construction est déjà présente dans la théorie de Zermelo, du moins dans sa version skolémisée que nous allons maintenant présenter.

2.3.4 Le système Z^{sk}

On introduit un système formel noté Z^{sk} dont les termes et les formules sont mutuellement définis par :

$$\begin{array}{lll} \textbf{Termes} & t, u & ::= \ x \mid \omega \mid \{t_1; t_2\} \mid \mathfrak{P}(t) \mid \bigcup t \mid \{x \in t \ : \ \phi\} \\ \textbf{Formules} & \phi, \psi & ::= \ t = u \mid t \in u \mid \top \mid \bot \\ & & \quad \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \Rightarrow \psi \mid \forall x \ \phi \mid \exists x \ \phi \end{array}$$

Les ensembles de variables libres dans les formules et dans les termes (notation : $FV(\phi)$, $FV(t)$) sont définis de la manière attendue de même que les notions de substitution externe correspondantes (notation : $\phi\{x := u\}$, $t\{x := u\}$), en tenant compte du fait que toutes les occurrences libres de la variable x dans la formule ϕ sont liées dans la construction de terme $\{x \in t \ : \ \phi\}$ (mais les occurrences libres de x dans t restent libres).

Bien que le système Z^{sk} ne soit pas basé sur un langage du premier ordre, les notions de séquent, d'axiome, de règle d'inférence et de dérivation sont définies comme dans les théories du premier ordre (en les adaptant au langage de termes et de formules de Z^{sk}). Les axiomes de Z^{sk} (Fig. 2.4) sont les mêmes que ceux de Z, à cette différence près que chacun des axiomes existentiels de la théorie est remplacé par sa forme skolémisée. Dans la Fig. 2.4 comme dans la suite de cette section, on utilise les abréviations

$$\begin{array}{llll} \neg\phi & \equiv \ \phi \Rightarrow \bot & \phi \Leftrightarrow \psi & \equiv \ (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi) \\ x \notin y & \equiv \ \neg(x \in y) & x \subseteq y & \equiv \ \forall z \ (z \in x \Rightarrow z \in y) \\ \mathrm{zero}(x) & \equiv \ \forall z \ (z \notin x) & \mathrm{succ}(x, y) & \equiv \ \forall z \ (z \in y \Leftrightarrow z \in x \vee z = x) \end{array}$$

$$\mathrm{nat}(n) \ \equiv \ \forall a \ \Big(\begin{array}{l} \forall x \ (\mathrm{zero}(x) \ \Rightarrow \ x \in a) \ \ \wedge \\ \forall x \ \forall y \ (x \in a \wedge \mathrm{succ}(x, y) \ \Rightarrow \ y \in a) \ \Rightarrow \ n \in a \end{array} \Big)$$

[11] En fait on a besoin du schéma de remplacement de manière essentielle uniquement dans le cas où la famille d'ensembles $U(x)$ (pour $x \in T$) est définie au moyen d'une relation fonctionnelle. Dans le cas où cette famille est définie au sens d'une fonction ensembliste $x \in T \mapsto U(x)$ (ce qui ne sera pas le cas ici), le schéma de remplacement est bien évidemment superflu.

Axiomes d'égalité

(Refl.)	$\forall x\ (x = x)$
(Sym.)	$\forall x\ \forall y\ (x = y\ \Rightarrow\ y = x)$
(Trans.)	$\forall x\ \forall y\ \forall z\ (x = y\ \wedge\ y = z\ \Rightarrow\ x = z)$
(Compat-G.)	$\forall x\ \forall y\ \forall z\ (x = y\ \wedge\ y \in z\ \Rightarrow\ x \in z)$
(Compat-D.)	$\forall x\ \forall y\ \forall z\ (x \in y\ \wedge\ y = z\ \Rightarrow\ x \in z)$

Axiomes de Zermelo (skolémisés)

(Ext.)	$\forall a\ \forall b\ (\forall x\ (x \in a \Leftrightarrow x \in b)\ \Rightarrow\ a = b)$
(Paire$^{\mathrm{sk}}$)	$\forall a_1\ \forall a_2\ \forall x\ (x \in \{a_1; a_2\}\ \Leftrightarrow\ x = a_1\ \vee\ x = a_2)$
(Compr.$^{\mathrm{sk}}$)	$\forall x_1 \cdots \forall x_n\ \forall a\ \forall x\ (x \in \{z \in a\ :\ \phi\}\ \Leftrightarrow\ x \in a\ \wedge\ \phi\{z := x\})$ pour chaque formule ϕ telle que $FV(\phi) \subseteq \{x_1; \ldots; x_n; z\}$.
(Parties$^{\mathrm{sk}}$)	$\forall a\ \forall x\ (x \in \mathfrak{P}(a)\ \Leftrightarrow\ x \subseteq a)$
(Union$^{\mathrm{sk}}$)	$\forall a\ \forall x\ (x \in \bigcup a\ \Leftrightarrow\ \exists y\ (y \in a\ \wedge\ x \in y))$
(Infini$^{\mathrm{sk}}$)	$\forall x\ (x \in \omega\ \Leftrightarrow\ \mathrm{nat}(x))$

Fig. 2.4 – Les axiomes de Z^{sk}

Le fragment intuitionniste du système Z^{sk} est noté $\mathrm{IZ}^{\mathrm{sk}}$.

Le système (I)Z^{sk} est clairement une extension de (I)Z,[12] en ce sens que pour toute formule ϕ de la théorie des ensembles, (I)$\mathrm{Z} \vdash \phi$ entraîne (I)$\mathrm{Z}^{\mathrm{sk}} \vdash \phi$.

À partir des axiomes de la Fig. 2.4, on vérifie facilement que les symboles de fonction $\{\_;\_\}$, $\mathfrak{P}(\_)$ et $\bigcup\_$ sont compatibles avec l'égalité (dans $\mathrm{IZ}^{\mathrm{sk}}$) de même que la construction $\{x \in t\ :\ \phi\}$ en ce sens que :

$$\begin{array}{lll} \mathrm{IZ}^{\mathrm{sk}} & \vdash & \forall x_1 \cdots \forall x_n\ \forall a\ \forall a'\ (a = a'\ \Rightarrow\ \{x \in a\ :\ \phi\} = \{x \in a'\ :\ \phi\}) \\ \mathrm{IZ}^{\mathrm{sk}} & \vdash & \forall x_1 \cdots \forall x_n\ \forall a\ (\forall x\ (\phi \Leftrightarrow \phi')\ \Rightarrow\ \{x \in a\ :\ \phi\} = \{x \in a\ :\ \phi'\}) \end{array}$$

(pour toutes formules ϕ, ϕ' de Z^{sk} telles que $FV(\phi) \cup FV(\phi') \subset \{x_1; \ldots; x_n; x\}$), ce qui nous donne le principe de Leibniz (i.e. remplacement des égaux par les égaux) à la fois dans les termes et dans les formules :

Proposition 8 (Principe de Leibniz) — *Pour chaque terme t et pour chaque formule ϕ du langage de Z^{sk} :*

$$\begin{array}{lll} \mathrm{IZ}^{\mathrm{sk}} & \vdash & x_1 = x_2\ \Rightarrow\ t\{x := x_1\} = t\{x := x_2\} \\ \mathrm{IZ}^{\mathrm{sk}} & \vdash & x_1 = x_2\ \Rightarrow\ \phi\{x := x_1\} \Leftrightarrow \phi\{x := x_2\} \end{array}$$

Dans Z^{sk} (et, en fait, dans $\mathrm{IZ}^{\mathrm{sk}}$), la plupart des notations mathématiques standard telles que $\varnothing$ (ensemble vide), $x \cup y$ (union binaire), $x \cap y$ (intersection

[12] L'abréviation (I)Z signifie : « Z (resp. IZ) ». De même pour (I)Z^{sk}.

binaire), $x \setminus y$ (différence), $f(x)$ (application de fonction), $\langle x, y\rangle$ (couple), B^A (espace des fonctions de A dans B) sont définissables comme des macros dans l'algèbre de termes enrichie.

2.3.5 La procédure de déskolémisation

La preuve de conservativité de (I)Z^{sk} par rapport à (I)Z repose sur une procédure de déskolémisation définie à partir de deux traductions :

- Une traduction des *termes*, qui à chaque couple (t, z) formé d'un terme t de Z^{sk} et d'une variable z (éventuellement libre dans t) associe une formule de la théorie des ensembles notée $z \in^\circ t$;[13]
- Une traduction des *formules*, qui à chaque formule ϕ de Z^{sk} associe une formule de la théorie des ensembles notée ϕ°.

Les deux traductions sont définies par récursion mutuelle sur t et ϕ à partir des équations de la Fig. 2.5.

$$
\begin{array}{lcl}
z \in^\circ x & \equiv & z \in x \\
z \in^\circ \omega & \equiv & \mathrm{nat}(z) \\
z \in^\circ \{t_1; t_2\} & \equiv & (z = t_1)^\circ \ \vee \ (z = t_2)^\circ \\
z \in^\circ \mathfrak{P}(t) & \equiv & \forall x \ (x \in z \ \Rightarrow \ x \in^\circ t) \\
z \in^\circ \bigcup t & \equiv & \exists y \ (y \in^\circ t \wedge z \in y) \\
z \in^\circ \{x \in t \ : \ \phi\} & \equiv & z \in^\circ t \ \wedge \ \phi^\circ\{x := z\}
\end{array}
$$

$$
\begin{array}{lcl@{\qquad}lcl}
(t = u)^\circ & \equiv & \forall z \ (z \in^\circ t \Leftrightarrow z \in^\circ u) & (\phi \wedge \psi)^\circ & \equiv & \phi^\circ \wedge \psi^\circ \\
(t \in u)^\circ & \equiv & \exists z \ ((z = t)^\circ \wedge z \in^\circ u) & (\phi \vee \psi)^\circ & \equiv & \phi^\circ \vee \psi^\circ \\
 & & & (\phi \Rightarrow \psi)^\circ & \equiv & \phi^\circ \Rightarrow \psi^\circ \\
\top^\circ & \equiv & \top & (\forall x \ \phi)^\circ & \equiv & \forall x \ \phi^\circ \\
\bot^\circ & \equiv & \bot & (\exists x \ \phi)^\circ & \equiv & \exists x \ \phi^\circ
\end{array}
$$

FIG. 2.5 – Déskolémisation des termes et des formules de Z^{sk}

La procédure de déskolémisation préserve la signification des termes et des formules dans IZ^{sk} en ce sens que :

Proposition 9 (Équivalence de traduction) — *Pour tout terme t et pour toute formule ϕ du langage de Z^{sk}, on a :*

$$IZ^{sk} \ \vdash \ (z \in^\circ t) \Leftrightarrow z \in t \qquad \textit{et} \qquad IZ^{sk} \ \vdash \ \phi^\circ \Leftrightarrow \phi$$

Si en outre ϕ est exprimée dans le langage de la théorie des ensembles (où les variables sont les seuls termes), alors : $IZ \ \vdash \ \phi^\circ \Leftrightarrow \phi$.

Il reste deux résultats intermédiaires[14] à établir avant de pouvoir démontrer que la traduction d'une formule prouvable dans (I)Z^{sk} (à travers la procédure de déskolémisation) est prouvable dans (I)Z.

[13] On notera la similarité conceptuelle entre le design de la procédure de déskolémisation pour les termes ($z \in^\circ t$) et les notions de réalisabilité et de forcing ($t \Vdash \phi$).

[14] Ces résultats intermédiaires ne sont pas mentionnés dans [76] faute de place. Ceux-ci donnent pourtant des indications très intéressantes sur la façon dont la structure des preuves est réorganisée au cours du processus de déskolémisation.

Il faut tout d'abord montrer que chaque terme définissable dans la syntaxe de Z^{sk} existe en tant qu'ensemble dans la théorie de Zermelo :

Lemme 1 (Collection) — *Pour tout terme t de Z^{sk} on a :*

$$\mathrm{IZ} \ \vdash \ \exists x \, \forall z \ (z \in x \Leftrightarrow z \in^{\circ} t) \qquad (x, z \notin FV(t))$$

Preuve. La preuve se fait par induction sur t, en utilisant à chaque étape l'axiome de Zermelo correspondant (sous forme existentielle). □

Grâce à ce résultat, on démontre ensuite un résultat de substitutivité vis-à-vis de la déskolémisation :

Lemme 2 (Substitutivité) — *Pour toute formule ϕ et pour tous termes t et u de Z^{sk} on a les équivalences :*

1. $\mathrm{IZ} \ \vdash \ y \in^{\circ} t\{x := u\} \ \Leftrightarrow \ \exists x \ (y \in^{\circ} t \ \wedge \ \forall z \ (z \in x \Leftrightarrow z \in^{\circ} u))$ *(x frais)*
2. $\mathrm{IZ} \ \vdash \ (\phi\{x := u\})^{\circ} \ \Leftrightarrow \ \exists x \ (\phi^{\circ} \ \wedge \ \forall z \ (z \in x \Leftrightarrow z \in^{\circ} u))$ *(x frais)*

Preuve. On démontre d'abord par induction mutuelle sur t et u que :

1. $\mathrm{IZ} \ \vdash \ \forall x \ (\forall z \ (z \in x \Leftrightarrow z \in^{\circ} u) \ \Rightarrow \ \forall z \ (z \in^{\circ} t\{x := u\} \Leftrightarrow z \in^{\circ} t))$
2. $\mathrm{IZ} \ \vdash \ \forall x \ (\forall z \ (z \in x \Leftrightarrow z \in^{\circ} u) \ \Rightarrow \ (\phi\{x := u\}^{\circ} \Leftrightarrow \phi))$

Les deux équivalences recherchées découlent des deux implications ci-dessus combinées avec le lemme 1. □

(On notera que les démonstrations des formules mentionnées dans les items 1 et 2 de ce lemme font intervenir les axiomes existentiels de Zermelo.)

Ceci nous permet alors de démontrer que la procédure de déskolémisation induit une transformation des preuves de $(\mathrm{I})\mathrm{Z}^{\mathrm{sk}}$ dans (I)Z :

Proposition 10 (Correction) — *Si une formule close ϕ est un théorème de $(\mathrm{I})\mathrm{Z}^{\mathrm{sk}}$, alors ϕ° est un théorème de* (I)Z.

Preuve. Par induction sur la structure de la dérivation. □

Il est intéressant de noter qu'au cours de la transformation d'une preuve de ϕ dans $(\mathrm{I})\mathrm{Z}^{\mathrm{sk}}$ en une preuve de ϕ° dans (I)Z :

- Les axiomes d'égalité et d'extensionnalité sont traduits en une combinaison de ces mêmes axiomes.
- Les axiomes de Z^{sk} qui définissent les constructions syntaxiques de Z^{sk} (i.e. les formes skolémisées des axiomes existentiels de Zermelo) disparaissent purement et simplement, et sont remplacés par une combinaison des axiomes d'égalité et d'extensionnalité. Aucune instance d'un axiome existentiel de Z n'intervient à cet endroit.
- En revanche, les règles de déduction logique qui font intervenir une substitution (∀-élim , ∃-intro) font apparaître en cours de traduction des instances d'axiomes existentiels de Zermelo, et ce à travers le lemme 2.

Autrement dit, la transformation d'une preuve de ϕ dans $(\mathrm{I})\mathrm{Z}^{\mathrm{sk}}$ en une preuve de ϕ° dans (I)Z réorganise la structure de la démonstration en poussant les axiomes de Zermelo vers les nœuds de la démonstration qui font intervenir une substitution (∀-elim et ∃-intro).

Des propositions 9 (dernière équivalence) et 10 on déduit alors que :

Proposition 11 (Conservativité) — *Le système* (I)Z^{sk} *est une extension conservative de* (I)Z.

2.3.6 Une forme faible de remplacement dans Z^{sk}

Historiquement, l'une des principales motivations qui ont conduit Fraenkel et Skolem à introduire le schéma de remplacement

$$\text{(Remplacement)} \qquad \forall u\ \big(\forall x \in u\ \exists! y\ \phi(x,y)\ \Rightarrow\ \exists v\ \forall x \in u\ \exists y \in v\ \phi(x,y)\big)$$

était de justifier l'existence de l'ensemble désigné par la notation $\{t(x)\ :\ x \in u\}$, à savoir l'image de l'ensemble u par la relation fonctionnelle $\phi(x,y)$, en notant $t(x)$ l'unique ensemble y tel que $\phi(x,y)$ pour $x \in u$.

De manière inattendue, l'étude de Z^{sk} révèle que la notation $\{t(x)\ :\ x \in u\}$ n'a pas besoin du schéma de remplacement pour être justifiée dans la mesure où le terme $t(x)$ est construit dans le langage de Z^{sk}.

La raison en est que pour chaque terme u de Z^{sk} et pour chaque terme $t(x)$ de Z^{sk} qui dépend (éventuellement) d'une variable x, il est possible de définir un terme noté $\mathsf{B}(t(x),\ x \in u)$ qui borne uniformément le terme $t(x)$ lorsque x décrit u. Formellement, ce terme est défini par induction structurelle sur $t(x)$ en posant :

$$\begin{array}{lcl}
\mathsf{B}(x,\ x \in u) & = & u \\
\mathsf{B}(y,\ x \in u) & = & \mathfrak{P}(y) \qquad (\text{si } y \not\equiv x) \\
\mathsf{B}(\omega,\ x \in u) & = & \mathfrak{P}(\omega) \\
\mathsf{B}(\{t_1;t_2\},\ x \in u) & = & \mathfrak{P}\big(\mathsf{B}(t_1,\ x \in u) \cup \mathsf{B}(t_2,\ x \in u)\big) \\
\mathsf{B}(\mathfrak{P}(t),\ x \in u) & = & \mathfrak{P}(\mathfrak{P}(\bigcup \mathsf{B}(t,\ x \in u))) \\
\mathsf{B}(\bigcup t,\ x \in u) & = & \mathfrak{P}(\bigcup\bigcup \mathsf{B}(t,\ x \in u)) \\
\mathsf{B}(\{y \in t\ :\ \phi\},\ x \in u) & = & \mathfrak{P}(\bigcup \mathsf{B}(t,\ x \in u))
\end{array}$$

Lemme 3 — *Pour tous termes $t(x)$ et u de Z^{sk} tels que $x \notin FV(u)$:*

$$\mathrm{IZ}^{\mathrm{sk}}\ \vdash\ \forall x\ (x \in u \Rightarrow t(x) \in \mathsf{B}(t(x),\ x \in u))\,.$$

En posant $\{t(x)\ :\ x \in u\}\ \equiv\ \{y \in \mathsf{B}(t(x),\ x \in u)\ :\ \exists x\ (x \in u \wedge y = t(x))\}$, on vérifie alors que :

Proposition 12 — *Pour tous termes $t(x)$ et u de Z^{sk} tels que $x \notin FV(u)$ et $y \notin FV(t)$ on a :*

$$\mathrm{IZ}^{\mathrm{sk}}\ \vdash\ \forall y\ \big(y \in \{t(x)\ :\ x \in u\}\ \Leftrightarrow\ \exists x\ (x \in u \wedge y = t(x))\big)\,.$$

Une conséquence importante de ce résultat est qu'on peut maintenant définir dans le langage de Z^{sk} les notations pour l'abstraction et le produit cartésien généralisé (en théorie des ensembles) qui sont des ingrédients essentiels dans la traduction d'un système de types vers la théorie des ensembles :

$$\begin{array}{rcl}
\lambda x \in t\,.\,u(x) & \equiv & \big\{\langle x, u(x)\rangle\ :\ x \in t\big\} \\
\prod_{x \in t} u(x) & \equiv & \Big\{f \in \big(\bigcup\{u(x)\ :\ x \in t\}\big)^t\ :\ \forall x\ (x \in t \Rightarrow f(x) \in u(x))\Big\}
\end{array}$$

(Ces notations ne sont pas des macros, mais désignent le résultat de transformations relativement complexes dans le langage des termes de Z^{sk}.)

2.3.7 La rétraction de λZ dans $\mathrm{IZ}^{\mathrm{sk}} + \mathrm{AFA}$

On définit à présent une traduction réciproque $(\_)^\dagger$ de λZ vers $\mathrm{IZ}^{\mathrm{sk}} + \mathrm{AFA}$, en utilisant la correspondance habituelle entre les concepts de la théorie des types et ceux de la théorie des ensembles [103, 2]. On remarquera qu'ici, on a besoin de l'antifondation pour justifier l'existence de l'ensemble HF des ensembles héréditairement finis (i.e. V_ω) [59], qu'on utilise ici pour traduire la sorte $\Box_1$. [15]

La traduction $M \mapsto M^\dagger$ Les termes-objets (non typés) de λZ (Fig. 2.3) sont traduits en des termes de Z^{sk} de la manière suivante :

$$\begin{array}{rcl}
x^\dagger & \equiv & x \\
*^\dagger & \equiv & \mathfrak{P}(\{\bullet\}) \\
\Box_1^\dagger & \equiv & \mathsf{HF} \\
\Box_2,\ \Box_3 & & \text{pas de traduction} \\
(\Pi x : T\,.\,U)^\dagger & \equiv & \prod_{x \in T^\dagger} U^\dagger \\
(\lambda x : T\,.\,M)^\dagger & \equiv & \lambda x \in T^\dagger\,.\,M^\dagger \\
(MN)^\dagger & \equiv & M^\dagger(N^\dagger) \\
(A \Rightarrow B)^\dagger & \equiv & \{\pi \in \{\bullet\} \ :\ \bullet \in A^\dagger \Rightarrow \bullet \in B^\dagger\} \\
\forall x : \Box_2\,.\,A & \equiv & \{\pi \in \{\bullet\} \ :\ \forall x\ (\bullet \in A^\dagger)\} \\
\forall x : T\,.\,A & \equiv & \{\pi \in \{\bullet\} \ :\ \forall x\ (x \in T^\dagger \Rightarrow \bullet \in A^\dagger)\}
\end{array}$$

Cette traduction est partielle, et n'associe aucun terme aux sortes $\Box_2$ et $\Box_3$. (Cependant on a $FV(M^\dagger) = FV(M)$ chaque fois que $M^\dagger$ est défini.[16])

Les propositions sont interprétées par les sous-ensembles du singleton $\{\bullet\}$, où $\bullet$ désigne n'importe quel terme clos de Z^{sk}. On utilise ici l'astuce standard qui consiste à représenter n'importe quelle formule (intuitionniste) ϕ de Z^{sk} par l'ensemble $\hat{\phi} \subset \{\bullet\}$ défini par $\hat{\phi} = \{\pi \in \{\bullet\} \ :\ \phi\}$ tandis qu'en retour, chaque sous-ensemble $p \subset \{\bullet\}$ se « décode » en la formule $\bullet \in p$. Cette correspondance entre les propositions et les sous-ensembles de $\{\bullet\}$ est bien évidemment bijective[17], en ce sens que l'équivalence $\phi \Leftrightarrow (\bullet \in \hat{\phi})$ est prouvable dans $\mathrm{IZ}^{\mathrm{sk}}$ (pour toute formule ϕ), de même que l'implication $p \subset \{\bullet\} \Rightarrow (p = \widehat{\bullet \in p})$.

À travers ce codage des propositions, les différentes formes de quantification universelle $\forall x : T\,.\,A$ sont interprétées exactement comme nous l'avons annoncé à la section 2.3.1. En particulier, les quantifications universelles de la forme $\forall x : \Box_2 \,....$ sont traitées à part, pour être traduites comme des quantifications non bornées (au sens ensembliste).

Chaque contexte de typage Γ de λZ est traduit dans Z^{sk} par une formule notée $\Gamma^\dagger$ et définie par :

$$\begin{array}{lcllcll}
([])^\dagger & \equiv & \top & (\Gamma;\ x : T)^\dagger & \equiv & \Gamma^\dagger \wedge (x \in T^\dagger) & (\text{si } T \not\equiv \Box_2) \\
(\Gamma;\ x : \Box_2)^\dagger & \equiv & \Gamma^\dagger & (\Gamma,\ \xi : A)^\dagger & \equiv & \Gamma^\dagger \wedge (\bullet \in A^\dagger) &
\end{array}$$

[15] Rappelons que l'existence de l'ensemble des ensembles héréditairement finis ne peut pas être justifiée dans IZ ([59], p. 238, exercice 10). Ce n'est plus le cas si on étend le système avec AFA, puisque HF peut être reconstruit comme la réification d'un graphe pointé universel (à support dénombrable) dont la racine pointe sur tous les arbres finis.

[16] La traduction fait également intervenir une constante supplémentaire HF qui ne fait pas partie du langage de Z^{sk}. Cependant, il est facile d'éliminer cette constante en bout de course en la remplaçant par une variable qu'on élimine avec la règle $\exists$-elim combinée avec la preuve d'existence de l'ensemble des ensembles héréditairement finis (conséquence de AFA). On ne traitera pas ce problème explicitement ici pour ne pas alourdir davantage les notations.

[17] Classiquement, c'est encore plus évident puisque $\mathfrak{P}(\{\bullet\}) = \{\varnothing; \{\bullet\}\}$.

On notera que $FV(\Gamma^\dagger) \subseteq FV(\Gamma)$. En particulier, les variables de termes de preuves sont systématiquement effacées, de même que les variables de termes-objets déclarées avec le type $\Box_2$.

Proposition 13 (Correction du typage) — *Pour tout jugement dérivable de λZ de la forme $\Gamma \vdash M : T$ où M et T sont des termes-objets tels que T n'est ni $\Box_2$ ni $\Box_3$, on a :*

$$\mathrm{IZ}^{\mathrm{sk}} + \mathrm{AFA} \ \vdash \ \Gamma^\dagger \ \Rightarrow \ M^\dagger \in T^\dagger$$

(On notera qu'ici, l'axiome d'antifondation AFA ne sert qu'à justifier l'existence de HF, l'ensemble des ensembles héréditairement finis.)

Proposition 14 (Correction des preuves) — *Pour tout jugement dérivable de λZ de la forme $\Gamma \vdash t : A$ où t est un terme de preuve et A une proposition, on a :*

$$\mathrm{IZ}^{\mathrm{sk}} + \mathrm{AFA} \ \vdash \ \Gamma^\dagger \ \Rightarrow \ \bullet \in A^\dagger$$

(Combiné avec la Prop. 7 et les résultats présentés dans [33], la proposition ci-dessus permet déjà de conclure que les théories des ensembles (I)Z, (I)Z + AFA, (I)Z + AFA + TC et le système λZ sont équiconsistants.)

2.3.8 Composition des deux traductions

La traduction $(\_)^*$ de IZ vers λZ exprime chaque formule de la théorie des ensembles en termes de graphes pointés en remplaçant chaque variable x (de la théorie des ensembles) par trois variables $\bar{x} : \Box_2$, $\tilde{x} : \bar{x} \to \bar{x} \to *$ et $\dot{x} : \bar{x}$ qui désignent le graphe pointé représentant x.

À travers la traduction $(\_)^\dagger$, chaque graphe pointé (X, R, r) de λZ devient à son tour un graphe pointé ensembliste $\langle X^\dagger, R^\dagger, r^\dagger \rangle$, à cette (petite) différence près que la relation d'adjacence $R^\dagger$ n'est pas donnée comme un sous-ensemble de $X^\dagger \times X^\dagger$, mais comme un élément de l'espace de fonctions $X^\dagger \to X^\dagger \to \mathfrak{P}(\{\bullet\})$ qui est clairement isomorphe à l'ensemble $\mathfrak{P}(X^\dagger \times X^\dagger)$. (Pour ne pas alourdir les notations, ces deux ensembles seront identifiés dans la suite.)

Par conséquent, la composée $(\_)^{*\dagger}$ des deux traductions n'est rien d'autre que la traduction en termes de graphes pointés de la théorie des ensembles de Zermelo avec antifondation dans la théorie de Zermelo elle-même. En utilisant l'axiome d'antifondation (AFA) et l'axiome de la clôture transitive (TC) on peut refermer le diagramme de la manière suivante :

Proposition 15 (Composition) — *Soit ϕ une formule de* IZ *de variables libres $x_1, \ldots, x_n$. Si $y_1, \ldots, y_n$ sont des variables fraîches telles que les variables $x_1, \ldots, x_n, \bar{y}_1, \ldots, \bar{y}_n, \tilde{y}_1, \ldots, \tilde{y}_n$ et $\dot{y}_1, \ldots, \dot{y}_n$ sont deux-à-deux distinctes, alors :*

$$\mathrm{IZ} + \mathrm{AFA} + \mathrm{TC} \ \vdash \ \Big(\bigwedge_{i=1}^{n} \mathrm{Reif}\big((\bar{y}_i, \tilde{y}_i, \dot{y}_i),\ x_i\big) \Big) \ \Rightarrow \ (\phi \ \Leftrightarrow \ \bullet \in \phi\{\vec{x} := \vec{y}\}^{*\dagger})$$

Dans le cas où ϕ est une formule close de la théorie des ensembles, l'équivalence $\phi \Leftrightarrow \bullet \in \phi^{*\dagger}$ est donc prouvable dans IZ + AFA + TC. Par conséquent :

Théorème 1 (Conservativité) — *À travers la traduction $\phi \mapsto \phi^*$, λZ est une extension conservative de* IZ + AFA + TC.

2.4 La théorie de Zermelo en déduction modulo

Parallèlement à mon travail sur la représentation de la théorie des ensembles de Zermelo dans λZ, Gilles Dowek et moi-même avons entrepris de transposer le résultat au cadre de la déduction modulo. (Ce travail a fait l'objet d'un article [31] en cours de soumission.)

La déduction modulo [32] est une extension de la logique du premier ordre avec des règles de calcul traitées en tant que telles — et de manière transparente — au niveau du système de déduction. Techniquement, une théorie en déduction modulo se présente comme une théorie du premier ordre enrichie avec des règles de réécriture au niveau des symboles de fonction et des symboles de prédicat, et dont le système de déduction est modifié de manière à permettre l'identification des formules convertibles partout dans les démonstrations. (On retrouve ainsi l'esprit de la règle de conversion des théories des types de Martin-Löf [67, 68, 69, 83], transposée au cadre de la logique du premier ordre.)

Si d'un point de vue logique la relation de conversion $A \cong A'$ induite par les règles de réécriture n'exprime rien d'autre qu'une équivalence logique $A \Leftrightarrow A'$, elle a l'avantage (contrairement aux axiomes de la théorie) de ne pas perturber le processus de contraction des radicaux dans les preuves. Tout l'enjeu de la déduction modulo consiste à déterminer dans quelle mesure les axiomes d'une théorie du premier ordre donnée peuvent être remplacés par des règles de réécriture, et, dans le cas où on a réussi à tous les faire disparaître, de déterminer si le système de preuves qui en découle vérifie la propriété de normalisation forte.

À la suite de Crabbé [26], Dowek avait déjà remarqué que si on remplaçait naïvement l'équivalence formulée dans le schéma de compréhension par la règle de réécriture

$$t \in \{x \in u \ : \ A\} \quad \rightarrow \quad t \in u \wedge A\{x := t\}\,,$$

on perdait toute propriété de normalisation au niveau des termes de preuves. (Le contre-exemple est basé sur une adaptation du paradoxe de Russell dans le cadre d'une théorie cohérente ou supposée l'être.) Il s'agissait donc d'utiliser le paradigme « ensemble = graphe pointé » pour définir un système de réécriture (moins naïf) sur les formules de la théorie des ensembles de manière à capturer la théorie de Zermelo tout en préservant la normalisation forte.

2.4.1 Description du système

De fait, le système présenté dans [31] constitue un codage d'une certaine théorie des graphes pointés (correspondant en fait à une certaine extension de la théorie de Zermelo) en déduction modulo, qui vérifie la propriété de normalisation forte.

On y distingue essentiellement deux sortes d'objets : d'une part les graphes pointés destinés à représenter des ensembles et d'autre part les nœuds sur lesquels s'appuie la structure de ces graphes pointés, les uns et les autres étant reliés par une relation ternaire $x\ \eta_z\ y$ exprimant que deux nœuds x et y sont connectés par un arc dans le graphe pointé z. (On dispose également d'un symbole de fonction permettant de récupérer la racine d'un graphe pointé.) Contrairement aux représentations basées sur un système de types dépendants, les nœuds ne sont pas ici attachés à un graphe particulier, et un même nœud peut être impliqué dans la construction d'un grand nombre de graphes pointés différents.

Les bissimulations (qui permettent d'exprimer l'égalité extensionnelle entre deux ensembles représentés sous forme de graphes pointés) sont elles-mêmes les objets d'une nouvelle sorte : la sorte des *relations binaires sur les nœuds*, c'est-à-dire, la sorte des classes de couples de nœuds. Cependant, comme le langage n'offre aucun moyen pour relativiser une quantification à tous les nœuds d'un graphe pointé donné, ces relations binaires doivent porter sur tous les nœuds de l'univers, et pas seulement sur les nœuds supportés par un ensemble donné. Aussi le système de classes qui en découle est-il un système de classes propres, ce qui explique que la preuve de normalisation s'effectue non pas relativement à la théorie des ensembles de Zermelo, mais relativement à la théorie des ensembles de Zermelo du second ordre, qui est strictement plus forte que la théorie des ensembles de Zermelo. (Un phénomène analogue se produit pour le schéma de compréhension, dont l'interprétation requiert également l'introduction d'un système de classes propres sur les nœuds.)

Pour compenser ce décrochage (en termes de force logique) entre la théorie source et la théorie cible, on introduit des restrictions syntaxiques sur la formation des classes[18] de façon à garantir que la théorie des ensembles exprimée à travers la représentation par les graphes pointés corresponde effectivement à la théorie des ensembles de Zermelo. De fait, le système qui en résulte constitue bien une extension conservative de IZ + TC + SE (où SE désigne le schéma d'*extensionnalité forte*, une forme affaiblie de l'axiome d'antifondation[19]), même si le modèle de normalisation correspondant est construit relativement à une théorie plus forte (la théorie des ensembles de Zermelo du second ordre).

Le système présenté est relativement lourd : il comporte 4 sortes d'objets, 28 symboles de fonction et de prédicat, 3 familles infinies de symboles de fonction (pour construire les deux sortes de classes et les ensembles définis par compréhension), et 36 règles de réduction. Cependant (contrairement au système λZ) il reste suffisamment proche de la théorie des ensembles pour qu'on puisse utiliser la propriété de normalisation forte pour déduire les deux propriétés du témoin attendues, à savoir la propriété du témoin numérique et la propriété du témoin généralisée (cf section 2.2.6) pour IZ + TC + SE.[20]

2.5 Un système de types pour IZF_C

À la suite des résultats présentés dans les sections 2.2 et 2.3, il est naturel de chercher à définir un système de types capturant exactement le système IZF_C (la théorie des ensembles de Zermelo-Fraenkel intuitionniste avec schéma de collection). Dans cette direction, je suis parti d'une extension de la logique d'ordre supérieur, le *système V*, et j'ai démontré qu'une extension particulière de ce système — le système $V + \mathsf{Nat} + (D)$ (où D désigne le *schéma de domination*, présenté à la section 2.5.4) — capture très exactement les formules prouvables

[18] Ces restrictions sur les classes sont similaires à celles qu'on trouve dans la théorie des ensembles von Neumann-Bernays-Gödel (NBG) [73].

[19] Formellement, SE est le schéma d'axiomes qui exprime que si une relation $\phi(x, y)$ définit une bissimulation entre l'univers des ensembles (muni de la relation d'appartenance) et lui-même, alors pour tous x et y tels que $\phi(x, y)$ on a $x = y$. On démontre aisément (à l'aide de TC) que ce schéma est équivalent au volet « unicitaire » de l'axiome d'antifondation, à savoir : tout graphe pointé admet au plus une réification.

[20] Je conjecture cependant qu'on pourrait faire la même chose avec le système λZ, même si la preuve risque d'être beaucoup plus technique.

dans la théorie IZF_C + AFA à travers la traduction $(\_)^*$.

Les résultats présentés dans cette section ne sont pas encore publiés.

2.5.1 Une extension de la logique d'ordre supérieur

Le système V est défini à partir du PTS λV dont les sortes, axiomes et règles sont donnés par

$$\begin{array}{rcl} \mathscr{S} & = & \{*;\ \Box;\ \triangle\} \\ \mathscr{A} & = & \{(* : \Box);\ (\Box : \triangle)\} \\ \mathscr{R} & = & \{(*,*,*);\ (\Box,*,*);\ (\triangle,*,*);\ (\Box,\Box,\Box)\} \end{array}$$

Il y a plusieurs manières de comprendre le PTS λV. On peut tout d'abord le voir comme le PTS obtenu à partir de $F\omega.2$ (cf section 2.2) en retirant les règles de formation de produit dépendant impliquant la sorte $\Box_2$, à savoir les règles $(\Box_1, \Box_2, \Box_2)$, $(\Box_2, \Box_1, \Box_2)$ et $(\Box_2, \Box_2, \Box_2)$. (Dans ce cadre on a $\Box = \Box_1$ et $\triangle = \Box_2$.) De manière équivalente, le PTS λV est le PTS λU (incohérent) privé de la règle $(\triangle, \Box, \Box)$ qui introduit le polymorphisme du système F au niveau des termes-objets. On peut également voir le PTS λV comme le PTS obtenu à partir du système λZ (cf section 2.3) en supprimant la sorte $\Box_1$ correspondant aux types finis[21]. (Dans ce cadre, on a donc $\Box = \Box_2$ et $\triangle = \Box_3$.)

Mais la manière la plus juste de comprendre ce système est de le voir comme la logique d'ordre supérieur (construite sur le PTS $F\omega$) étendue avec une quantification universelle sur tous les types $\alpha : \Box$[22], qu'on note par la suite $\forall \alpha \,.\, A$ (c'est-à-dire : $\forall \alpha : \Box_2 \,.\, A$), et qui correspond à travers la traduction $(\_)^*$ à la quantification (non-bornée) sur tous les ensembles.

2.5.2 Présentation stratifiée

Par définition, le système V est la présentation stratifiée du PTS λV avec des termes de preuve à la Curry. On y distingue trois catégories syntaxiques, à savoir les *types* (c'est-à-dire les genres suivant [10, 11]), les *termes-objets* (c'est-à-dire les constructeurs suivant [10, 11]) et les *termes de preuves*, dont la syntaxe est récapitulée dans la Fig. 2.6.

(On notera la présence de trois sortes de variables : les variables de type, les variables d'objets et les variables de preuves.)

Types	τ, σ	$::=$	$\alpha \mid * \mid \tau \to \sigma$
Termes-objets	M, N, A, B	$::=$	$x \mid \lambda x : \tau \,.\, M \mid M\,N$ $A \Rightarrow B \mid \forall x : \tau \,.\, A \mid \forall \alpha \,.\, A$
Termes de preuves	t, u	$::=$	$\xi \mid \lambda \xi \,.\, t \mid t\,u$

FIG. 2.6 – Syntaxe du système V

[21] La disparition de cette sorte sera compensée ultérieurement par l'introduction d'un type des entiers primitifs Nat.

[22] C'est à dire, sur tous les *genres* suivant la terminologie de [10, 11]. On notera que cette extension particulière de la logique d'ordre supérieur a déjà été envisagée dans [71].

Comme dans le système $F\omega.2$ on distingue les *signatures* (servant à déclarer les variables d'objets) des *contextes logiques* (servant à déclarer les variables de preuves), et le système de types correspondant est basé sur les trois formes de jugements $\Sigma \vdash M : \tau$ (typage des termes-objets), $\Sigma \vdash \Gamma$ ctx (bonne formation des contextes logiques) et $\langle\Sigma\rangle\ \Gamma \vdash t : A$ (typage des termes de preuves) dont les règles sont récapitulées à la Fig. 2.7. (On notera que dans cette présentation, les variables de type ne sont pas déclarées dans les signatures, et il n'est pas nécessaire d'introduire un jugement de bonne formation de signature.)

Typage des termes-objets : $\Sigma \vdash M : \tau$

$$\frac{}{\Sigma \vdash x : \tau}\ {\scriptstyle (x:\tau)\in\Sigma} \qquad \frac{\Sigma, x:\tau \vdash M : \sigma}{\Sigma \vdash \lambda x{:}\tau\,.\,M : \tau \to \sigma} \qquad \frac{\Sigma \vdash M : \tau \to \sigma \qquad \Sigma \vdash N : \tau}{\Sigma \vdash M\,N : \sigma}$$

$$\frac{\Sigma \vdash A : * \qquad \Sigma \vdash B : *}{\Sigma \vdash A \Rightarrow B : *} \qquad \frac{\Sigma, x:\tau \vdash A : *}{\Sigma \vdash \forall x{:}\tau\,.\,A : *} \qquad \frac{\Sigma \vdash A : *}{\Sigma \vdash \forall \alpha\,.\,A : *}$$

Formation des contextes logiques : $\Sigma \vdash \Gamma$ **ctx**

$$\frac{}{\Sigma \vdash \varnothing\ \text{ctx}} \qquad \frac{\Sigma \vdash \Gamma\ \text{ctx} \qquad \Sigma \vdash A : *}{\Sigma \vdash \Gamma, \xi : A\ \text{ctx}}\ {\scriptstyle \xi\notin \mathrm{dom}(\Gamma)}$$

Typage des termes de preuves : $\langle\Sigma\rangle\ \Gamma \vdash t : A$

$$\frac{\Sigma \vdash \Gamma\ \text{ctx}}{\langle\Sigma\rangle\ \Gamma \vdash \xi : A}\ {\scriptstyle (\xi:A)\in\Gamma} \qquad \frac{\langle\Sigma\rangle\ \Gamma \vdash t : A}{\langle\Sigma\rangle\ \Gamma \vdash t : A'}\ {\scriptstyle A =_\beta A'}$$

$$\frac{\langle\Sigma\rangle\ \Gamma, \xi : A \vdash t : B}{\langle\Sigma\rangle\ \Gamma \vdash \lambda\xi\,.\,t : A \Rightarrow B} \qquad \frac{\langle\Sigma\rangle\ \Gamma \vdash t : A \Rightarrow B \qquad \langle\Sigma\rangle\ \Gamma \vdash u : A}{\langle\Sigma\rangle\ \Gamma \vdash tu : B}$$

$$\frac{\langle\Sigma, x:\tau\rangle\ \Gamma \vdash t : A}{\langle\Sigma\rangle\ \Gamma \vdash t : \forall x{:}\tau\,.\,A}\ {\scriptstyle x\notin FV(\Gamma)} \qquad \frac{\Sigma \vdash N : \tau \qquad \langle\Sigma\rangle\ \Gamma \vdash t : \forall x{:}\tau\,.\,A}{\langle\Sigma\rangle\ \Gamma \vdash t : A\{x := N\}}$$

$$\frac{\langle\Sigma\rangle\ \Gamma \vdash t : A}{\langle\Sigma\rangle\ \Gamma \vdash t : \forall \alpha\,.\,A}\ {\scriptstyle \alpha\notin TV(\Sigma,\Gamma)} \qquad \frac{\langle\Sigma\rangle\ \Gamma \vdash t : \forall \alpha\,.\,A}{\langle\Sigma\rangle\ \Gamma \vdash t : A\{\alpha := \tau\}}$$

FIG. 2.7 – Règles de typage du système V

2.5.3 Traduction de la théorie de Zermelo

Grâce à traduction $\phi \mapsto \phi^*$ définie à la section 2.1.2, le système V permet d'interpréter la théorie de Zermelo intuitionniste sans axiome de l'infini (notée IZ^-), ainsi que les axiomes TC (clôture transitive) et AFA (antifondation) :

Proposition 16 — *À travers la traduction* $(\_)^*$*, le système* V *est une extension*

conservative de $\mathrm{IZ}^- + \mathrm{TC} + \mathrm{AFA}$.

Preuve. La conservativité du système V sur $\mathrm{IZ}^- + \mathrm{TC} + \mathrm{AFA}$ est établie en appliquant la même méthode que celle qui est décrite à la section 2.3. □

Il est intéressant de noter que la théorie IZ^- est équiconsistante à l'arithmétique de Heyting, et qu'il s'agit même d'une extension conservative de celle-ci (à travers le codage dû à von Neumann). Ce qui me permet de conjecturer que :

Conjecture 2 — *Le système V est une extension conservative de* HA *à travers un codage approprié des entiers dans le système V.*

Un type des entiers primitifs Le système $V + \mathsf{Nat}$ est le système V enrichi avec une constante de type Nat et des constantes de termes $\mathsf{0}$, s, pred et null munis des règles de réécriture décrites à la section 2.2.3. (Comme d'habitude, ces règles supplémentaires sont prises en charge au niveau de la règle de conversion dans le système de preuve.) On montre alors que :

Proposition 17 — *À travers la traduction* $(\_)^*$*, le système* $V + \mathsf{Nat}$ *est une extension conservative de* $\mathrm{IZ} + \mathrm{TC} + \mathrm{AFA}$.

2.5.4 Le schéma de domination

Il s'agit à présent de définir dans le système $V + \mathsf{Nat}$ un principe qui permette de capturer exactement la force du schéma de collection en logique intuitionniste. La difficulté est double, car ce principe doit être suffisamment fort pour impliquer le schéma de collection à travers la traduction $(\_)^*$ de la théorie des ensembles vers le système V ; mais il doit rester suffisamment faible pour qu'à travers la traduction réciproque $(\_)^\dagger$ du système V vers la théorie des ensembles (définie de manière similaire à la section 2.3.7) on puisse dériver ce principe à partir du schéma de collection.

Pour cela, on considère une proposition $A(x, \beta)$ du système V dépendant (éventuellement) d'une variable d'objet $x : \tau$ et d'une variable de type β, qui exprime une relation binaire entre l'objet $x : \tau$ et le type β. On dit que cette proposition est *monotone* par rapport à β si pour tous types β et β' tels qu'il existe une injection de β dans β', $A(x, \beta)$ entraîne $A(x, \beta')$. Le *schéma de domination* (D) exprime que si la relation $A(x, \beta)$ est monotone par rapport à β et si pour tout $x : \tau$ il existe un type β tel que $A(x, \beta)$, alors il existe un type β tel que pour tout $x : \tau$ on ait $A(x, \beta)$. Formellement :

$$(D)\quad (\forall x : \tau \,.\, \mathrm{mon}\, \beta \,.\, A(x, \beta)) \;\Rightarrow\; (\forall x : \tau \,.\, \exists \beta \,.\, A(x, \beta)) \;\Rightarrow\; \exists \beta \,.\, \forall x : \tau \,.\, A(x, \beta)$$

avec les abréviations

$$\begin{array}{lcl} x =_\beta y & \equiv & \forall z : (\beta \to *) \,.\, z\, x \Rightarrow z\, y \\ \exists \beta \,.\, A & \equiv & \forall z : * \,.\, ((\forall \beta \,.\, A) \Rightarrow z) \Rightarrow z \\ \mathrm{inj}(\beta, \beta', f) & \equiv & \forall x, y : \beta \,.\, f\, x =_{\beta'} f\, y \Rightarrow x =_\beta y \\ \mathrm{mon}\, \beta \,.\, A & \equiv & \forall \beta, \beta' \,.\, \forall f : (\beta \to \beta') \,.\, \mathrm{inj}(\beta, \beta', f) \Rightarrow A \Rightarrow A\{\beta := \beta'\} \end{array}$$

Intuitivement, le type β dont l'existence est donnée par la conclusion du schéma de domination (D) « domine » — au sens de la cardinalité — au moins un type β_x tel que $A(x, \beta_x)$ pour chaque $x : \tau$ (d'où $A(x, \beta)$ par monotonie).

Il est facile de démontrer que le schéma de collection est une conséquence intuitionniste du schéma de domination à travers la représentation des ensembles par des graphes pointés dans le système V. Réciproquement, l'interprétation ensembliste du schéma de domination (à travers la traduction $(\_)^\dagger$) est une conséquence (intuitionniste) presque immédiate du schéma de collection. Ce qui nous permet de généraliser la technique de preuve décrite à la section 2.3 aux systèmes $V + \mathsf{Nat} + (D)$ et IZF_C pour démontrer que :

Proposition 18 — *À travers la traduction* $(\_)^*$*, le système* $V + \mathsf{Nat} + (D)$ *est une extension conservative de* $\mathrm{IZF}_C + \mathrm{AFA}$.

2.5.5 Réalisation du schéma de domination

Tel que nous l'avons introduit dans les lignes qui précèdent, le schéma de domination n'est pour l'instant qu'un schéma d'axiomes exprimé dans le système V, et il nous reste encore à préciser comment il est implémenté au niveau des termes de preuves.

Formellement, on définit donc le système $V + \mathsf{Nat} + (D)$ comme l'extension du système $V + \mathsf{Nat}$ avec la règle de typage (ad hoc)[23]

$$(D)\qquad \frac{\begin{array}{l}\langle\Sigma\rangle\ \Gamma \vdash t_1 : \forall x:\tau\,.\,\mathrm{mon}\,\beta\,.\,A\\ \langle\Sigma\rangle\ \Gamma \vdash t_2 : \forall x:\tau\,.\,\exists\beta\,.\,A\\ \langle\Sigma\rangle\ \Gamma \vdash u : \forall\beta\,.\,((\forall x:\tau\,.\,A) \Rightarrow B)\end{array}}{\langle\Sigma\rangle\ \Gamma \vdash u\,(t_2\,(t_1\,\mathbf{I})) : B}\ {\scriptstyle \beta\notin TV(B)}$$

où $\mathbf{I}$ désigne ici le terme de preuve identité. L'intérêt immédiat de cette règle de typage est de permettre la construction d'un terme de preuve clos

$$\begin{array}{l}\lambda\xi_1\xi_2\xi_3\,.\,\xi_3\,(\xi_2\,(\xi_1\,\mathbf{I}))\quad :\\ \qquad(\forall x:\tau\,.\,\mathrm{mon}\,\beta\,.\,A) \Rightarrow (\forall x:\tau\,.\,\exists\beta\,.\,A) \Rightarrow \exists\beta\,.\,\forall x:\tau\,.\,A\end{array}$$

pour toutes les instances du schéma de domination, et le lecteur vérifiera sans peine que la règle (D) a exactement le même contenu logique (en faisant abstraction des termes de preuves) que le schéma de domination lui-même.

2.5.6 La preuve de normalisation forte

Bien entendu, le terme de preuve $u\,(t_2\,(t_1\,\mathbf{I}))$ qui figure dans la conclusion de la règle (D) n'a pas été choisi au hasard, puisqu'il s'agit en réalité d'un terme dont on peut justifier l'existence dans un modèle de normalisation adéquat, basé sur les ensembles saturés de Tait.

Plutôt que de construire le modèle de normalisation forte du système $V + \mathsf{Nat} + (D)$ dans un ZF-univers de la forme V_μ où μ est un cardinal inaccessible, on construit le modèle de normalisation dans l'univers tout entier, ce qui nous permet de nous passer de l'hypothèse d'inaccessibilité (et de la logique classique) et d'obtenir ainsi une preuve de normalisation forte relativement à IZF_C.

[23] Comme pour la règle de typage des preuves associées à l'opérateur ε de Hilbert intuitionniste dans le système $F\omega.2^{++}$ (cf section 2.2.5), la règle de typage présentée ici a toutes les chances de casser la propriété de *subject-reduction* au niveau des termes de preuves. Une fois de plus, il semble que ce soit le prix à payer pour obtenir toute la force du schéma de collection. (Voir à ce sujet la discussion de la section 2.2.6.)

Formellement on se donne comme théorie cible le système $\mathrm{IZF}_C^{\mathrm{sk}}$, obtenu en ajoutant à $\mathrm{IZ}^{\mathrm{sk}}$ (cf section 2.3.4) toutes les instances du schéma de collection (sans aucune forme de skolémisation). Bien entendu, ce système est une extension conservative de IZF_C, et son seul intérêt est de nous donner toutes les notations nécessaires pour définir les traductions syntaxiques. Dans ce qui suit, on suppose fixée une représentation des λ-termes purs en théorie des ensembles. Avec une telle représentation, chaque terme *ouvert* t du λ-calcul pur est représenté par un terme *clos* noté $\overline{t}$ dans le langage de $\mathrm{IZF}_C^{\mathrm{sk}}$. On désigne par $\mathbf{\Lambda}$ (resp. $\mathbf{X}$, $\mathbf{SN}$, $\mathbf{SAT}$) n'importe quel terme clos de $\mathrm{IZF}_C^{\mathrm{sk}}$ qui représente l'ensemble des λ-termes ouverts (resp. l'ensemble des variables, l'ensemble des λ-termes fortement normalisables, l'ensemble des ensembles saturés au sens de Tait). On a donc par définition :

$$\mathrm{IZF}_C^{\mathrm{sk}} \;\vdash\; \forall x \in \mathbf{SAT}\ (\mathbf{X} \subseteq x \wedge x \subseteq \mathbf{SN})\,.$$

Enfin, pour tout $n \geq 0$ on suppose donnée une macro $\_[\_ := \_; \ldots; \_ := \_]$ à $2n+1$ arguments (dans le langage de $\mathrm{IZF}_C^{\mathrm{sk}}$) qui représente la substitution parallèle $t\{\xi_1 := u_1; \ldots; \xi_n := u_n\}$ du λ-calcul pur, et telle que pour tous $t, \xi_1, u_1, \ldots, \xi_n, u_n$ on ait

$$\mathrm{IZF}_C^{\mathrm{sk}} \;\vdash\; \overline{t}[\overline{\xi_1} := \overline{u_1}; \ldots; \overline{\xi_n} := \overline{u_n}] = \overline{t\{\xi_1 := u_1; \ldots; \xi_n := u_n\}}$$

À chaque variable de type α (resp. à chaque variable d'objet x, à chaque variable de preuve ξ) du système V on associe une variable de la théorie des ensembles notée $\alpha^\dagger$ (resp. $x^\dagger$, $\xi^\dagger$[24]) de telle sorte que les variables-cible associées à deux variables-source distinctes soient distinctes. Plus généralement, on associe à chaque type τ (resp. à chaque terme-objet M) du système $V + \mathsf{Nat} + (D)$ un terme noté $\tau^\dagger$ (resp. $M^\dagger$) dans le langage de $\mathrm{IZF}_C^{\mathrm{sk}}$ en posant :

$$\begin{array}{rcl}
(\alpha)^\dagger & \equiv & \alpha^\dagger \\
(*)^\dagger & \equiv & \mathbf{SAT} \\
(\mathsf{Nat})^\dagger & \equiv & \omega \\
(\tau \to \sigma)^\dagger & \equiv & (\sigma^\dagger)^{(\tau^\dagger)}
\end{array}
\qquad\qquad
\begin{array}{rcl}
(x)^\dagger & \equiv & x^\dagger \\
(\lambda x\!:\!T\,.\,M)^\dagger & \equiv & \lambda x^\dagger \in T^\dagger\,.\,M^\dagger \\
(M\,N)^\dagger & \equiv & M^\dagger(N)^\dagger \\
(A \Rightarrow B)^\dagger & \equiv & A^\dagger \to B^\dagger \\
(\forall x\!:\!\tau\,.\,A)^\dagger & \equiv & \{z \in \mathbf{\Lambda} \;:\; \forall x^\dagger \in T^\dagger\ (z \in A^\dagger)\} \\
(\forall \alpha\,.\,A)^\dagger & \equiv & \{z \in \mathbf{\Lambda} \;:\; \forall \alpha^\dagger\,(\mathrm{Inh}(\alpha^\dagger) \Rightarrow z \in A^\dagger)\} \\
0^\dagger & \equiv & 0 \quad (\text{idem pour } \mathsf{s},\ \mathsf{pred},\ \mathsf{null})
\end{array}$$

où $\_ \to \_$ est la macro désignant la flèche de la réalisabilité dans le langage de $\mathrm{IZF}_C^{\mathrm{sk}}$, et où $\mathrm{Inh}(x)$ désigne la formule $\exists z\ (z \in x)$.

On montre alors les deux propositions suivantes :

Proposition 19 (Correction du système de types) — *Si le jugement*

$$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash M : \sigma$$

est dérivable dans le système $V + \mathsf{Nat} + (D)$, alors :

$$\mathrm{IZ}^{\mathrm{sk}} \;\vdash\; \Big(\bigwedge_{i=1}^{k} \mathrm{Inh}(\alpha_i^\dagger)\Big) \wedge \Big(\bigwedge_{i=1}^{n} x_i^\dagger \in \tau_i^\dagger\Big) \;\Rightarrow\; \big(M^\dagger \in \sigma^\dagger\big)$$

où $\alpha_1, \ldots, \alpha_k$ sont les variables de types libres du jugement ci-dessus.

Preuve. Par induction sur la dérivation. □

[24] On veillera à ne pas confondre la variable $\xi^\dagger$ avec le terme clos $\overline{\xi}$ de $\mathrm{IZF}_C^{\mathrm{sk}}$.

On notera que les hypothèses $\mathrm{Inh}(\alpha_i^\dagger)$ pour $1 \leq i \leq k$ sont essentielles pour garantir que les dénotations de tous les types sont habitées. En revanche, la preuve de cette proposition n'utilise jamais le schéma de collection, et peut donc s'effectuer dans le système $\mathrm{IZ}^{\mathrm{sk}}$.

Proposition 20 (Correction du système de preuves) — *Si le jugement*

$$\langle x_1 : \tau_1, \ldots, x_n : \tau_n \rangle \ \xi_1 : A_1, \ldots, \xi_m : A_m \vdash t : B$$

est dérivable dans le système $V + \mathsf{Nat} + (D)$, alors :

$$\begin{array}{rcl} \mathrm{IZF}_C^{\mathrm{sk}} & \vdash & \Big(\bigwedge_{i=1}^{k} \mathrm{Inh}(\alpha_i^\dagger)\Big) \wedge \Big(\bigwedge_{i=1}^{n} x_i^\dagger \in \tau_i^\dagger\Big) \ \Rightarrow \\ & & \qquad \forall \xi_1^\dagger \in A_1^\dagger \cdots \forall \xi_m^\dagger \in A_m^\dagger \ \big(\overline{t[\xi_1} := \xi_1^\dagger; \cdots; \overline{\xi_m} := \xi_m^\dagger] \in B^\dagger\big) \end{array}$$

où $\alpha_1, \ldots, \alpha_k$ sont les variables de types libres du jugement ci-dessus.

Preuve. Par induction sur la dérivation. On ne traite que le cas de la règle (D), en laissant de côté la signature et le contexte. Supposons (dans $\mathrm{IZF}_C^{\mathrm{sk}}$) que

$$\begin{array}{rcl} t_1 & \in & (\forall x : \tau \,.\, \mathrm{mon}\, \beta \,.\, A(x, \beta))^\dagger \\ t_2 & \in & (\forall x : \tau \,.\, \exists \beta \,.\, A(x, \beta))^\dagger \\ u & \in & (\forall \beta \,.\, ((\forall x : \tau \,.\, A(x, \beta)) \Rightarrow B))^\dagger \end{array}$$

avec $\beta \notin TV(B)$. On considère alors l'ensemble $E \subseteq \tau^\dagger \times \mathbf{\Lambda}$ défini par

$$E = \{(x^\dagger, t) \in \tau^\dagger \times \mathbf{\Lambda} \ : \ \exists \beta^\dagger \ (t \in A^\dagger(x^\dagger, \beta^\dagger))\}$$

D'après le schéma de collection, il existe un ensemble F tel que

$$\forall (x^\dagger, t) \in E \ \ \exists \beta^\dagger \in F \ \ (t \in A^\dagger(x^\dagger, \beta^\dagger))$$

ce qui nous permet de poser $\widehat{\beta} = \bigcup F$. On commence par remarquer que

$$\mathbf{I} \ \in \ \mathrm{inj}^\dagger(\beta^\dagger, \widehat{\beta}, i_{\beta^\dagger})$$

pour tout ensemble $\beta^\dagger \subseteq \widehat{\beta}$, en notant $i_{\beta^\dagger}$ l'inclusion de $\beta^\dagger$ dans $\widehat{\beta}$. Ce qui permet de démontrer que

$$t_1 \, \mathbf{I} \ \in \ \bigcap_{x^\dagger \in \tau^\dagger} \bigcap_{\beta^\dagger} \Big(A^\dagger(x^\dagger, \beta^\dagger) \to A^\dagger(x^\dagger, \widehat{\beta}) \Big)$$

(en utilisant la définition de la flèche de la réalisabilité combinée avec les propriétés des ensembles E et $\widehat{\beta}$). Comme en outre on a

$$t_2 \ \in \ \bigcap_{x^\dagger \in \tau^\dagger} \bigcap_{\beta^\dagger} \Big(A^\dagger(x^\dagger, \beta^\dagger) \to A^\dagger(x^\dagger, \widehat{\beta}) \Big) \ \to \ \bigcap_{x^\dagger \in \tau^\dagger} A^\dagger(x^\dagger, \widehat{\beta})$$

(par une relation de sous-typage immédiate) il vient

$$t_2 \, (t_1 \, \mathbf{I}) \ \in \ \bigcap_{x^\dagger \in \tau^\dagger} A^\dagger(x^\dagger, \widehat{\beta})$$

et finalement $u \, (t_2 \, (t_1 \, \mathbf{I})) \in B^\dagger$ (en instanciant β par $\widehat{\beta}$ dans le type de u). □

Ceci nous permet de conclure que :

Théorème 2 — *Si la théorie* IZF_C *est* 1*-cohérente*[25]*, alors tous les termes de preuves du système* $V + \mathsf{Nat} + (D)$ *sont fortement normalisables.*

[25]Rappelons qu'une théorie des ensembles $\mathcal{T}$ est 1-cohérente si pour tout prédicat primitif récursif $P(x)$ représenté dans le langage de $\mathcal{T}$ par une formule $\phi(x)$ dépendant d'une variable x de domaine ω, la prouvabilité de la formule $\exists x \in \omega\, \phi(x)$ dans la théorie $\mathcal{T}$ entraîne l'existence d'un entier n tel que $P(n)$. (La 1-cohérence de $\mathcal{T}$ entraîne évidemment la cohérence de $\mathcal{T}$, mais la réciproque est fausse en général.) La propriété de 1-cohérence est ici nécessaire pour déduire qu'un terme de preuve t est fortement normalisable à partir de la prouvabilité de la formule $\overline{t} \in \mathbf{SN}$ dans IZF_C. (Le prédicat primitif récursif $P(n)$ étant ici : « toute séquence de réduction issue de t est de longueur inférieure ou égale à n ».)

Chapitre 3

Le λ-calcul avec constructeurs

3.1 Motivation

Le λ-calcul avec constructeurs est né d'une volonté de mieux comprendre les interactions entre les différents modes de construction et de destruction des valeurs dans les extensions « intuitionnistes » du λ-calcul, c'est-à-dire dans les extensions du λ-calcul avec des paires primitives et/ou un mécanisme de filtrage. Une des motivations initiales de cette étude était de fournir une sémantique opérationnelle propre *dans le cas non typé* à des calculs tels que le calcul des constructions inductives ou la théorie des types de Martin-Löf, avec le secret espoir de reconstruire le système de types de manière sémantique (par exemple avec des techniques par orthogonalité) dans l'esprit de la Ludique de Girard.

J'ai défini le λ-calcul avec constructeurs dans les mois qui ont suivi mon arrivée dans le laboratoire PPS. Au cours de ma première visite à l'Université de Buenos Aires (en décembre 2004), j'ai présenté le formalisme à Alejandro Ríos et Ariel Arbiser (alors en thèse avec Alejandro), et c'est à trois que nous avons étudié le formalisme et démontré ses deux principales propriétés : le théorème de Church-Rosser modulaire et le théorème de séparation.

Ces résultats ont été publiés dans [4, 5].

3.1.1 Construction et destruction

Dans le λ-calcul pur [22], la situation est très simple puisqu'il n'y a qu'un seul mode de construction — la λ-abstraction — et qu'un seul mode de destruction — l'application. La seule façon de se faire suivre (dans cet ordre) une étape de construction et une étape de destruction est de former un β-radical, un assemblage dont le comportement calculatoire est bien compris et constitue la base de la sémantique opérationnelle du calcul.

Les choses se compliquent dès qu'on étend le λ-calcul avec d'autres modes de construction de valeurs accompagnés des mécanismes de destruction correspondants : par exemple les couples avec leurs projections, ou les constructeurs avec leur mécanisme de filtrage. Pour simplifier les choses, on ne considère ici que des constructeurs constants (correspondant aux valeurs des types énumérés) dont la destruction est assurée par un mécanisme de filtrage très simplifié correspondant à la construction `case` du langage Pascal.

La situation est résumée dans le tableau suivant :

	Construction	Destruction
Fonctions	λ-abstraction	application
Couples	formation de paire	projections
Valeurs énumérées	constructeurs constants	filtrage (`case`)

Pour se fixer les idées, on se donne un ensemble infini de constructeurs, notés c, c_1, c', etc. et on adopte (temporairement) la syntaxe suivante

Termes
$$\begin{array}{rcl} M, N & ::= & x \\ & | & \lambda x\,.\,M \quad | \quad M\,N \\ & | & \langle M_1; M_2\rangle \quad | \quad \pi_1(M) \quad | \quad \pi_2(M) \\ & | & c \quad | \quad \{\!|\theta|\!\}\,.\,M \end{array}$$

où θ est un *analyseur de cas*[1], c'est-à-dire une suite finie de la forme

Analyseur de cas
$$\theta, \phi \quad ::= \quad c_1 \mapsto M_1; \cdots; c_n \mapsto M_n$$

où les constructeurs c_i ($1 \leq i \leq n$) sont deux à deux distincts. (Il s'agit donc d'une définition par induction mutuelle.)

Dans la littérature, ce type d'extension du λ-calcul vient généralement avec une sémantique opérationnelle qui se contente de décrire pour chaque forme de valeur (ici : les fonctions, les couples et les constructeurs) l'interaction entre le mode de construction et le mode de destruction correspondant. Ce qui dans le cas qui nous occupe donne les trois règles suivantes :

$$\begin{array}{rcll} (\lambda x\,.\,M)\,N & \to & M\{x := N\} & \\ \pi_i(\langle M_1; M_2\rangle) & \to & M_i & (i = 1, 2) \\ \{\!|\theta|\!\}\,.\,c & \to & M & ((c \mapsto M) \in \theta) \end{array}$$

En revanche, la plupart des présentations s'arrêtent là[2] et passent sous silence le problème de l'interaction entre un mode de construction et un mode de destruction qui ne lui correspond pas. Quid de l'application d'un constructeur à un argument ou de la projection d'un constructeur ? de la projection d'une fonction ou d'un constructeur ? du filtrage d'une abstraction ou d'un couple ?

Il y a évidemment une bonne raison à cela. En programmation fonctionnelle, des agrégats tels que

$$\langle M_1; M_2\rangle N, \qquad \pi_1(\lambda x\,.\,M) \qquad \text{ou encore} \qquad \{\!|\theta|\!\}\,.\,(\lambda x\,.\,M)$$

sont considérés comme dénués de sens, et déclenchent à l'exécution des erreurs non rattrapables (de type : faute de mémoire). De plus, les systèmes de types statiques sont précisément conçus pour éviter que de telles situations surviennent en cours d'exécution. Dès lors, pourquoi s'occuper de situations que le vérificateur de type va rejeter de toutes façons ?

Pourtant, rejeter d'emblée de tels agrégats en invoquant le typage, c'est oublier un peu vite qu'une des raisons du succès du λ-calcul vient précisément des excellentes propriétés de sa sémantique opérationnelle dans le cas non typé. Parmi ces propriétés, l'une des plus remarquables est la propriété de séparation de Böhm, qui spécifie que deux λ-termes en forme $\beta\eta$-normale qui convergent

[1] En anglais : *case binding*.

[2] À l'exception notable du calcul de Révész [91]. Voir aussi la note 5 p. 55

face aux mêmes contextes d'évaluation sont identiques (à α-conversion près). Intuitivement, ce résultat exprime que le λ-calcul est un calcul sans redondance, dans lequel il n'y a pas moyen d'exprimer le même comportement calculatoire par des termes différents — pourvu qu'on se restreigne aux termes en forme $\beta\eta$-normale et qu'on considère tous les contextes d'évaluation, et pas seulement les contextes « bien typés ».[3]

C'est sans doute l'une des raisons qui expliquent pourquoi la propriété de séparation n'avait (à ma connaissance) jamais été considérée pour des formalismes avec filtrage avant le λ-calcul avec constructeurs, du moins dans le cas non typé. Car une telle étude présuppose en effet de définir proprement l'interaction entre tous les modes de construction et de destruction du calcul, faute de quoi apparaissent des agrégats qui bloquent le calcul et font obstacle à toute tentative de séparation.[4]

À partir de maintenant, nous allons nous restreindre au calcul sur les fonctions et les constructeurs constants, en laissant de côté les couples :

Termes	M, N	$::=$	x
		$\mid$	$\lambda x \,.\, M \quad \mid \quad M\, N$
		$\mid$	$c \quad \mid \quad \{\!\{\theta\}\!\}.\, M$
Analyseurs de cas	θ, ϕ	$::=$	$c_1 \mapsto M_1; \ldots; c_n \mapsto M_n$

(avec la contrainte habituelle que $c_i \neq c_j$ si $i \neq j$).

Avant d'aller plus loin, il est utile de mentionner que le problème de l'interaction hétérogène que nous nous proposons d'étudier ici a déjà été traité par György Révész [90, 91] dans le cadre d'un λ-calcul avec des couples primitifs.[5] Cette approche et la nôtre sont donc parfaitement complémentaires, et l'histoire d'un calcul qui décrirait l'interaction hétérogène entre les trois modes de construction/destruction reste à écrire.

3.1.2 Une machine à pile(s)

Pour mieux comprendre ce problème d'interaction hétérogène, nous allons temporairement abandonner le cadre de la *réduction* pour nous focaliser sur l'*évaluation* (de tête et en appel par nom) dans le cadre d'une machine à pile.

Dans un tel cadre, un *état* d'exécution est un couple $M \star \pi$ formé par un terme clos M (le terme de tête) et par une pile π constituée d'une liste finie de

[3] Par exemple, les termes qui implémentent le tri à bulles et le tri fusion sont extensionnellement équivalent sur les listes, ce qui signifie que dans tout contexte attendant une liste, ces deux programmes se comportent de la même manière — il n'y donc pas moyen de les distinguer. En revanche, si on accepte de tester ces deux programmes dans des contextes arbitraires, le théorème de séparation de Böhm nous dit qu'il est possible de trouver un contexte qui fera converger l'un des deux programmes tout en faisant diverger l'autre. Autrement dit, on a bien la propriété d'*extensionnalité* sur les λ-termes (en tant que fonctions), à condition de considérer tous les contextes d'évaluation, et pas seulement ceux qui sont bien typés.

[4] On verra cependant que même dans le λ-calcul avec constructeurs, il est nécessaire d'exclure un certain nombre de configurations bloquantes pour assurer la propriété de séparation. Dans le cadre conceptuel qu'on s'est fixé, le théorème de séparation du λ-calcul avec constructeurs est donc tout autant un succès qu'un échec (partiels).

[5] Le formalisme de Révész est en fait un λ-calcul avec des n-uplets primitifs muni des règles β, η, δ (incluant les règles de destruction des n-uplets) ainsi que des règles de commutation $\langle M_1; \ldots; M_n\rangle N \to \langle M_1 N; \ldots; M_n N\rangle$ et $\lambda x \,.\, \langle M_1; \ldots; M_n\rangle \to \langle \lambda x \,.\, M_1; \ldots; \lambda x \,.\, M_n\rangle$. Révész montre que le système obtenu est confluent [91], mais ne considère pas la propriété de séparation qu'il serait sans doute intéressant d'étudier dans un tel cadre.

destructeurs. Comme le calcul qui nous intéresse ne considère que deux sortes de valeurs (les fonctions et les constructeurs), on distingue deux sortes de destructeurs : les arguments de fonction (notation : N), qui sont des termes clos, et les analyseurs de cas (notation : θ), qu'on suppose clos également.

La relation d'évaluation (de tête en appel par nom), notée $M \star \pi \succ M' \star \pi'$, est alors définie à partir des quatre règles suivantes :

Empilement/ dépilement d'un argument de fonction	
$\begin{array}{rcl} M\,N & \star & \pi \\ & \vee & \\ M & \star & N \cdot \pi \end{array}$	$\begin{array}{rcl} \lambda x\,.\,M & \star & N \cdot \pi \\ & \vee & \\ M\{x := N\} & \star & \pi \end{array}$
Empilement/ dépilement d'un analyseur de cas	
$\begin{array}{rcl} \{\!\!\{\theta\}\!\!\}\,.\,M & \star & \pi \\ & \vee & \\ M & \star & \theta \cdot \pi \end{array}$	$\begin{array}{rcll} c & \star & \theta \cdot \pi & \\ & \vee & & (\text{si } (c \mapsto M) \in \theta) \\ M & \star & \pi & \end{array}$

Ainsi définie, la relation d'évaluation fait clairement apparaître deux situations de blocage correspondant à une interaction hétérogène, à savoir :

1. Le terme de tête est une abstraction tandis que le sommet de la pile contient un analyseur de cas : $\lambda x\,.\,M \star \theta \cdot \pi$
2. Le terme de tête est un constructeur tandis que le sommet de la pile contient un argument de fonction : $c \star N \cdot \pi$

On notera que dans les deux cas, l'évaluation bloque uniquement parce que le destructeur au sommet de la pile n'est pas de la même nature que la valeur en tête d'exécution. Qu'il y ait un peu plus loin dans la pile un destructeur de même nature que la valeur de tête n'y change rien...

Il y a pourtant une façon très simple de résoudre le problème, qui consiste à travailler avec les deux formes de destructeurs de manière asynchrone. Pour cela, il suffit simplement de séparer la pile d'exécution en deux piles : une pile d'analyseurs de cas (qu'on fait figurer à gauche du terme de tête) et une pile d'arguments (qu'on fait figurer à droite). Dans cette nouvelle géométrie du calcul, les règles d'évaluation deviennent alors :

Empilement/ dépilement d'un argument de fonction	
$\begin{array}{rcccl} \pi_1 & \star & M\,N & \star & \pi_2 \\ & & \vee & & \\ \pi_1 & \star & M & \star & N \cdot \pi_2 \end{array}$	$\begin{array}{rcccl} \pi_1 & \star & \lambda x\,.\,M & \star & N \cdot \pi_2 \\ & & \vee & & \\ \pi_1 & \star & M\{x := N\} & \star & \pi_2 \end{array}$
Empilement/ dépilement d'un analyseur de cas	
$\begin{array}{rcccl} \pi_1 & \star & \{\!\!\{\theta\}\!\!\}\,.\,M & \star & \pi_2 \\ & & \vee & & \\ \pi_1 \cdot \theta & \star & M & \star & \pi_2 \end{array}$	$\begin{array}{rcccll} \pi_1 \cdot \theta & \star & c & \star & \pi_2 & \\ & & \vee & & & (\text{si } (c \mapsto M) \in \theta) \\ \pi_1 & \star & M & \star & \pi_2 & \end{array}$

Comme on pouvait s'y attendre, l'utilisation de deux piles au lieu d'une a fait disparaître les situations de blocage dues à une interaction hétérogène. Il reste cependant deux situations de blocage, à savoir :

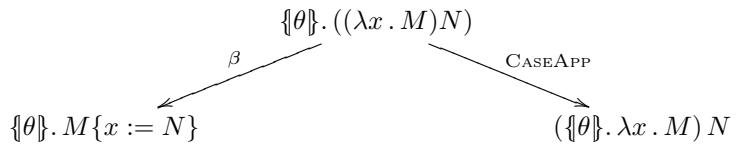
– La pile concernée est vide : soit le terme de tête est une abstraction et la pile d'arguments est vide, soit le terme de tête est un constructeur et la pile d'analyseurs de cas est vide. C'est le cas de blocage habituel par « exhaustion des ressources » (on le retrouve également dans le λ-calcul pur) et nous ne le traiterons pas ici.
– Erreur de filtrage : le terme de tête est un constructeur, et l'analyseur de cas au sommet de la pile idoine n'associe aucun terme à ce constructeur. C'est sans doute le seul cas de blocage vraiment intéressant dans un calcul avec filtrage, et c'est celui qui aura besoin d'être traité de manière spécifique lors de la preuve du théorème de séparation.

3.1.3 Retour à la réduction

La discussion qui précède fait donc apparaître une solution simple et élégante pour résoudre le problème de l'interaction hétérogène, qui est de faire commuter les différents modes de destruction entre eux (ici : l'application et l'analyse par cas). Dans un calcul basé sur une notion de réduction, cette propriété de commutation se traduit naturellement par la règle de réduction

$$(\textsc{CaseApp}) \qquad \{\!|\theta|\!\}.\,(MN) \quad \to \quad (\{\!|\theta|\!\}.\,M)N$$

On peut tout de suite noter qu'une telle règle fait apparaître une paire critique avec la règle de β-réduction :



Pour refermer cette paire critique, on doit donc également introduire une règle de commutation entre la construction *case* et l'abstraction :

$$(\textsc{CaseLam}) \qquad \{\!|\theta|\!\}.\,\lambda x\,.\,M \quad \to \quad \lambda x\,.\,\{\!|\theta|\!\}.\,M \qquad\qquad (x \notin FV(\theta))$$

Du point de vue de la machine à double pile décrite plus haut, cette dernière règle exprime que l'opération d'empilement d'un analyseur de cas (symbolisée par la construction *case*) commute avec l'opération de dépilement d'un argument (symbolisée par la λ-abstraction).

Les deux règles (CaseApp) et (CaseLam) confèrent ainsi à la construction *case* un comportement de *substitution linéaire de tête*, c'est-à-dire une forme de substitution qui traverse les abstractions et les applications — uniquement à gauche — pour se fixer sur la variable ou le constructeur de tête du programme. Dans le cas d'une variable de tête, la construction *case* reste bloquée (jusqu'au moment où la variable sera substituée) tandis que dans dans le cas d'un constructeur de tête, la construction *case* se réduit à l'aide de la règle

$$(\textsc{CaseCons}) \qquad \{\!|\theta|\!\}.\,c \quad \to \quad M \qquad\qquad ((c \mapsto M) \in \theta)$$

si le constructeur c est traité par l'analyseur de cas θ. (Sinon, le terme $\{\!|\theta|\!\}.\,c$ reste bloqué indéfiniment : c'est l'erreur de filtrage.)

Filtrage En pratique, le comportement de substitution linéaire de tête de la construction *case* permet de faire du filtrage même lorsqu'un constructeur est appliqué à des arguments. Par exemple, le programme

$$\lambda z \,.\, \{\!|c \mapsto \lambda xy \,.\, y|\!\}\,.\, z$$

permet de récupérer le deuxième argument auquel est appliqué le constructeur c ainsi que le montre la séquence de réduction suivante :

$$\begin{array}{lll} & (\lambda z \,.\, \{\!|c \mapsto \lambda xy \,.\, y|\!\}\,.\, z)\,(c\,M\,N) & \\ \to & \{\!|c \mapsto \lambda xy \,.\, y|\!\}\,.\,(c\,M\,N) & (\beta) \\ \to & (\{\!|c \mapsto \lambda xy \,.\, y|\!\}\,.\, c)\,M\,N & (\textsc{CaseApp}) \times 2 \\ \to & (\lambda xy \,.\, y)\,M\,N & (\textsc{CaseCons}) \\ \to & N & (\beta) \times 2 \end{array}$$

Ceci explique pourquoi nous n'avons considéré qu'un filtrage simple sur des constructeurs constants, dans l'esprit du `case` de Pascal. Car en présence de la règle (CaseApp), l'expressivité de ce mécanisme élémentaire est démultipliée et permet de traiter les constructeurs appliqués. Nous verrons en 3.2.3 que ce mécanisme permet non seulement d'implémenter le filtrage algébrique habituel qu'on trouve dans les langages de la famille ML (i.e. le filtrage sur des motifs algébriques linéaires), mais aussi d'utiliser les constructeurs comme des constructeurs variadiques.

Autres règles Si les règles (β), (CaseApp) et (CaseCons) suffisent en pratique pour le calcul (la règle (CaseLam) n'est même pas nécessaire), la relation de convertibilité qu'elles engendrent ne suffit pas à identifier tous les termes qui partagent un même comportement calculatoire (au sens du théorème de séparation que nous présenterons en 3.4). Comme pour le λ-calcul, il est nécessaire de considérer la règle de η-réduction

$$(\eta) \qquad \lambda x \,.\, M\,x \quad \to \quad M \qquad (x \notin FV(M)$$

ainsi qu'une nouvelle règle, pour composer les constructions *case* :

$$(\textsc{CaseCase}) \qquad \{\!|\theta|\!\}\,.\,\{\!|\phi|\!\}\,.\,M \quad \to \quad \{\!|\theta \circ \phi|\!\}\,.\,M$$

où la composition de deux analyseurs de cas θ et $\phi \equiv (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)$ est définie par

$$\theta \circ \phi \quad \equiv \quad (c_1 \mapsto \{\!|\theta|\!\}\,.\,M_1; \ldots; c_n \mapsto \{\!|\theta|\!\}\,.\,M_n)\,.$$

Le démon À l'instar de la Ludique [42], le λ-calcul avec constructeurs est muni d'un démon, noté ✠, dont la fonction est d'absorber le contexte d'évaluation dans lequel il est placé. (En cela, le démon est similaire à la fonction `exit` des langages de programmation.) Cette construction, qui vient avec trois règles de réduction spécifiques que nous introduirons formellement en 3.2.2, jouera un rôle majeur dans le théorème de séparation.

3.2 Le formalisme

3.2.1 Syntaxe

Le λ-calcul avec constructeurs distingue deux espaces de noms : les *variables* (notation : x, y, z, etc.) et les *constructeurs* (notation : c, c_1, c', etc.) Dans ce qui suit, on suppose ces deux espaces de noms dénombrables et disjoints.

Les termes (notation : M, N, etc.) et les analyseurs de cas (notation : θ, ϕ, etc.) du λ-calcul avec constructeurs sont définis par induction mutuelle à partir de la BNF suivante :

Termes	M, N	::=	x	(Variable)
		$\mid$	c	(Constructeur)
		$\mid$	$\maltese$	(Démon)
		$\mid$	MN	(Application)
		$\mid$	$\lambda x\,.\,M$	(Abstraction)
		$\mid$	$\{\!\{\theta\}\!\}\,.\,M$	(Filtrage)
Analyseurs de cas	θ, ϕ	::=	$c_1 \mapsto M_1; \ldots; c_n \mapsto M_n$	$(c_i \neq c_j \text{ si } i \neq j)$

Analyseurs de cas Un analyseur de cas θ est une liste finie de liaisons de la forme $c \mapsto M$ dont les membres gauches sont deux-à-deux distincts. On dit qu'un constructeur c est *lié* à un terme M dans un analyseur de cas θ si la liaison $c \mapsto M$ appartient à la liste θ. D'après la définition des analyseurs de cas, il est clair qu'un constructeur c est lié à au plus un terme M dans un analyseur de cas θ donné. Lorsqu'il n'existe pas de tel terme M, on dit que c *n'est pas lié* dans θ. Le domaine d'un analyseur de cas $\theta = (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)$ est défini par $\mathrm{dom}(\theta) = \{c_1, \ldots, c_n\}$.

On introduit également une opération (externe) de composition de deux analyseurs de cas θ et $\phi \equiv (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)$ en posant

$$\theta \circ \phi \quad \equiv \quad (c_1 \mapsto \{\!\{\theta\}\!\}\,.\,M_1; \ldots; c_n \mapsto \{\!\{\theta\}\!\}\,.\,M_n)\,.$$

Syntaxiquement, cette opération n'est pas associative puisque

$$\begin{array}{rcl} (\theta \circ \phi) \circ (c_i \mapsto M_i)_{i=1}^n & \equiv & (c_i \mapsto \{\!\{\theta \circ \phi\}\!\}\,.\,M_i)_{i=1}^n \\ \not\equiv \quad \theta \circ \big(\phi \circ (c_i \mapsto M_i)_{i=1}^n\big) & \equiv & (c_i \mapsto \{\!\{\theta\}\!\}\,.\,\{\!\{\phi\}\!\}\,.\,M_i)_{i=1}^n \end{array}$$

Cependant, la composition d'analyseurs de cas n'a de sens qu'en présence de la règle de réduction $\{\!\{\theta\}\!\}\,.\,\{\!\{\phi\}\!\}\,.\,M \to \{\!\{\theta \circ \phi\}\!\}\,.\,M$ (CaseCase) qui rend les deux termes ci-dessus convertibles.

Variables libres et substitution Les notions d'occurrence libre et d'occurrence liée d'une variable dans un terme ou un analyseur de cas sont définies de la manière attendue. On note $FV(M)$ (resp. $FV(\theta)$) l'ensemble des variables libres du terme M (resp. de l'analyseur de cas θ). En particulier :

$$\begin{array}{rcl} FV((c_i \mapsto M_i)_{i=1}^n) & = & FV(M_1) \cup \cdots \cup FV(M_n) \\ FV(\{\!\{\theta\}\!\}\,.\,M) & = & FV(\theta) \cup FV(M)\,. \end{array}$$

Comme dans le λ-calcul pur, les termes (ainsi que les analyseurs de cas) sont considérés à α-conversion près. On notera que les questions de renommage ne concernent que les variables, les constructeurs n'étant pas affectés.

L'opération de substitution externe du λ-calcul pur, notée $M\{x := N\}$, est étendue au λ-calcul avec constructeurs de la manière attendue. (Cette opération est aussi définie pour les analyseur de cas.) En particulier, on a :

$$\begin{aligned}(c_i \mapsto M_i)_{i=1}^n\{x := N\} &\equiv (c_i \mapsto M_i\{x := N\})_{i=1}^n \\ (\{\!|\theta|\!\}.\, M)\{x := N\} &\equiv \{\!|\theta\{x := N\}|\!\}.\, M\{x := N\}\,.\end{aligned}$$

3.2.2 Règles de réduction

Le λ-calcul avec constructeurs dispose de 9 règles de réduction primitives qui sont données dans la Fig. 3.1.

Béta-réduction				
AppLam	$(\lambda x\,.\,M)N$	$\to$	$M\{x := N\}$	
AppDai	$\maltese N$	$\to$	$\maltese$	
Êta-réduction				
LamApp	$\lambda x\,.\,Mx$	$\to$	M	$(x \notin FV(M))$
LamDai	$\lambda x\,.\,\maltese$	$\to$	$\maltese$	
Propagation du *case*				
CaseCons	$\{\!|\theta|\!\}.\, c$	$\to$	M	$((c \mapsto M) \in \theta)$
CaseDai	$\{\!|\theta|\!\}.\, \maltese$	$\to$	$\maltese$	
CaseApp	$\{\!|\theta|\!\}.\, (MN)$	$\to$	$(\{\!|\theta|\!\}.\, M)N$	
CaseLam	$\{\!|\theta|\!\}.\, \lambda x\,.\,M$	$\to$	$\lambda x\,.\,\{\!|\theta|\!\}.\, M$	$(x \notin FV(\theta))$
Conversion de *case*				
CaseCase	$\{\!|\theta|\!\}.\, (\{\!|\phi|\!\}.\, M)$	$\to$	$\{\!|\theta \circ \phi|\!\}.\, M$	

Fig. 3.1 – Règles de réduction du λ-calcul avec constructeurs

On y retrouve les règles habituelles du $\lambda(\eta)$-calcul : la règle β (maintenant appelée AppLam) et la règle η (LamApp). Le filtrage est propagé à travers les termes à l'aide des règles CaseApp et CaseLam, la substitution de constructeur étant effectuée par la règle CaseCons. La règle CaseCase permet quant à elle de fusionner des filtrages adjacents.

Enfin, le démon $\maltese$ vient avec trois règles de réduction AppDai (la règle β du démon) LamDai (la règle η du démon) et CaseDai qui décrivent comment le démon détruit son contexte d'évaluation. (Cette notion sera définie formellement en 3.4.2.) Intuitivement, le démon est une forme d'exception non rattrapable ; on peut le voir comme le dual de la non-terminaison. (Dans un domaine de Scott, on interpréterait naturellement le démon par le plus grand élément.)

Dans ce qui suit, on ne s'intéressera pas seulement au système complet — défini par l'ensemble des 9 règles de réduction — mais plus généralement aux

$2^9 = 512$ sous-systèmes formés par tous les sous-ensembles de ces 9 règles. On désigne par $\lambda\mathcal{B}_\mathcal{C}$ le calcul engendré par toutes les règles de la Fig. 3.1, et on note $\mathcal{B}_\mathcal{C}$ le sous-calcul engendré par toutes les règles sauf AppLam (i.e. β).

3.2.3 Exemples

Le filtrage à la ML Le λ-calcul avec constructeurs permet avant tout d'implémenter l'analyse par cas sur les valeurs des types énumérés. Dans $\lambda\mathcal{B}_\mathcal{C}$, les booléens sont représentés par deux constructeurs true et false, et on pose :

$$\texttt{if } N \texttt{ then } M_1 \texttt{ else } M_2 \quad \equiv \quad \{\!|\mathsf{true} \mapsto M_1;\ \mathsf{false} \mapsto M_2|\!\}.\, N$$

D'après la règle CaseCons, on a :

$$\begin{array}{lcl} \texttt{if true then } M_1 \texttt{ else } M_2 & \equiv & \{\!|\mathsf{true} \mapsto M_1;\ \mathsf{false} \mapsto M_2|\!\}.\,\mathsf{true} \ \to\ M_1 \\ \texttt{if false then } M_1 \texttt{ else } M_2 & \equiv & \{\!|\mathsf{true} \mapsto M_1;\ \mathsf{false} \mapsto M_2|\!\}.\,\mathsf{false} \ \to\ M_2 \end{array}$$

Cependant, la règle CaseApp permet d'aller bien au-delà de la simple analyse par cas, et d'implémenter le filtrage sur des motifs non constants. Pour cela, considérons la fonction prédécesseur (sur les entiers unaires) qui envoie $\mathsf{0}$ sur $\mathsf{0}$ et $\mathsf{s}\,n$ sur n. Dans $\lambda\mathcal{B}_\mathcal{C}$, cette fonction est implémentée par le terme

$$\text{pred} \quad \equiv \quad \lambda n\,.\,\{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\, n$$

(où $\mathsf{0}$ et s sont deux constructeurs distincts). Le calcul de $\text{pred}\,\mathsf{0}$ est immédiat d'après les règles AppLam ($=\beta$) et CaseCons :

$$\text{pred } \mathsf{0} \ \to\ \{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,\mathsf{0} \ \to\ \mathsf{0}\,.$$

Le calcul $\text{pred}\,(\mathsf{s}\,N)$ est plus intéressant :

$$\begin{array}{l} \text{pred } (\mathsf{s}\ N) \to \{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,(\mathsf{s}\ N) \\ \qquad\qquad \to (\{\!|\mathsf{0} \mapsto \mathsf{0};\ \mathsf{s} \mapsto \lambda z\,.\,z|\!\}.\,\mathsf{s})\ N \ \to\ (\lambda z\,.\,z)\ N \ \to\ N \end{array}$$

(où N désigne un terme arbitraire). On voit ici comment la construction *case* capture l'occurrence de tête du constructeur s à l'aide de la règle CaseApp.

Plus généralement, le filtrage à la ML (sur des motifs disjoints) est traduit dans le λ-calcul avec constructeurs de la manière suivante :

$$\begin{array}{l} \texttt{match } N \texttt{ with} \\ \mid\ c_1(x_1,\ldots,x_{n_1}) \ \mapsto\ M_1 \\ \qquad\vdots \\ \mid\ c_k(x_1,\ldots,x_{n_k}) \ \mapsto\ M_k \end{array} \qquad \text{devient} \qquad \begin{array}{l} \{\!|c_1 \mapsto \lambda x_1 \cdots x_{n_1}\,.\,M_1\,; \\ \qquad\vdots \\ c_k \mapsto \lambda x_1 \cdots x_{n_k}\,.\,M_k\,; \\ |\!\}\cdot N \end{array}$$

Filtrage et récursion La récursion est implémentée dans $\lambda\mathcal{B}_\mathcal{C}$ de la même manière que dans le λ-calcul pur, à l'aide d'un combinateur de point fixe, par exemple le combinateur de Turing : $\Theta \equiv (\lambda yf\,.\,f(yyf))\ (\lambda yf\,.\,f(yyf))$. En associant récursion et analyse par cas, on peut définir des fonctions récursives sur les structures de données algébriques, comme par exemple la fonction de concaténation de listes :

$$\text{append} \quad \equiv \quad \Theta\ \big(\lambda f l_1 l_2\,.\,\{\!|\mathsf{nil} \mapsto l_2;\ \mathsf{cons} \mapsto \lambda x l'\,.\,\mathsf{cons}\ x\ (f\ l'\ l_2)|\!\}.\, l_1\big)$$

(en supposant les listes représentées de la manière habituelle, à partir de deux constructeurs nil et cons). Dans ce qui suit, on utilisera les opérateurs arithmétiques suivants :

$$\begin{aligned} \text{plus} &\equiv \Theta\left(\lambda fnm\,.\,\{\!|\mathsf{0} \mapsto m;\ \mathsf{s} \mapsto \lambda p\,.\,\mathsf{s}\ (f\ p\ m)|\!\}\,.\,n\right) \\ \text{minus} &\equiv \Theta\left(\lambda fnp\,.\,\{\!|\mathsf{0} \mapsto n;\ \mathsf{s} \mapsto \lambda q\,.\,f\ (\text{pred}\ n)\ q)|\!\}\,.\,p\right) \end{aligned}$$

Constructeurs variadiques Comme les constructeurs n'ont pas d'arité fixée dans $\lambda\mathcal{B}_\mathcal{C}$, il est tentant d'utiliser certains d'entre eux comme des constructeurs variadiques, c'est-à-dire comme des constructeurs susceptibles d'être appliqués à un nombre arbitraire d'arguments.

Par exemple, il est possible de représenter les vecteurs (les n-uplets variadiques) avec un unique constructeur vec en posant :

$$(v_1, \ldots, v_n) \quad \equiv \quad \mathsf{vec}\ x_1\ \cdots\ x_n\,.$$

Avec cette représentation naïve, on peut écrire la fonction de concaténation

$$\text{vappend} \quad \equiv \quad \lambda v_1 v_2\,.\,\{\!|\mathsf{vec} \mapsto v_1|\!\}\,.\,v_2\,,$$

mais pas la fonction qui accède à un élément d'un vecteur à partir de son rang. En effet, pour accéder au i-ème élément d'un vecteur $v = \mathsf{vec}\ x_1 \cdots x_n$, il faut non seulement sauter les $i-1$ premiers arguments, mais aussi faire disparaître les $n-i-1$ arguments restants. Malheureusement, cette dernière opération ne peut pas être implémentée dans $\lambda\mathcal{B}_\mathcal{C}$ car le langage ne fournit aucun moyen de compter le nombre d'arguments auquel un constructeur est appliqué.

Pour résoudre ce problème, la solution la plus simple consiste à fournir cette longueur explicitement comme premier argument du constructeur vec. La représentation des vecteurs devient alors :

$$(v_1, \ldots, v_n) \quad \equiv \quad \mathsf{vec}\ \bar{n}\ x_1\ \cdots\ x_n\,.$$

(où $\bar{n}$ désigne la représentation unaire de l'entier n à l'aide des constructeurs $\mathsf{0}$ et s). Avec cette nouvelle représentation, on peut à présent écrire des fonctions qui calculent la longueur d'un vecteur ou accèdent à un élément de rang donné :

$$\begin{aligned} \text{skip} &\equiv \Theta\left(\lambda fnx\,.\,\{\!|\mathsf{0} \mapsto x;\ \mathsf{s} \mapsto \lambda pz\,.\,fpx|\!\}\,.\,n\right) \\ \text{length} &\equiv \lambda v\,.\,\{\!|\mathsf{vec} \mapsto \lambda n\,.\,\text{skip}\ n\ n|\!\}\,.\,v \\ \text{access} &\equiv \lambda vi\,.\,\{\!|\mathsf{vec} \mapsto \lambda n\,.\,\text{skip}\ i\ (\lambda x\,.\,\text{skip}\ (\text{minus}\ n\ (\mathsf{s}\ i))\ x)|\!\}\,.\,v \end{aligned}$$

Bien entendu, il est toujours possible de définir une fonction de concaténation, bien que son code soit un peu plus compliqué :

$$\text{vappend} \quad \equiv \quad \lambda v_1 v_2\,.\,\{\!|\mathsf{vec} \mapsto \lambda n_1\,.\,\{\!|\mathsf{vec} \mapsto \lambda n_2\,.\,\mathsf{vec}\ (\text{plus}\ n_1\ n_2)|\!\}\,.\,v_2|\!\}\,.\,v_1$$

3.3 Un théorème de Church-Rosser modulaire

On s'intéresse maintenant à la propriété de Church-Rosser pour le λ-calcul avec constructeurs ($\lambda\mathcal{B}_\mathcal{C}$) et ses sous-systèmes. L'originalité du théorème que

nous allons présenter (Théorème 3 p. 3) réside avant tout dans la technique de preuve utilisée, qui est à ma connaissance originale dans un tel contexte. Plutôt que d'utiliser la technique des réductions parallèles (dans le style de Tait et Martin-Löf [67]) ou la méthode d'interprétation (voir par exemple [93]), nous allons morceler le problème en analysant les propriétés de confluence et de commutation pour les 512 sous-systèmes formés à partir de tous les sous-ensembles des 9 règles de la Fig. 3.1. (Dans ce qui suit, on utilisera l'expression « sous-système » uniquement dans ce sens.) Nous verrons comment les propriétés de commutation d'un faible nombre de sous-systèmes judicieusement choisis permettent non seulement de déduire (quasi-mécaniquement) la propriété de Church-Rosser pour le système complet, mais plus généralement de répondre à la question « le sous-système s est-il confluent ? » pour chacun des 512 sous-systèmes s du λ-calcul avec constructeurs. Au final, nous aurons une preuve de confluence pour 248 d'entre-eux, et un contre-exemple pour chacun des 264 sous-systèmes restants, avec un critère simple pour décider si un sous-système est confluent ou non.

Dans la suite, on note

- $s_1 \mathbin{/\!/} s_2$ lorsque s_1 commute avec s_2
- $s_1 \mathbin{/\!/_w} s_2$ lorsque s_1 commute faiblement avec s_2

où s_1 et s_2 désignent des sous-systèmes arbitraires, formés chacun à partir d'un ensemble quelconque des 9 règles de la Fig. 3.1. (On se reportera à [7, 99, 5] pour les définitions correspondantes.) En règle générale, la propriété de commutation faible $s_1 \mathbin{/\!/_w} s_2$ n'implique pas la propriété de commutation $s_1 \mathbin{/\!/} s_2$, sauf dans le cas particulier où le système $s_1 \cup s_2$ est fortement normalisable, où ces deux propriétés sont équivalentes (d'après le lemme de Newmann [7, 99]).

3.3.1 Normalisation forte de $\mathcal{B}_\mathcal{C}$

Dans le λ-calcul avec constructeurs, la règle AppLam (β) est la seule règle responsable de la non terminaison du calcul. Toutes les autres règles prises ensemble définissent un système de réécriture fortement normalisable :

Proposition 21 (Normalisation forte de $\mathcal{B}_\mathcal{C}$) — *Tous les termes sont fortement normalisables au sens de $\mathcal{B}_\mathcal{C}$ (i.e. $\lambda\mathcal{B}_\mathcal{C} \setminus \{\text{AppLam}\}$).*

Preuve. Il suffit de considérer la mesure de complexité (sur les termes et analyseurs de cas) définie par

$$\begin{array}{rclcrcl} h(x) & = & 1 & & h(MN) & = & h(M) + h(N) \\ h(c) & = & 1 & & h(\lambda x\,.\,M) & = & h(M) + 1 \\ h(\maltese) & = & 1 & & h(\{\!|\theta|\!\}\,.\,M) & \underset{n}{=} & (h(\theta) + 2)h(M) \\ & & & & h((c_i \mapsto M_i)_{i=1}^n) & = & \displaystyle\sum_{i=1} h(M_i) \end{array}$$

et de montrer que cette mesure décroît à chaque étape de contraction suivant l'une des règles AppDai, LamApp, LamDai, CaseApp, CaseLam, CaseCons, CaseDai ou CaseCase. □

De ceci il résulte immédiatement que :

Corollaire 1 (Sous-systèmes fortement normalisables) — *Un sous-système s est fortement normalisable si et seulement si* AppLam $\notin s$.

Une conséquence importante de ce corollaire est la suivante :

Corollaire 2 — *Si deux sous-systèmes s_1 et s_2 sont tels que* AppLam $\notin (s_1 \cup s_2)$, *alors s_1 // s_2 si et seulement si $s_1 //_w s_2$.*

3.3.2 Paires critiques et conditions de fermeture

Chacune des 9 règles de réduction primitives de $\lambda\mathcal{B}_\mathcal{C}$ décrit l'interaction entre deux constructions syntaxiques du langage, reflétées dans le nom de la règle : AppLam pour « APPlication sur un LAMbda », etc. Ces règles de réduction induisent 13 paires critiques, qui sont données dans les Fig. 3.2 et 3.3.

Les paires critiques surviennent pour chaque couple de règles de la forme GaBu/BuZo [94], dans une situation où un GaBu-radical et un BuZo-radical se recouvrent sur un agrégat syntaxique de la forme

$$\text{GaBu-radical} \left\{ \begin{array}{c} \text{Ga} \\ | \\ \text{Bu} \\ | \\ \text{Zo} \end{array} \right\} \text{BuZo-radical}$$

Un rapide examen des Fig. 3.2 et 3.3 montre que pour refermer une paire critique induite par un couple de règles GaBu/BuZo, il est nécessaire d'utiliser la règle GaZo quand cette règle existe. Ce qui survient pour les 6 paires critiques (2), (4), (5), (6), (7) et (8) de la Fig. 3.2. Dans les autres cas, la paire critique est refermée en utilisant seulement les règles GaBu et BuZo.

Cette remarque suggère très naturellement la définition suivante :

Définition 1 (Conditions de fermeture) — On dit qu'un sous-système s satisfait les *conditions de fermeture* et on note $s \models$ CC si les 6 conditions suivantes sont satisfaites :

(CC1)	AppLam	$\in s$	$\wedge$	LamDai	$\in s$	$\Rightarrow$	AppDai	$\in s$
(CC2)	LamApp	$\in s$	$\wedge$	AppDai	$\in s$	$\Rightarrow$	LamDai	$\in s$
(CC3)	CaseApp	$\in s$	$\wedge$	AppLam	$\in s$	$\Rightarrow$	CaseLam	$\in s$
(CC4)	CaseApp	$\in s$	$\wedge$	AppDai	$\in s$	$\Rightarrow$	CaseDai	$\in s$
(CC5)	CaseLam	$\in s$	$\wedge$	LamApp	$\in s$	$\Rightarrow$	CaseApp	$\in s$
(CC6)	CaseLam	$\in s$	$\wedge$	LamDai	$\in s$	$\Rightarrow$	CaseDai	$\in s$

Intuitivement, un sous-système qui satisfait aux 6 conditions de fermeture définit un système de réduction dans lequel toutes les paires critiques peuvent être refermées ; c'est donc un bon candidat pour la propriété de Church-Rosser.

Il s'agit à présent de transformer cette intuition en un résultat formel :

Théorème 3 (Church-Rosser) — *Pour chacun des* 512 *sous-systèmes s, les assertions suivantes sont équivalentes :*

1. *s satisfait les conditions de fermeture (CC1)–(CC6) ;*
2. *s est faiblement confluent ;*
3. *s est confluent.*

Comme le système complet (i.e. $\lambda\mathcal{B}_\mathcal{C}$) satisfait évidemment toutes les conditions de fermeture, on aura immédiatement :

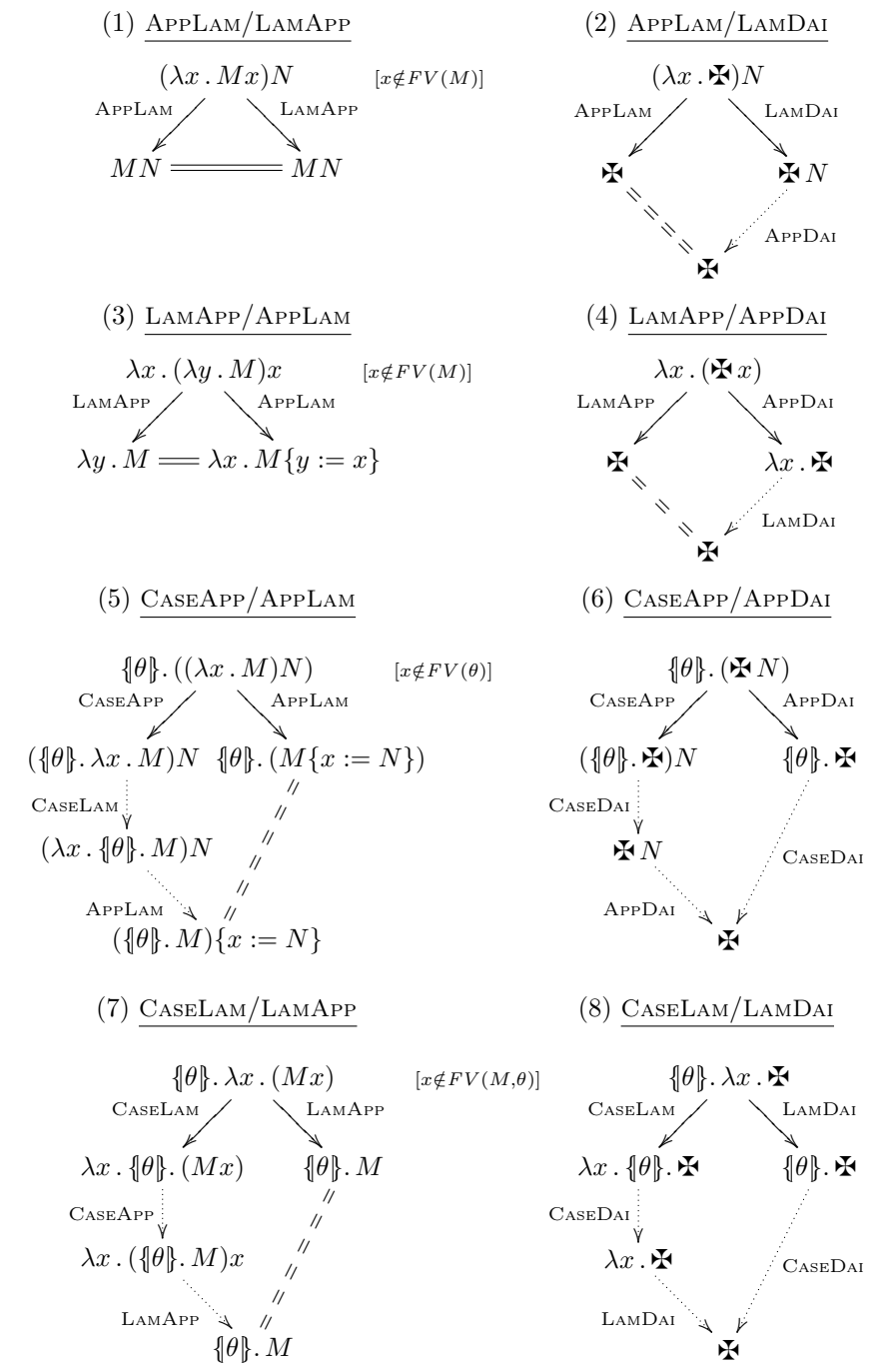


FIG. 3.2 – Paires critiques 1–8 (/13)

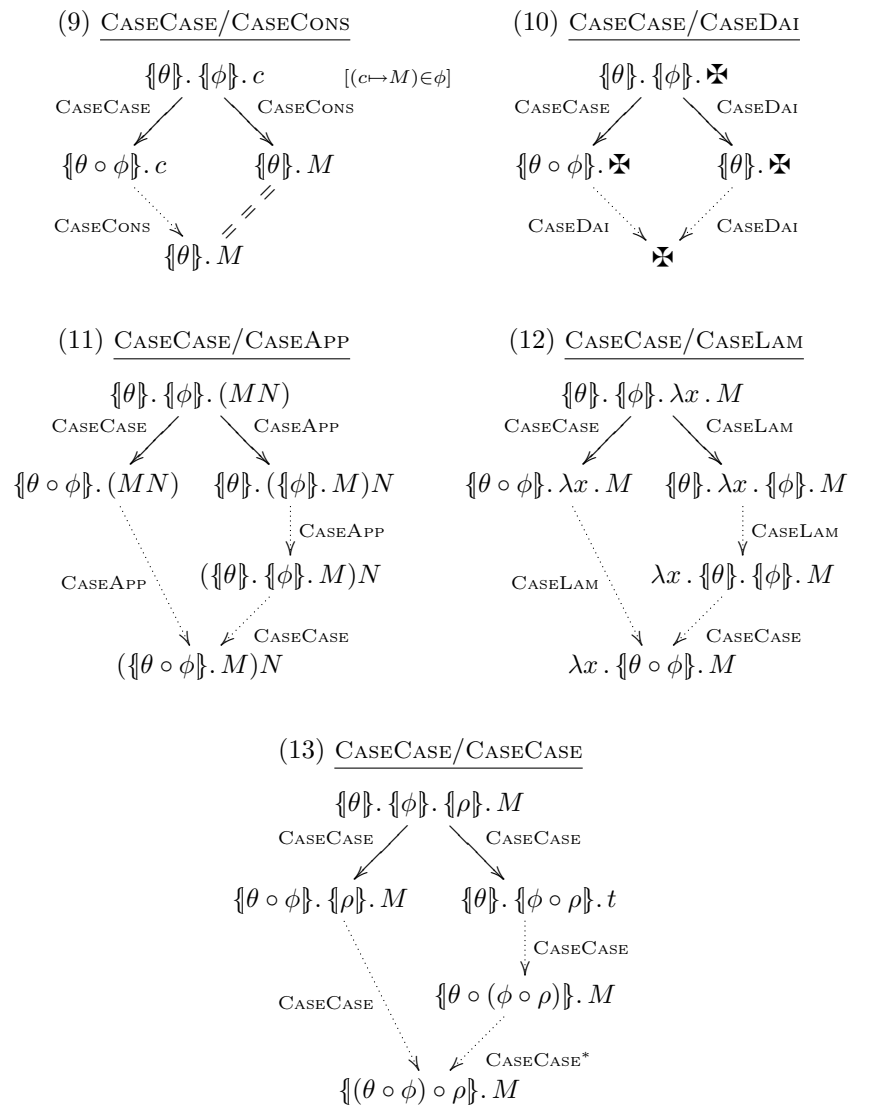


FIG. 3.3 – Paires critiques 9–13 (/13)

Corollaire 3 (Church-Rosser) — *$\lambda\mathcal{B}_\mathcal{C}$ est confluent.*

La preuve du théorème 3 repose sur une analyse systématique de la propriété de commutation pour chaque paire de sous-systèmes $\{s_1, s_2\}$. Pour cela, nous devons d'abord généraliser la notion de condition de fermeture à chaque paire $\{s_1; s_2\}$ de sous-systèmes. Ce qui nous conduit à la définition suivante :

Définition 2 (Conditions de fermeture binaire) — On dit qu'une paire $\{s_1; s_2\}$ de sous-systèmes satisfait les *conditions de fermeture binaire* et on note $\{s_1; s_2\} \models \text{BCC}$ si elle satisfait chacune des 9 conditions suivantes

(BCC1)	AppLam	$\in s_1$	$\wedge$	LamDai	$\in s_2$	$\Rightarrow$	AppDai	$\in s_1$
(BCC2)	LamApp	$\in s_1$	$\wedge$	AppDai	$\in s_2$	$\Rightarrow$	LamDai	$\in s_1$
(BCC3)	CaseApp	$\in s_1$	$\wedge$	AppLam	$\in s_2$	$\Rightarrow$	CaseLam	$\in s_2$
(BCC4)	CaseApp	$\in s_1$	$\wedge$	AppDai	$\in s_2$	$\Rightarrow$	CaseDai	$\in (s_1 \cap s_2)$
(BCC5)	CaseLam	$\in s_1$	$\wedge$	LamApp	$\in s_2$	$\Rightarrow$	CaseApp	$\in s_2$
(BCC6)	CaseLam	$\in s_1$	$\wedge$	LamDai	$\in s_2$	$\Rightarrow$	CaseDai	$\in (s_1 \cap s_2)$
(BCC7)	CaseCase	$\in s_1$	$\wedge$	CaseDai	$\in s_2$	$\Rightarrow$	CaseDai	$\in s_1$
(BCC8)	CaseCase	$\in s_1$	$\wedge$	CaseApp	$\in s_2$	$\Rightarrow$	CaseApp	$\in s_1$
(BCC9)	CaseCase	$\in s_1$	$\wedge$	CaseLam	$\in s_2$	$\Rightarrow$	CaseLam	$\in s_1$

de même que chacune des 9 conditions symétriques (en échangeant s_1 et s_2).

À l'instar des conditions de fermeture simple (CC1)–(CC6), les 9 conditions de fermeture binaire (BCC1)–(BCC9) sont issues d'une analyse des paires critiques. Par exemple, (BCC1) provient de l'observation suivant laquelle la paire critique (2) de la Fig. 3.2 ne peut être formée que si AppLam $\in s_1$ et LamDai $\in s_2$, et que cette paire critique ne peut être refermée que si AppDai $\in s_1$.

Remarquons également que dans le cas où $s_1 = s_2$, les conditions de fermeture binaire (BCC1)–(BCC6) se ramènent aux conditions de fermeture simple (CC1)–(CC6), tandis que les conditions (BCC7)–(BCC9) deviennent des tautologies, d'où il ressort que :

Lemme 4 — *Pour chaque sous-système s :* $s \models \text{CC}$ *ssi* $\{s, s\} \models \text{BCC}$.

On démontre d'abord que

Proposition 22 — *Pour chaque paire $\{s_1; s_2\}$ de sous-systèmes, les assertions suivantes sont équivalentes :*

1. $\{s_1, s_2\} \models \text{BCC}$ *(conditions de fermeture binaire)*
2. $s_1 \,/\!/_w\, s_2$ *(commutation faible).*

Preuve. Voir [5].

3.3.3 La technique de preuve « diviser pour régner »

Considérons à présent la matrice 512×512 (symétrique) formée par les 131 328 paires (non ordonnées) de sous-systèmes de $\lambda\mathcal{B}_\mathcal{C}$.[6] D'après la Prop. 22 et le corollaire 1 on a

$$\begin{array}{rcl} s_1 \,/\!/_w\, s_2 & \Leftrightarrow & (s_1, s_2) \models \text{BCC} \\ (s_1 \cup s_2) \text{ fortement normalisable} & \Leftrightarrow & \text{AppLam} \notin (s_1 \cup s_2) \end{array}$$

[6] On compte $\{s_1; s_2\}$ et $\{s_2; s_1\}$ comme une seule et même paire.

pour chaque paire $\{s_1; s_2\}$. Il est donc clair que les relations « $s_1 \,/\!/_w\, s_2$ » et « $s_1 \cup s_2$ est fortement normalisable » sont toutes les deux décidables.

À l'aide d'un petit programme, on peut dénombrer que parmi ces 131 328 paires de sous-systèmes $\{s_1; s_2\}$, il y en a très exactement 13 396 pour lesquelles on a $s_1 \,/\!/_w\, s_2$ (d'après la Prop. 22), dont 5 612 pour lesquelles en outre l'union des deux systèmes est fortement normalisable. (D'après le corollaire 2, on a donc $s_1 \,/\!/\, s_2$ pour les 5 612 paires de sous-systèmes en question.)

La situation est résumée dans le tableau ci-dessous :

	Paires $\{s_1; s_2\}$	$s_1 = s_2$
Commutation faible	13 396	248 WCR
dont SN	5 612	160 CR+SN
Pas de commutation faible	117 932	264 ¬WCR
Total	131 328	512

Il reste maintenant à établir que la propriété de commutation « $s_1 \,/\!/\, s_2$ » vaut également pour chacune des $13\,396 - 5\,612 = 7\,784$ paires de sous-systèmes qui commutent faiblement, mais dont l'union n'est pas fortement normalisable.

Pour réduire le nombre de résultats de commutation à démontrer (7 784), on utilise une technique de type « diviser pour régner » qui consiste à appliquer mécaniquement le résultat suivant :

$$(*) \qquad \text{Si} \;\; s_1 \,/\!/\, s_2 \;\; \text{et} \;\; s_1 \,/\!/\, s_3, \;\; \text{alors} \;\; s_1 \,/\!/\, (s_2 \cup s_3)$$

afin de propager la connaissance des paires de sous-systèmes commutant à travers la matrice (en utilisant l'algorithme décrit dans la Fig. 3.4). En pratique, cette méthode permet de réduire la preuve des 7 784 lemmes de commutation à seulement 12 lemmes, qui sont donnés dans la Table 3.1, car :

Lemme 5 — *Si les 12 paires de sous-systèmes de la Table 3.1 commutent, alors les* 13 396 *paires de sous-systèmes faiblement commutant commutent.*

Preuve. Ce résultat s'établit mécaniquement à partir de l'algorithme décrit dans la Fig. 3.4 p. 69.

(1) AppLam // AppLam	(8) AppLam + AppDai // LamDai
(2) AppLam // AppDai	(9) AppLam + AppDai // LamApp + LamDai
(3) AppLam // LamApp	(10) AppLam + CaseLam // CaseApp
(4) AppLam // CaseCons	(11) AppLam + CaseLam // LamApp + CaseApp
(5) AppLam // CaseDai	(12) AppLam + AppDai + CaseDai + CaseLam //
(6) AppLam // CaseLam	LamApp + LamDai + CaseDai + CaseApp
(7) AppLam // CaseCase	

Tab. 3.1 – Les 12 lemmes de commutation initiaux

La preuve du théorème 3 se termine alors en démontrant à la main chacun des 12 lemmes de la Table 3.1. (Le détail de la démonstration de ces lemmes est donné dans les appendices de [5].)

Dans le code ci-dessous, $C[s_1, s_2]$ désigne une matrice symétrique de booléens indicée par toutes les paires de sous-systèmes $\{s_1; s_2\}$. On suppose que chaque affectation $C[s_1, s_2] := b$ implique l'affectation symétrique $C[s_2, s_1] := b$.

Invariant global : $\quad C[s_1, s_2] \quad \Rightarrow \quad s_1 \mathbin{//} s_2$

[Initialise C avec les paires SN + commutant]
for each $\{s_1; s_2\}$ **do**
  $C[s_1, s_2] \;:=\; check\_bcc(s_1, s_2) \;\wedge\; \textsc{AppLam} \notin (s_1 \cup s_2)$
done;

[Positionne les 12 lemmes de commutation initiaux]
for each $\{s_1; s_2\} \in$ Table 3.1 **do**
  $C[s_1, s_2] \;:=$ **true**
done;

[Fermeture de la matrice par $(*)$]
while
  there are $s_1,\ s_2,\ s_3$ such that :
  $C[s_1, s_2] \wedge C[s_1, s_3] \wedge \neg C[s_1, (s_2 \cup s_3)]$
do $C[s_1, (s_2 \cup s_3)] \;:=$ **true** **done**;

[Vérifie que les paires BCC commutent]
for each $\{s_1; s_2\}$ **do**
  assert$\big(C[s_1, s_2] \;\Leftrightarrow\; check\_bcc(s_1, s_2)\big)$
done
[Si aucune erreur n'est rencontrée, alors le lemme 5 est vrai]

FIG. 3.4 – Construction de la matrice de commutation

3.4 Le théorème de séparation

Nous allons maintenant présenter la preuve du théorème de séparation faible pour le λ-calcul avec constructeurs, qui exprime que deux formes normales distinctes peuvent être séparées (faiblement) par un contexte bien choisi. Comme le calcul ne fournit aucun mécanisme pour éliminer les termes de la forme $\{\!|\theta|\!\}.\, c$ où $c \notin \mathrm{dom}(\theta)$ (erreur de filtrage), nous devrons tout d'abord exclure ces termes du processus de séparation — d'où la notion de forme normale complètement définie que nous introduirons à la fin de la section 3.4.3.

Comme pour le théorème de Böhm [16] dans le λ-calcul pur, la principale difficulté technique est de séparer des formes normales d'arités distinctes (i.e. commençant avec un nombre différent de λ-abstractions). Pour résoudre ce problème, on procède en deux étapes. D'abord, on définit une notion de désaccord (à une certaine profondeur), et on démontre que deux formes normales distinctes peuvent être η-expansées de telle sorte à faire apparaître un désaccord à une certaine profondeur. L'avantage de cette notion de désaccord est de préserver une certaine cohérence entre les arités des deux termes le long du chemin qui conduit au désaccord. Une fois cette propriété établie, on démontre que deux termes en désaccord peuvent être séparés à l'aide d'un contexte approprié.

3.4.1 Formes quasi-normales

Définition 3 (Terme de tête) — On appelle *terme de tête* (notation : H, H_1, H', etc.) tout terme qui est de l'une des quatre formes suivantes :

$$\textbf{Terme de tête} \qquad \begin{array}{rcl} H & ::= & x \quad | \quad c \\ & | & \{\!|\theta|\!\}.\,x \quad | \quad \{\!|\theta|\!\}.\,c \qquad (c \notin \mathrm{dom}(\theta)) \end{array}$$

Quand un terme de tête est de l'une des trois premières formes (variable, constructeur ou analyseur de cas sur une variable), on dit que H est *défini*. Dans le cas où H est de la dernière forme (analyseur de cas sur un constructeur en dehors du domaine), on dit que H est *indéfini*.

On dit qu'un terme, un terme de tête ou un analyseur de cas est en *forme quasi-normale* s'il est en forme normale par rapport à toutes les règles de réduction sauf LamApp (η). Comme l'ensemble de toutes les règles de réduction privé de la règle LamApp satisfait les conditions de fermeture de la Déf. 1, le sous-système qui en résulte est confluent et la forme quasi-normale d'un terme, d'un terme de tête ou d'un analyseur de cas, lorsque elle existe, est unique.

On a alors la caractérisation syntaxique suivante :

Proposition 23 — *Les termes en forme quasi-normale (N), les termes de tête en forme quasi-normale (H) et les analyseurs en forme quasi-normale (θ) sont caractérisés par la BNF suivante :*

$$\begin{array}{rcll} N & ::= & \Lightthorpe \quad | \quad \lambda x_1 \cdots x_n\,.\,H N_1 \cdots N_k & (n,k \geq 0) \\ H & ::= & x \quad | \quad c \quad | \quad \{\!|\theta|\!\}.\,x \quad | \quad \{\!|\theta|\!\}.\,c & (c \notin \mathrm{dom}(\theta)) \\ \theta & ::= & (c_1 \mapsto N_1; \ldots; c_p \mapsto N_p) & (p \geq 0) \end{array}$$

Définition 4 (Terme indéfini) — On dit qu'un terme M est *indéfini* s'il se réduit en zéro, une ou plusieurs étapes sur un terme de la forme

$$\lambda x_1 \cdots x_n\,.\,H N_1 \cdots N_k \qquad (n,k \geq 0)$$

où H est un terme de tête indéfini (i.e. $H \equiv \{\!|\theta|\!\}.\,c$ avec $c \notin \mathrm{dom}(\theta)$).

(Dans cette définition on considère la réduction au sens des 9 règles de réduction prises ensemble, mais on pourrait tout aussi bien retirer la règle AppLam sans changer la notion sous-jacente.)

3.4.2 Contextes d'évaluation

La notion de contexte à un trou (notation : C, C_1, C', etc.) est définie de la même manière que dans le λ-calcul pur [9]. Le terme obtenu en insérant un terme M dans le trou d'un contexte C à un trou est noté $C[M]$. Dans ce qui suit, on s'intéressera plus particulièrement à des contextes d'une forme particulière : les contextes d'évaluation :

$$\textbf{Contextes d'évaluation} \qquad E \quad ::= \quad []N_1 \cdots N_n \quad | \quad \{\!|\theta|\!\}.\,[]N_1 \cdots N_n$$

(La seconde forme se lit : $(\{\!|\theta|\!\}.\,[])N_1 \cdots N_n$.)

On notera que la composition $E'[E]$ de deux contextes d'évaluation E et E' n'est pas toujours un contexte d'évaluation. En revanche, elle se réduit toujours sur un contexte d'évaluation en appliquant zéro, une ou plusieurs fois la règle CaseApp, suivie éventuellement d'une application de la règle CaseCase :

$$\begin{array}{rl} & \left[\begin{array}{r} []N_1\cdots N_k \\ \{\!|\theta|\!\}.\,[]N_1\cdots N_k \end{array}\right] N'_1\cdots N'_{k'} \equiv \left[\begin{array}{r} []N_1\cdots N_kN'_1\cdots N'_{k'} \\ \{\!|\theta|\!\}.\,[]N_1\cdots N_kN'_1\cdots N'_{k'} \end{array}\right] \\ \{\!|\theta'|\!\}. & \left[\begin{array}{r} []N_1\cdots N_k \\ \{\!|\theta|\!\}.\,[]N_1\cdots N_k \end{array}\right] N'_1\cdots N'_{k'} \to^* \left[\begin{array}{r} \{\!|\theta'|\!\}.\,[]N_1\cdots N_kN'_1\cdots N'_{k'} \\ \{\!|\theta'\circ\theta|\!\}.\,[]N_1\cdots N_kN'_1\cdots N'_{k'} \end{array}\right] \end{array}$$

Il est utile de remarquer qu'un contexte d'évaluation E peut toujours être vu comme un terme d'une forme particulière qui contient exactement une occurrence d'une variable notée [] (i.e. le trou). Comme cette unique occurrence n'est dans le scope d'aucun lieur figurant dans E, l'opération de remplacement $E[M]$ fonctionne comme une substitution ordinaire $E\{[] := M\}$. (Ce n'est évidemment pas le cas pour la forme générale des contextes à un trou [9].)

3.4.3 Critères de séparation

Le démon ✠ absorbe tous les contextes d'évaluation :

Lemme 6 — *Pour tout contexte d'évaluation E on a* $E[✠] \to^* ✠$.

Symétriquement, les termes de la forme $\{\!|\theta|\!\}.\,c$ avec $c \notin \mathrm{dom}(\theta)$ (erreur de filtrage) bloquent le calcul en position de tête, si bien que les termes indéfinis absorbent également tout les contextes d'évaluation :

Lemme 7 — *Pour tout contexte d'évaluation E et pour tout terme indéfini U, le terme $E[U]$ est indéfini.*

Le démon et les termes indéfinis sont donc les candidats naturels (les « observables ») pour définir la notion de séparation :

Définition 5 (Séparation) — On dit de deux termes M_1 et M_2 qu'ils sont
- *faiblement séparables* s'il existe un contexte à un trou C tel que :
  - $C[M_1] \to^* ✠$ et $C[M_2]$ est indéfini, ou bien
  - $C[M_2] \to^* ✠$ et $C[M_1]$ est indéfini
- *fortement séparables* s'il existe deux contextes C_1 et C_2 à un trou tels que
  - $C_1[M_1] \to^* ✠$ et $C_1[M_2]$ est indéfini, et
  - $C_2[M_2] \to^* ✠$ et $C_2[M_1]$ est indéfini.

Comme deux termes indéfinis ne peuvent pas être séparés, il est nécessaire de les exclure du processus de séparation :

Définition 6 (Forme quasi-normale complètement définie) — On dit qu'une forme quasi-normale est *complètement définie* si elle ne contient aucun sous-terme de tête indéfini, c'est-à-dire de la forme $\{\!|\theta|\!\}.\,c$ avec $c \notin \mathrm{dom}(\theta)$. (Cette notion vaut pour les termes et les analyseurs de cas.)

3.4.4 Désaccord

Il s'agit à présent de définir la notion de *désaccord* entre deux termes M_1 et M_2 (à une certaine profondeur $d \geq 0$), notion à partir de laquelle nous pourrons effectuer la preuve du théorème de séparation.

Définition 7 (Équivalence de squelette) — On dit que deux termes de tête définis H_1 et H_2 ont *le même squelette* et on note $H_1 \approx H_2$ si l'une des conditions suivantes est vérifiée :

1. ou bien $H_1 \equiv H_2 \equiv x$ pour une certaine variable x ;
2. ou bien $H_1 \equiv H_2 \equiv c$ pour un certain constructeur c ;
3. ou bien $H_1 \equiv \{\!|\theta_1|\!\}\,.\,x$ et $H_2 \equiv \{\!|\theta_2|\!\}\,.\,x$ pour une certaine variable x et pour certains analyseurs de cas θ_1 et θ_2 tels que $\mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$.

En considérant la négation de la relation d'équivalence ainsi définie, il est clair que deux termes de tête définis H_1 et H_2 n'ont pas le même squelette si l'une des conditions suivantes est remplie :

- H_1 est une variable et H_2 est un constructeur (ou symétriquement) ;
- H_1 est un analyseur de cas sur une variable et H_2 est un constructeur (ou symétriquement) ;
- H_1 est un analyseur de cas sur une variable et H_2 est une variable (ou symétriquement) ;
- H_1 et H_2 sont des variables distinctes ;
- H_1 et H_2 sont des constructeurs distincts ;
- H_1 et H_2 sont de la forme $H_1 \equiv \{\!|\theta_1|\!\}\,.\,x_1$ et $H_2 \equiv \{\!|\theta_2|\!\}\,.\,x_2$ avec ou bien $x_1 \neq x_2$, ou bien $\mathrm{dom}(\theta_1) \neq \mathrm{dom}(\theta_2)$.

(Il n'est pas nécessaire le cas des termes de tête correspondant à une erreur de filtrage, lesquels sont exclus de notre définition.)

Définition 8 (Désaccord à la profondeur d) — Pour chaque entier $d \in \mathbb{N}$ on définit une relation binaire sur les termes quasi-normaux complètement définis, notée $\mathrm{dis}_d(M_1, M_2)$ (« M_1 et M_2 sont en désaccord à la profondeur d »). Cette relation est définie par induction sur $d \in \mathbb{N}$ de la façon suivante :

- (*Cas de base*) On note $\mathrm{dis}_0(M_1, M_2)$ si ou bien
  - $M_1 \equiv \Lightarrow$ et $M_2 \equiv \lambda x_1 \cdots x_n\,.\,H N_1 \cdots N_k$; ou
  - $M_1 \equiv \lambda x_1 \cdots x_n\,.\,H N_1 \cdots N_k$ et $M_2 \equiv \Lightarrow$; ou
  - $M_1 \equiv \lambda x_1 \cdots x_n\,.\,H_1 N_{1,1} \cdots N_{1,k_1}$ et $M_2 \equiv \lambda x_1 \cdots x_n\,.\,H_2 N_{2,1} \cdots N_{2,k_2}$ avec $n, k_1, k_2 \geq 0$ et $H_1 \not\approx H_2$.
- (*Cas inductif*) Pour tout $d \in \mathbb{N}$, on note $\mathrm{dis}_{d+1}(M_1, M_2)$ si $M_1 \equiv \lambda x_1 \cdots x_n\,.\,H_1 N_{1,1} \cdots N_{1,k_1}$ et $M_2 \equiv \lambda x_1 \cdots x_n\,.\,H_2 N_{2,1} \cdots N_{2,k_2}$ (avec $n, k_1, k_2 \geq 0$) et $H_1 \approx H_2$, et si en outre l'une des deux conditions suivantes est vérifiée :
  - $H_1 \equiv \{\!|\theta_1|\!\}\,.\,y$ et $H_2 \equiv \{\!|\theta_2|\!\}\,.\,y$ avec $\mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$ et $\mathrm{dis}_d(\theta_1(c), \theta_2(c))$ pour un certain $c \in \mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$; ou bien
  - $\mathrm{dis}_d(N_{1,k}, N_{2,k})$ pour au moins un entier $1 \leq k \leq \min(k_1, k_2)$.

L'intérêt de cette notion de « désaccord » est de garantir que tout au long du chemin qui mène au désaccord final (i.e. le désaccord à la profondeur 0), les

deux sous-termes considérés commencent toujours par le même nombre d'abstractions. On démontre alors que deux formes normales distinctes (et complètement définies) peuvent toujours être mises en désaccord, quitte à effectuer certaines η-expansions :

Lemme 8 (Lemme de préparation) — *Si M_1 et M_2 sont deux formes normales (i.e. pour toutes les règles de réduction) complètement définies telles que $M_1 \not\equiv M_2$, alors on peut trouver deux formes quasi-normales complètement définies M'_1 et M'_2 telles que $M'_1 \to_\eta^* M_1$, $M'_2 \to_\eta^* M_2$, et $\mathrm{dis}_d(M'_1, M'_2)$ à une certaine profondeur $d \in \mathbb{N}$.*

3.4.5 Le théorème de séparation

Étant donné un terme M en forme quasi-normale, on appelle la *force d'application* de M le plus grand entier $k \geq 0$ tel que M contient un sous-terme de la forme $HN_1 \cdots N_k$, où H est un terme de tête.

Pour tout $K \geq 0$ et pour tout ensemble fini de variables $X = \{x_1; \ldots; x_n\}$ on note σ_X^K la substitution définie par

$$\sigma_X^K = \{x_1 := \langle \mathtt{x}_1; *_K \rangle; \ldots; x_n := \langle \mathtt{x}_n; *_K \rangle\}$$

où $\mathtt{x}_i$ désigne un entier de Peano unique associé à la variable x_i[7], et où $\langle M; *_K \rangle$ désigne le terme défini par $\langle M; *_K \rangle \equiv \lambda y_1 \cdots y_k z \,.\, z\, M\, y_1 \cdots y_K$.

On commence par démontrer le résultat suivant :

Lemme 9 (Séparation immédiate) — *Soit M un terme quasi-normal complètement défini qui n'est pas le démon, c'est-à-dire, un terme de la forme*

$$M \equiv \lambda x_1 \cdots x_n \,.\, HN_1 \cdots N_k$$

(où H est une tête définie). Étant donné un ensemble fini de variables X tel qu'on ait $FV(M) \subseteq X$ et un entier $K \geq k$, on peut trouver des contextes d'évaluation E_1 et E_2 tels que $E_1\big[M[\sigma_X^K]\big] \to^* \maltese$ et $E_2\big[M[\sigma_X^K]\big]$ est indéfini.

Ce lemme nous permet alors de séparer (faiblement) n'importe quel couple de termes quasi-normaux et complètement définis qui sont en désaccord :

Proposition 24 (Séparation des termes en désaccord) — *Si M_1 et M_2 sont deux termes quasi-normaux complètement définis en désaccord à une profondeur $d \in \mathbb{N}$, alors il existe un contexte d'évaluation clos E tel que ou bien*
- *$E\big[M_1[\sigma_X^K]\big] \to^* \maltese$ et $E\big[M_2[\sigma_X^K]\big]$ est indéfini, ou*
- *$E\big[M_2[\sigma_X^K]\big] \to^* \maltese$ et $E\big[M_1[\sigma_X^K]\big]$ est indéfini ;*

où X est n'importe quel ensemble fini de variables contenant au moins toutes les variables libres de M_1 et M_2, et où K est un entier supérieur ou égal à la force d'application des deux termes M_1 et M_2.

Preuve. Par récurrence sur la profondeur d. (Voir [5].) □

Combinée avec le lemme 8, la proposition ci-dessus nous permet de conclure :

Théorème 4 (Séparation) — *Si M_1 et M_2 sont deux termes normaux complètement définis tels que $M_1 \not\equiv M_2$, alors M_1 et M_2 sont faiblement séparables.*

[7] C'est par ce procédé qu'on séparera deux variables libres distinctes.

3.4.6 Pourquoi un théorème de séparation *faible* ?

Il n'est pas possible de transformer le théorème de séparation faible (Théorème 4) en un théorème de séparation forte. Pour s'en convaincre il suffit de considérer les deux termes suivants :

$$M_1 \equiv \{\!\}.\, x \qquad \text{et} \qquad M_2 \equiv x$$

(où le terme de gauche est constitué d'un analyseur de cas vide au-dessus de la variable x). En effet, il est possible (mais fastidieux) de démontrer que si C est un contexte à un trou tel que $C[M_1] \to^* \maltese$, alors :

- Ou bien le terme M_1 inséré dans le trou n'apparaît jamais en position de tête au cours de la réduction de $C[M_1]$ (en suivant une stratégie de réduction standard[8]), auquel cas on a $C[M] \to^* \maltese$ pour tout terme M.
- Ou bien le terme M_1 inséré dans le trou apparaît en position de tête (au cours de la réduction de $C[M_1]$) après que la variable x a été substituée par le démon, ou par un terme se réduisant vers le démon. (C'est en effet la seule possibilité de traverser l'analyseur de cas vide.)

Dans les deux cas de figure on a alors $C[M_2] \to^* \maltese$, ce qui montre qu'il est impossible de trouver un contexte C pour lequel on a $C[M_1] \to^* \maltese$ tandis que $C[M_2]$ est indéfini. (En revanche, on peut séparer M_1 et M_2 dans le sens inverse en considérant par exemple le contexte $C \equiv (\lambda x\,.\,\{\!|c \mapsto \maltese|\!\}.\,[])c$, où c est un constructeur quelconque.) Les termes M_1 et M_2 sont donc bien faiblement séparables, mais ils ne sont pas fortement séparables.

Plus généralement, on peut montrer que si un contexte C est tel que

$$C[\{\!|c_1 \mapsto c_1; \ldots; c_p \mapsto c_p|\!\}.\, M] \;\to^*\; \maltese$$

alors on a $C[M] \to^* \maltese$ également. Ceci est dû au fait que le terme

$$\lambda x\,.\,\{\!|c_1 \mapsto c_1; \ldots; c_p \mapsto c_p|\!\}.\, x$$

se comporte comme une fonction « sous-identité » qui, appliquée à un terme M, retourne le terme M lui-même (ou un terme convertible) si celui-ci a une forme normale de tête de la forme $\lambda x_1 \cdots x_m\,.\,c_i N_1 \cdots N_k$ (avec $1 \leq i \leq n$), mais retourne un terme indéfini (ou même diverge) sinon.

L'existence de ces fonctions « sous-identité » aura nécessairement des répercussions importantes dans la définition de l'ordre extensionnel sur les arbres de Böhm correspondants, et plus généralement dans la définition d'une sémantique dénotationnelle pour le λ-calcul avec constructeurs (cf section 3.5.2).

3.5 Perspectives

3.5.1 Un système de types pour $\lambda\mathcal{B}_\mathcal{C}$

Une direction de recherche très naturelle concerne la mise au point d'un système de types pour le λ-calcul avec constructeurs, de manière à garantir des

[8] Techniquement, on suit le calcul de $C[M]$ en se plaçant dans une machine à environnement et à double pile. L'environnement sert à conserver la trace de toutes les substitutions effectuées au cours de la réduction, ce qui permet d'effectuer les captures nécessaires à chaque fois que le trou est remplacé par le terme M (quand il arrive en position de tête).

propriétés de sûreté telles que la normalisation forte ou l'absence de situations de blocage dues à un échec de filtrage. Ici, la principale difficulté réside dans la présence de la règle

$$(\textsc{CaseApp}) \qquad \{\!|\theta|\!\}.\,(MN) \quad \to \quad (\{\!|\theta|\!\}.\,M)N$$

dont le membre gauche suggère que le terme M a un type flèche (car ce terme est appliqué) tandis que le membre droit suggère que M a un type somme (car il fait l'objet d'une analyse par cas).

Dans cette direction, Barbara Petit (en thèse au LIP sous ma direction) a mis au point un système de types très riche pour le λ-calcul avec constructeurs, avec du polymorphisme, des types union et intersection et des quantifications existentielles et universelles au second ordre [87].[9] L'originalité de son système de types réside dans le fait qu'il distingue deux formes de types : les types (au sens ordinaire) et les types de *données*, c'est-à-dire les types habités par des structures de données, en considérant ici que les structures de données sont les termes de la forme $cN_1 \cdots N_k$, où c est un constructeur. (Dans ce système, les types de données constituent une sous-catégorie syntaxique des types ordinaires.)

Le système dispose ainsi d'une construction de *type application*, notée DT, dont les habitants sont les structures de données obtenues en appliquant un habitant du type de données D à un habitant du type (ordinaire) T[10]. Le paradoxe soulevé par la règle CaseApp est résolu en introduisant une règle de sous-typage de la forme

$$D \;\leq\; \forall X\,(X \to DX)\,,$$

et qui exprime que toute donnée de type D peut être appliquée à un objet de type X (quelconque) pour former une donnée de type DX. Ainsi, le type somme $A + B$ qui se définit naturellement dans ce cadre par $A + B \equiv \mathbf{c}_A\vec{T}_A \cup \mathbf{c}_B\vec{T}_B$ (où $\mathbf{c}_A$ et $\mathbf{c}_B$ sont les types-singletons associés à des constructeurs distincts) peut être vu comme un sous-type d'un type flèche :

$$A + B \quad\equiv\quad c_A\vec{T}_A \cup c_B\vec{T}_B \quad\leq\quad U \to (c_A\vec{T}_A U \cup c_B\vec{T}_B U)$$

(où U est un type quelconque).

Barbara a démontré [87] que ce système de types garantit la propriété de normalisation forte ainsi que l'absence d'erreur à l'exécution suite à une erreur de filtrage (i.e. le typage rejette les termes indéfinis au sens de la Déf. 4). Le travail de recherche de Barbara semble indiquer qu'il est très difficile — sinon impossible — d'adapter les définitions usuelles des différentes formes de candidats de réductibilité [41, 102] au cadre du λ-calcul avec constructeurs. De fait, il s'agit d'une preuve de normalisation forte indirecte, puisque le calcul est d'abord traduit dans un calcul auxiliaire avec une forme de filtrage plus conventionnelle, la propriété de normalisation forte étant ensuite démontrée pour le calcul auxiliaire avec la technique des candidats de réductibilité de Girard [41] et en utilisant les propriétés de clôture par union démontrées dans [92].

[9] Bien entendu, la richesse d'un tel système de types exclut toute forme de décidabilité du typage. L'intérêt d'une telle approche réside surtout dans les garanties offertes par le typage : normalisation forte, absence d'erreur de filtrage, etc. Il sera toujours possible de restreindre ultérieurement le système de types de manière à restaurer les propriétés de décidabilité.

[10] La notion de type application ne peut évidemment pas être généralisée au cas où D est un type « ordinaire » sous peine de casser la propriété de normalisation. Il suffit pour cela de penser aux habitants du type TT, où T est un type du terme $\lambda x\,.\,xx$ dans le système (par exemple, l'un des deux types $(X \to Y) \cap X$ ou $\forall ZZ \to \forall ZZ$).

3.5.2 Arbres de Böhm et sémantique dénotationnelle

Le théorème de séparation faible suggère la possibilité de définir une notion d'arbre de Böhm [9] pour le λ-calcul avec constructeurs en posant :

$$\begin{array}{lrcl}
\textbf{Arbres} & B & ::= & \bot \mid \Lsh \mid \lambda x_1 \cdots x_n \,.\, H B_1 \cdots B_k \\
\textbf{Têtes} & H & ::= & x \mid c \mid \{\!|\Theta|\!\}\,.\,x \mid \{\!|\Theta|\!\}\,.\,c \\
\textbf{Forêts} & \Theta & ::= & (c_1 \mapsto B_1; \ldots; c_p \mapsto B_p)
\end{array}$$

(Cette BNF se comprenant ici au sens d'une définition co-inductive.)

Au-delà des problèmes (habituels) liés à la nécessité technique d'identifier les arbres de Böhm à η-équivalence près, l'introduction d'une notion d'arbre de Böhm dans le λ-calcul avec constructeurs pose le problème de la définition de l'ordre extensionnel qui va avec. Dans cette direction, le contre-exemple présenté à la section 3.4.6 nous donne une indication précieuse, à savoir qu'il convient de traiter les fonctions sous-identité de la forme

$$\lambda x \,.\, \{\!|c_1 \mapsto c_1; \ldots; c_p \mapsto c_p|\!\}\,.\,x$$

en tant que telles. Une définition possible de l'ordre extensionnel sur les arbres de Böhm du λ-calcul avec constructeurs pourrait donc être la suivante, en distinguant un ordre sur les arbres (noté $\leq_b$), un ordre sur les têtes (noté $\leq_h$) et un ordre sur les forêts (noté $\leq_\theta$) :

$$\frac{}{\bot \leq_b B} \qquad \frac{}{B \leq_b B} \qquad \frac{}{B \leq_b \Lsh}$$

$$\frac{H \leq_h H' \qquad B_1 \leq_b B'_1 \quad \cdots \quad B_k \leq_b B'_k}{\lambda x_1 \cdots x_n \,.\, H B_1 \cdots B_k \;\leq_b\; \lambda x_1 \cdots x_n \,.\, H' B'_1 \cdots B'_k}$$

$$\frac{}{H \leq_h H} \qquad \frac{B_1 \leq_b c_1 \quad \cdots \quad B_p \leq_b c_p}{\{\!|c_1 \mapsto B_1; \ldots; c_p \mapsto B_p|\!\}\,.\,x \;\leq_h\; x}$$

$$\frac{B_1 \leq_b B'_1 \quad \cdots \quad B_p \leq_b B'_p}{(c_1 \mapsto B_1; \ldots; c_p \mapsto B_p) \;\leq_\theta\; (c_1 \mapsto B'_1; \ldots; c_p \mapsto B'_p)}$$

(En considérant la notion de dérivabilité induite par ces règles au sens d'une définition co-inductive, et en travaillant à η-équivalence près.) Reste à déterminer si la relation d'ordre définie ci-dessus capture effectivement les propriétés de séparation du calcul — ce qui est loin d'être évident.

Dans la même optique, il serait sans doute intéressant de définir une ou plusieurs sémantiques dénotationnelles pour le calcul (basées sur les domaines de Scott et ses variantes), sachant qu'une étude abstraite des domaines susceptibles d'interpréter le calcul pourrait en retour nous donner des indications sur la définition correcte de l'ordre extensionnel sur les arbres de Böhm associés au termes du λ-calcul avec constructeurs.[11]

[11] Une autre piste consisterait à dissocier l'échec de filtrage (représenté dans le calcul par le terme $\{\!|\}\,.\,c$) de la divergence (représenté par le terme $(\lambda x \,.\, xx)(\lambda x \,.\, xx)$), ce qui nécessiterait en retour d'apporter des modifications profondes au formalisme dans la mesure où l'on souhaite conserver un résultat de séparation.

Chapitre 4

Réalisabilité classique

Ce chapitre est consacré à plusieurs résultats autour de la réalisabilité classique de Krivine [60, 61, 62, 63], et notamment son extension au calcul des constructions avec univers. Comme la réalisabilité classique est un sujet assez peu connu, je commencerai par rappeler le cadre général à la section 4.1. J'aborderai ensuite ce qui constitue le fil conducteur du chapitre — le problème de l'extraction de témoin à partir d'une preuve d'un énoncé existentiel en logique classique (section 4.2) — en mettant en évidence les liens entre la méthode de Krivine et la méthode d'extraction (due à Kreisel et Friedman) basée sur une traduction négative de la logique classique vers la logique intuitionniste. Je montrerai ensuite comment le modèle de réalisabilité classique de Krivine (défini à l'origine pour l'arithmétique classique du second ordre) peut être étendu au calcul des constructions avec univers et principe du tiers-exclu (section 4.3). J'expliquerai enfin comment ces idées sont mises en œuvre dans le module d'extraction classique de Coq (section 4.4).

Une partie du matériel présenté dans ce chapitre a été publiée dans [77, 79]. Un chapitre de livre concernant le théorème d'effectivité expérimentale présenté à la section 4.2.6 est en cours de préparation.

4.1 Une introduction à la réalisabilité classique

4.1.1 Présentation des concepts

La théorie de la réalisabilité classique [62] a été introduite par Krivine dans les années 1990 pour explorer le champ de recherches ouvert par l'extension de la correspondance preuves-programmes à la logique classique suite à la découverte de Felleisen et Griffin sur les liens entre les opérateurs de contrôle et la logique classique (voir section 1.2.3) et aux travaux de Parigot sur le $\lambda\mu$-calcul [85]. Techniquement, la réalisabilité classique est basée sur une extension du λ-calcul avec l'opérateur *call/cc*, un calcul qu'on peut voir également comme une version simplifiée du $\lambda\mu$-calcul où l'on n'aurait conservé que le minimum nécessaire pour faire « tourner » la logique classique.

La réalisabilité classique est un cadre très flexible. Bien qu'elle ait été définie à l'origine pour l'arithmétique classique du second ordre, il est possible de l'étendre à l'arithmétique d'ordre supérieur et même à la théorie des ensembles de Zermelo-Fraenkel [60]. Par ailleurs, le calcul des réalisateurs sup-

porte un grand nombre d'extensions (sous la forme d'instructions ajoutées au formalisme) de manière à réaliser des extensions de la logique classique, et notamment diverses formes de l'axiome du choix [61]. Plus récemment, Krivine a même montré [63] qu'il est possible de combiner la théorie de la réalisabilité classique avec la méthode du *forcing* [23, 24, 12, 59] pour réaliser des axiomes supplémentaires tels que l'hypothèse du continu ou sa négation.

Un changement de paradigmes Bien que la théorie ait acquis une certaine maturité scientifique au cours des quinze dernières années, la réalisabilité classique est encore aujourd'hui assez peu étudiée, le plus souvent parce qu'elle est jugée difficile d'accès. La difficulté du sujet réside pourtant moins dans sa technicité (très surestimée) que dans le fait que son étude nécessite une remise en cause, parfois très profonde, d'un certain nombre de paradigmes (et d'automatismes intellectuels) auxquels une longue pratique de la logique intuitionniste nous a habitués. Pour bien comprendre la théorie de la réalisabilité classique, nous allons donc commencer par isoler un certain nombre de concepts et d'outils qui occupent une place centrale dans l'étude de la logique intuitionniste, mais qui, dans un cadre classique, tiennent un rôle plus secondaire — et dans certains cas deviennent même complètement inopérants.

La réalisabilité plutôt que le typage La première difficulté (qui n'est pas directement liée au passage à la logique classique) vient du choix d'aborder les problèmes sous l'angle de la réalisabilité plutôt que sous l'angle du typage. S'il est vrai que d'un point de vue technique, la réalisabilité est au typage ce que la théorie des modèles est à la prouvabilité, le choix de la réalisabilité est ici dicté par une volonté d'étudier les programmes en fonction de leur comportement calculatoire plutôt qu'en fonction de la façon dont ces programmes sont construits (voir à ce sujet la discussion de la section 1.2.2).

Dans un tel cadre, le typage peut constituer un point de départ possible (dans la mesure où un terme de preuve d'une formule est aussi un réalisateur de cette formule), mais il ne constitue jamais le point d'arrivée, qui reste l'étude de la dynamique calculatoire des programmes attachés aux formules à travers la relation de réalisabilité. L'un des problèmes centraux de la réalisabilité classique est le problème de la spécification, qui consiste à déterminer, pour une formule A donnée, le comportement calculatoire commun à tous les réalisateurs de A (ce qui inclut les termes de preuves de A).

De ceci on retiendra deux choses.

D'abord, que l'ensemble des réalisateurs (classiques ou intuitionnistes) d'une formule A est considérablement plus vaste que l'ensemble des termes de preuves (classiques ou intuitionnistes) de la formule A. En pratique, il est souvent possible d'associer directement un réalisateur à une formule vraie (dans le modèle) sans avoir la moindre idée d'une preuve de cette formule[1] Et même dans le cas où l'on dispose d'un terme de preuve de A, il est souvent plus facile de construire à la main un réalisateur beaucoup plus concis.

La deuxième chose (assez déconcertante au début) est que contrairement à ce qu'on observe dans le cadre de la prouvabilité, la réalisabilité classique n'est *pas* une extension de la réalisabilité intuitionniste — elle repose en réalité sur

[1] Par exemple, la formule $\forall^{\mathbb{N}} x\ \forall^{\mathbb{N}} y\ \forall^{\mathbb{N}} z\ \forall^{\mathbb{N}} n\ (x+1)^{n+3} + (y+1)^{n+3} \neq (z+1)^{n+3}$ admet un réalisateur qui tient dans cette marge : $\lambda xyznu\,.\,uu$.

une construction de modèle très différente de celle qui définit la réalisabilité intuitionniste. En pratique, cela signifie qu'un réalisateur intuitionniste d'une formule donnée (c'est-à-dire : un réalisateur défini dans un modèle de réalisabilité intuitionniste) n'est pas forcément un réalisateur correct au sens de la réalisabilité classique.[2] La situation est résumée dans le tableau suivant :

$$\begin{array}{ccc} \text{Preuves intuitionnistes de } A & \subset & \text{Preuves classiques de } A \\ \cap & & \cap \\ \text{Réalisateurs intuitionnistes de } A & \not\supset & \text{Réalisateurs classiques de } A \end{array}$$

(Toutes les inclusions ci-dessus sont strictes.)

De la normalisation à l'évaluation Dans le λ-calcul et ses extensions intuitionnistes[3], le calcul est organisé autour des notions de *réduction* et de *normalisation*. Un programme « exécutable » est représenté par un terme clos dont l'évaluation peut s'effectuer en dehors de toute référence à un objet extérieur (le contexte). Il n'est donc pas étonnant que dans un tel cadre, la notion de *forme normale* (c'est-à-dire un terme où il n'est plus possible d'effectuer le moindre calcul) coïncide avec la notion de *valeur* (c'est-à-dire un programme ayant la forme d'une structure de donnée), au moins dans le cas où l'on se restreint aux programmes vivant dans les types de données. Dans la mesure où la valeur calculée par un programme est implicitement contenue dans ce programme, et seulement dans ce programme, il est naturel de faire en sorte que l'exécution aboutisse à une valeur — et toujours la même valeur — indépendamment de la stratégie d'évaluation. D'où l'importance de la notion de *confluence* (qui garantit l'unicité de la forme normale, et donc de la valeur) et de la notion de *normalisation forte* (qui garantit la terminaison du calcul quel que soit l'ordre dans lequel les étapes de réduction sont effectuées).

Dans le λ_c-calcul (c'est-à-dire dans le λ-calcul étendu avec *call/cc*), il n'est plus possible de définir le calcul à partir d'une relation de réduction sur les termes, car l'opérateur de contrôle *call/cc* a besoin de connaître le contexte d'évaluation courant pour effectuer son travail. Pour cette raison, le calcul est défini à partir d'une relation d'*évaluation* sur les *processus*, c'est-à-dire sur des entités constituées d'un terme clos accompagné d'une pile, qui représente ici le contexte d'évaluation courant. Dans un tel cadre, il est clair que la notion de forme normale (quel que soit le sens donné à cette expression) ne peut plus coïncider avec la notion de valeur, ce qui réduit considérablement l'intérêt des preuves de normalisation.[4] Comme en outre l'évaluation suit une stratégie fixée (en réalisabilité classique : l'appel par nom), la propriété de Church-Rosser (sur les termes) perd toute sa pertinence, et il devient alors possible d'envisager

[2] Par exemple, le terme $\lambda uv\,.\,v$ est un réalisateur de la formule $s(x) = s(y) \Rightarrow x = y$ dans un modèle de réalisabilité intuitionniste (en définissant l'égalité au second ordre), mais ce n'est plus un réalisateur de cette formule dans le modèle de réalisabilité de Krivine. (En revanche, le terme $\lambda u\,.\,u$ est un réalisateur correct dans les deux modèles.)

[3] C'est-à-dire toutes les extensions liées à la théorie de la réécriture : les couples, les enregistrements, le filtrage, etc. par opposition aux extensions basées sur l'utilisation d'un contexte externe au calcul, comme par exemple les opérateurs de contrôle.

[4] À ce titre, il est intéressant de souligner que même dans le $\lambda\mu$-calcul, où le calcul est pourtant défini à partir de la notion de réduction, une forme normale de type Nat (défini au second ordre) ne correspond pas forcément à un entier de Church (c'est-à-dire : une valeur de type Nat). Même dans ce cadre, il est nécessaire d'utiliser d'autres outils pour pouvoir extraire un « vrai » entier naturel à partir d'une preuve d'un énoncé de la forme $\exists^{\mathbb{N}} x\ A(x)$.

l'extension du calcul avec de nouvelles primitives qui n'auraient aucun sens dans un cadre où la confluence est requise. (Par exemple : le calcul du temps écoulé depuis le début de l'exécution, ou la récupération du code de Gödel d'un terme suivant une énumération fixée à l'avance.)

Un rapprochement avec la programmation usuelle Il est intéressant de noter que la situation que nous venons de décrire est très familière dans le cadre de la programmation usuelle, car la plupart des langages de programmation intègrent depuis longtemps des primitives qui permettent au programme de communiquer avec le monde extérieur. Dans ces langages, les notions de réduction et de normalisation n'ont d'intérêt que dans le cadre très restreint de l'évaluation *partielle*, tant il est illusoire de chercher à récupérer une valeur à partir de la forme normale d'un terme comme « `input_int stdin` », quel que soit le sens accordé à la notion de forme normale dans un tel cadre.

L'expérience de la programmation nous enseigne par ailleurs que l'adoption d'une stratégie d'évaluation fixée n'a pas que des mauvais côtés, puisque c'est elle qui nous permet d'enrichir à volonté le langage avec des primitives définissant de nouvelles interactions avec le système ou le monde extérieur. De fait, la réalisabilité classique utilise déjà de telles extensions (notamment l'horloge et l'instruction *quote* [61]) pour réaliser diverses formes de l'axiome du choix. Loin du modèle d'exécution fermé du λ-calcul et de ses extensions intuitionnistes, la réalisabilité classique repose sur un modèle d'exécution ouvert, qui permet non seulement d'envisager une plus grande intégration des traits de programmation usuels dans le cadre de la correspondance preuves-programmes, mais également l'extension de cette correspondance en dehors du cadre strict des mathématiques. (Nous reviendrons en détail sur ce point à la section 4.2.6.)

L'irruption des parapreuves et ses conséquences La correspondance de Curry-Howard dans le cas intuitionniste nous a habitués à l'idée qu'une formule fausse est essentiellement un type « vide », c'est-à-dire un type qui n'a pas d'habitant clos. Comme nous allons le voir, la situation est très différente dès qu'on se place dans un cadre classique.

Dans le modèle de réalisabilité classique défini par Krivine, les termes jouent le rôle des *preuves* d'une certaine formule tandis que les piles (c'est-à-dire les contextes) jouent le rôle des *contre-preuves* de la même formule.[5] Un processus, formé par un terme clos accompagné d'une pile (qui est également un objet clos) représente donc une *contradiction*, au sein de laquelle la preuve et la contre-preuve « débattent » en s'échangeant des informations au cours de l'évaluation. (On notera que la formule « débattue » change en permanence au cours de l'évaluation ; ce qui ne change pas, c'est que le terme et la pile débattent toujours de la même formule, l'un pour la défendre, l'autre pour la réfuter.)

Dans un tel cadre, il n'est évidemment pas possible de faire s'affronter une « preuve » et une « contre-preuve » de la même formule sans se placer dans un contexte incohérent, au moins localement. La réalisabilité classique résout ce paradoxe en remplissant l'espace des réalisateurs avec un grand nombre de *parapreuves* — c'est-à-dire des réalisateurs (clos) de la proposition fausse — dont

[5] Ici, les mots « preuve » et « contre-preuve » sont à prendre au sens figuré dans la mesure où une formule a beaucoup plus de réalisateurs (et de contre-réalisateurs) que de preuves ou de contre-preuves au sens formel du terme.

la seule fonction est de porter la contradiction en toute circonstance. Techniquement, ces parapreuves apparaissent sous la forme de constantes de continuation k_π (où π est une pile quelconque) qui servent à récupérer dans le monde des preuves le point de vue de l'opposant (ici : la pile π). Bien entendu, de telles parapreuves ne correspondent à aucune démonstration formelle, et leur présence dans le modèle de réalisabilité ne remet pas en cause la cohérence du formalisme — qui est une propriété des « vraies » preuves, pas des réalisateurs.

La présence de parapreuves a cependant une conséquence très importante, qui est que l'existence d'un réalisateur n'atteste en rien que la formule réalisée est vraie. Nous verrons à la section 4.2 qu'un réalisateur classique d'une formule existentielle de la forme $\exists^{\mathbb{N}} x\, A(x)$ produit, comme en réalisabilité intuitionniste, un couple formé d'un « témoin » $n \in \mathbb{N}$ et d'un réalisateur de $A(n)$. Mais à la différence de la réalisabilité intuitionniste, la présence d'un réalisateur de $A(n)$ ne signifie pas que cette formule est vraie, et que le témoin proposé est fiable. Nous verrons comment il est possible (dans certains cas) de débusquer les faux-témoins et de retourner contre eux le faux-témoignage qui les accompagne afin d'obtenir la production d'un nouveau témoin.

4.1.2 Le calcul des réalisateurs

Les *termes* de λ_c (notation : t, u, etc.) sont les termes du λ-calcul enrichi avec des constantes de deux sortes, à savoir :

- Les *instructions*, parmi lesquelles figure une instruction particulière notée cc et appelée « *call/cc* » (pour : *call with current continuation*).
- Les *constantes de continuation* k_π, qui sont en bijection avec les piles π.

Les *piles* de λ_c (notation : π, π', π_1, etc.) sont quant à elles des listes de termes clos terminées par une constante de pile (notation : α, β, etc.)

Formellement, les termes (ouverts) et les piles (closes) du langage λ_c sont donc définis par induction mutuelle à partir de la BNF suivante :

$$\begin{array}{lrcl} \textbf{Termes} & t, u & ::= & x \mid \lambda x\,.\,t \mid tu \mid \mathsf{cc} \mid \cdots \mid \mathsf{k}_\pi \\ \textbf{Piles} & \pi & ::= & \alpha \mid t \cdot \pi \qquad (t \text{ clos}) \end{array}$$

(où « $\cdots$ » désigne l'emplacement des instructions supplémentaires.) L'ensemble des termes clos est noté Λ tandis que l'ensemble des piles (closes) est noté Π.

Dans ce qui suit, on dit qu'un terme (éventuellement ouvert) est une *quasi-preuve* s'il ne contient aucune constante de continuation k_π. Cette terminologie vient du fait que les termes issus d'une vraie preuve (au sens qui sera défini dans les sections 4.1.5 et 4.1.7) sont tous de cette forme. On peut également comprendre les quasi-preuves comme les termes qui ne sont pas « corrompus » par le point de vue de l'opposant (représenté par les constantes k_π).

Processus et évaluation Un processus (notation : p, q, etc.) est un couple $p \equiv t \star \pi$ formé par un terme clos $t \in \Lambda$ et une pile Π :

$$\textbf{Processus} \qquad p, q \quad ::= \quad t \star \pi \qquad (t \in \Lambda,\ \pi \in \Pi)$$

L'ensemble des processus est noté $\Lambda \star \Pi$.

On suppose en outre que cet ensemble est muni d'une relation d'*évaluation*,

notée $p \succ p'$, qui satisfait les axiomes suivants :

GRAB	$\lambda x\,.\,t$	$\star$	$u \cdot \pi$	$\succ$	$t\{x := u\}$	$\star$	π
PUSH	tu	$\star$	π	$\succ$	t	$\star$	$u \cdot \pi$
SAVE	cc	$\star$	$t \cdot \pi$	$\succ$	t	$\star$	$\mathsf{k}_\pi \cdot \pi$
RESTORE	k_π	$\star$	$t \cdot \pi'$	$\succ$	t	$\star$	π

(La relation d'évaluation n'est pas définie mais axiomatisée pour laisser la place aux règles d'évaluation associées à d'éventuelles instructions supplémentaires.)

4.1.3 Le langage de la logique du second ordre

On se donne un ensemble infini de variables du premier ordre (notation : x, y, z, etc.) et, pour chaque entier $k \geq 0$, un ensemble infini de variables du second ordre d'arité k (notation : X, Y, Z, etc.) On omettra le plus souvent d'indiquer l'arité des variables du second ordre, celle-ci pouvant être facilement inférée à partir du contexte. Le langage de la logique (minimale) du second ordre distingue deux classes d'expressions : les *expressions arithmétiques* (ou *termes du premier ordre*) et les *formules*.

Les expressions arithmétiques (notation : e, e_1, e', etc.) sont construites à partir des variables du premier ordre à l'aide des symboles de constante et de fonction d'une signature fixée :

Expressions arithmétiques $\quad e, e' \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_k)$

Dans tout ce qui suit, on se place dans une signature où chaque symbole de fonction d'arité k est associé à la définition d'une fonction primitive récursive à k arguments. On supposera en particulier que cette signature contient les symboles 0 (zéro), s (successeur), $+$ (addition), $\times$ (multiplication), etc. Les notations $FV(e)$ (variables libres de e) et $e\{x := e'\}$ (substitution) sont définies de la manière habituelle.

Les formules de la logique du second ordre sont construites à partir des variables du second ordre et des expressions arithmétiques à l'aide de l'application (d'une variable de prédicat à ses arguments), de l'implication et des quantifications universelles du premier et du second ordre :

Formules $\quad A, B \quad ::= \quad X(e_1, \ldots, e_k) \quad | \quad A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X A$

L'ensemble des variables libres (du premier et du second ordre) de la formule A est noté $FV(A)$. On note respectivement $A\{x := e\}$ et $A\{X := \hat{x}_1 \cdots \hat{x}_k.B\}$ le résultat de la substitution du premier et du second ordre. On rappelle que cette dernière opération consiste à remplacer dans A toutes les sous-formules atomiques de la forme $X(e_1, \ldots, e_k)$ par la formule $B\{x_1 := e_1; \ldots; x_k := e_k\}$. (Voir [44, 57] pour une présentation plus formelle.)

Comme d'habitude, les autres constructions logiques (absurdité, négation, conjonction, disjonction, quantifications existentielles du premier et du second ordre, ainsi que l'égalité de Leibniz) sont définies par des codages au second ordre. On utilise notamment l'abréviation

$$\mathsf{Nat}(e) \quad \equiv \quad \forall Z\ (Z(0) \Rightarrow \forall x\,(Z(x) \Rightarrow Z(s(x))) \Rightarrow Z(e))$$

qui permet de définir les quantifications numériques

$$\begin{array}{lcl} \forall^{\mathsf{N}} x\, A(x) & \equiv & \forall x\ (\mathsf{Nat}(x) \Rightarrow A(x)) \\ \exists^{\mathsf{N}} x\, A(x) & \equiv & \forall Z\ (\forall^{\mathsf{N}} x\,(A(x) \Rightarrow Z) \Rightarrow Z) \end{array}$$

4.1.4 La définition du modèle de réalisabilité

Le pôle La construction du modèle de réalisabilité est paramétrée par un ensemble de processus $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$, qu'on appelle le *pôle*. On suppose de cet ensemble qu'il est *saturé* (ou encore : *clos par antiréduction*) en ce sens que pour tous $p, p' \in \Lambda \star \Pi$, les conditions $p \succ p'$ et $p' \in \perp\!\!\!\perp$ entraînent que $p \in \perp\!\!\!\perp$.

Intuitivement, cet ensemble définit la notion de contradiction dans le modèle de réalisabilité considéré.

Valeur d'une expression arithmétique Pour chaque expression arithmétique close e, on note $\downarrow e$ la *valeur* de l'expression e (calculée en appliquant les équations définissantes des symboles de fonction figurant dans e), qui n'est autre que la dénotation de e dans le modèle standard de l'arithmétique.

Valeur de vérité et valeur de fausseté d'une formule À chaque formule close A de la logique du second ordre, on associe deux ensembles, à savoir un ensemble de termes noté $|A| \subseteq \Lambda$ (la *valeur de vérité* de A) et un ensemble de piles noté $\|A\| \subseteq \Pi$ (la *valeur de fausseté* de A).

En réalisabilité classique, la valeur de fausseté $\|A\|$ d'une formule A est primitive tandis que sa valeur de vérité $|A|$ est définie par orthogonalité (par rapport au pôle $\perp\!\!\!\perp$) en posant

$$|A| \quad = \quad \|A\|^{\perp\!\!\!\perp} \quad = \quad \{t \in \Lambda \ : \ \forall \pi \in |A| \ \ (t \star \pi \in \perp\!\!\!\perp)\}\,.$$

Formules à paramètres Le langage des *formules à paramètres* est obtenu en ajoutant au langage des formules de la logique du second ordre un symbole de prédicat $\dot{F}$ d'arité k pour chaque fonction $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$[6] :

Formules à paramètres $\qquad A, B \quad ::= \quad \cdots \quad | \quad \dot{F}(e_1, \ldots, e_k)$

(Intuitivement, $\dot{F}$ désigne le prédicat dont la valeur de fausseté est définie par la « fonction de fausseté » $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$.)

Les notations $FV(A)$, $A\{x := e\}$ et $A\{X := \hat{x}_1 \cdots \hat{x}_k.B\}$ s'étendent immédiatement au langage des formules avec paramètres de la manière attendue.

Valuations Une valuation est une fonction ρ qui à chaque variable du premier ordre x associe un entier naturel $\rho(x) \in \mathbb{N}$ et qui à chaque variable du second ordre X d'arité k associe une fonction de fausseté $\rho(X) : \mathbb{N}^k \to \mathfrak{P}(\Pi)$.

Étant donnée une formule ouverte A (éventuellement à paramètres) et une valuation ρ, on note $A[\rho]$ la clôture de la formule A par la valuation ρ. (Le résultat de cette opération est une formule close à paramètres).

Valeur de fausseté d'une formule à paramètres Formellement, la valeur de fausseté $\|A\|$ d'une formule close A à paramètres est définie par récurrence

[6] Cette extension de la syntaxe fait bien évidemment passer le cardinal de l'ensemble des formules du dénombrable au continu. En passant des formules aux formules avec paramètres, on a déjà parcouru la majeure partie du chemin qui mène de la syntaxe à la sémantique.

sur la taille de cette formule à l'aide des équations :

$$\begin{aligned}
\|\dot{F}(e_1,\dots,e_k)\| &= F(\downarrow e_1,\dots,\downarrow e_k)\\
\|A\Rightarrow B\| &= |A|\cdot\|B\| \;=\; \{t\cdot\pi \;:\; t\in|A|,\ \pi\in\|B\|\}\\
\|\forall x\,A\| &= \bigcup_{n\in\mathbb{N}}\|A\{x:=n\}\|\\
\|\forall X A\| &= \bigcup_{F:\mathbb{N}^k\to\mathfrak{P}(\Pi)}\|A\{X:=\dot{F}\}\|
\end{aligned}$$

On rappelle que la valeur de vérité $|A|$ est définie quant à elle par

$$|A| \;=\; \|A\|^{\perp\!\!\!\perp} \;=\; \{t\in\Lambda \;:\; \forall\pi\in\|A\|\ \ (t\star\pi\in\perp\!\!\!\perp)\}\,.$$

Comme les ensembles $|A|$ et $\|A\|$ dépendent du pôle $\perp\!\!\!\perp$, on les note parfois $|A|_{\perp\!\!\!\perp}$ et $\|A\|_{\perp\!\!\!\perp}$ pour indiquer explicitement cette dépendance.

Étant donnée une formule close A (éventuellement à paramètres) et un terme clos $t\in\Lambda$, on dit que :

- t *réalise* A (notation : $t\Vdash A$) si $t\in|A|$ (cette notion dépend du pôle $\perp\!\!\!\perp$) ;
- t est un *réalisateur universel* de A (notation : $t\Vdash\!\!\!\Vdash A$) si $t\in|A|_{\perp\!\!\!\perp}$ pour tout pôle $\perp\!\!\!\perp$.

Exemples On vérifie aisément les égalités suivantes :

$$\begin{aligned}
\|\bot\| \;=\; \|\forall X X\| &= \Pi \quad\text{(fausseté maximale)}\\
\|\mathbf{1}\| \;=\; \|\forall X(X\Rightarrow X)\| &= \{u\cdot\pi \;:\; (u\star\pi)\in\perp\!\!\!\perp\}
\end{aligned}$$

Par symétrie avec la formule $\bot\equiv\forall X X$, il est commode d'étendre le langage avec une constante propositionnelle $\top$ avec pour dénotation $\|\top\|=\varnothing$ (fausseté minimale). Dans le langage des formules avec paramètres, cette constante est définissable par $\top=\dot{\varnothing}$. (On fera attention à ne pas confondre la formule $\top$ avec la formule $\mathbf{1}\equiv\forall X(X\Rightarrow X)$.)

À l'instar de la réalisabilité intuitionniste, l'appartenance d'un terme à la valeur de vérité d'une formule est déterminée par son comportement calculatoire. En utilisant les règles de réduction des constantes cc et k_π, on démontre facilement que pour toutes formules closes A et B (éventuellement à paramètres) et pour toute pile π :

1. Si $\pi\in\|A\|$, alors $\mathsf{k}_\pi\Vdash A\Rightarrow B$;
2. Le terme cc est un réalisateur universel de $((A\Rightarrow B)\Rightarrow A)\Rightarrow A$.

Modèle induit par le pôle vide Dans le cas particulier où $\perp\!\!\!\perp=\varnothing$ (pôle vide), le modèle de réalisabilité qui en découle « mime » le modèle standard de l'arithmétique du second ordre, où les individus sont interprétés par les entiers naturels et où les variables du second ordre d'arité k sont interprétées par toutes les relations d'arité k sur $\mathbb{N}$ (i.e. le modèle plein [57]) :

Proposition 25 — *Si $\perp\!\!\!\perp=\varnothing$, alors pour toute formule close A on a*

$$|A| \;=\; \begin{cases}\Lambda & \textit{si } A \textit{ est vraie}\\ \varnothing & \textit{si } A \textit{ est fausse}\end{cases}$$

dans le modèle standard de l'arithmétique du second ordre.

(Ce résultat se généralise aux formules closes à paramètres, en convenant que la formule atomique $\dot{F}(n_1, \ldots, n_k)$ est vraie si la valeur de fausseté $F(n_1, \ldots, n_k)$ est vide, et qu'elle est fausse sinon.)

Une conséquence immédiate de ce résultat est la suivante :

Corollaire 4 (Validité) — *Toute formule close qui admet un réalisateur universel (i.e. ne dépendant pas de $\perp\!\!\!\perp$) est vraie dans le modèle standard de l'arithmétique du second ordre.*

Dans le cas où $\perp\!\!\!\perp \neq \varnothing$, il est facile de voir que n'importe quelle formule close est réalisée par tous les termes de la forme $\mathsf{k}_\pi t$, où $(t \star \pi) \in \perp\!\!\!\perp$.

4.1.5 Typage et lemme d'adéquation

Pour faciliter la construction des réalisateurs, on introduit un système de types basé un jugement de la forme $\Gamma \vdash t : A$, où t est un terme (ouvert), A une formule (ouverte et à paramètres), et où Γ est une liste finie de déclarations de la forme $x_i : A_i$ où les variables (de terme) x_i sont deux à deux distinctes. Les règles d'inférence qui définissent ce jugement sont données à la Fig. 4.1.

$$\frac{}{\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{}{\Gamma \vdash \mathsf{cc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x\, A}\ {\scriptstyle x \notin FV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall x\, A}{\Gamma \vdash t : A\{x := e\}}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X\, A}\ {\scriptstyle X \notin FV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall X\, A}{\Gamma \vdash t : A\{X := \hat{x}_1 \cdots \hat{x}_k.B\}}$$

FIG. 4.1 – Règles de typage pour la logique classique du second ordre

Le système de types ainsi défini est correct vis-à-vis du modèle de réalisabilité associé à n'importe quel pôle $\perp\!\!\!\perp$:

Proposition 26 (Adéquation) — *Si le jugement*

$$x_1 : A_1, \ldots, x_n : A_n \vdash t : B$$

est dérivable à partir des règles d'inférence de la Fig. 4.1, alors pour tout pôle $\perp\!\!\!\perp$, pour toute valuation ρ et pour tous termes $u_1 \Vdash A_1[\rho]$, ..., $u_n \Vdash A_n[\rho]$ on a $t\{x_1 := u_1; \ldots; x_n := u_n\} \Vdash B[\rho]$ (vis-à-vis du pôle $\perp\!\!\!\perp$).

Preuve. Voir [62].

Comme les règles d'inférence de la Fig. 4.1 implémentent les règles de déduction de la logique classique du second ordre (basée sur la loi de Peirce plutôt que le tiers-exclu), il est immédiat que :

Proposition 27 (Réalisation des tautologies) — *Toute formule close qui est dérivable dans le calcul des prédicats classique du second ordre admet un réalisateur universel qui est une quasi-preuve.*

Sous-typage Pour raisonner plus facilement sur les valeurs de vérité et de fausseté des formules, il est commode d'introduire une relation de sous-typage (sémantique) sur les formules qui est la suivante. Étant donné un pôle fixé, on dit qu'une formule A est un *sous-type* d'une formule B et on note $A \leq B$ lorsque pour toute valuation ρ on a l'inclusion $\|B[\rho]\| \subseteq \|A[\rho]\|$ — qui implique également l'inclusion $|A[\rho]| \subseteq |B[\rho]|$ par orthogonalité. Cette relation est un préordre sur les formules (dont l'équivalence associée est notée $A \approx B$) qui satisfait les règles de sous-typage habituelles, à savoir :

$$\frac{A \leq A' \qquad B \leq B'}{A' \Rightarrow B \ \leq A \Rightarrow B'} \qquad \frac{\Gamma \vdash t : A \qquad A \leq B}{\Gamma \vdash t : B}$$

$$\frac{}{\forall x\, A \ \leq \ A\{x := e\}} \qquad \frac{}{\forall X A \ \leq \ A\{X := \hat{x}_1 \cdots \hat{x}_k \,.\, B\}}$$

$$\frac{A \leq B}{A \ \leq \ \forall x\, B}\ {\scriptstyle x \notin FV(A)} \qquad \frac{A \leq B}{A \ \leq \ \forall X B}\ {\scriptstyle X \notin FV(A)}$$

$$\frac{}{\forall x\,(A \Rightarrow B) \ \approx \ A \Rightarrow \forall x\, B}\ {\scriptstyle x \notin FV(A)} \qquad \frac{}{\forall X (A \Rightarrow B) \ \approx \ A \Rightarrow \forall X B}\ {\scriptstyle X \notin FV(A)}$$

4.1.6 Réalisation des égalités

Il s'agit à présent d'étendre la proposition précédente (Prop. 27) à tous les *théorèmes* de l'arithmétique classique du second ordre. Pour cela, on doit d'abord étudier la dénotation de l'égalité de Leibniz dans le modèle de réalisabilité (pour un pôle quelconque) :

Lemme 10 (Dénotation de l'égalité) — *Soient e_1 et e_2 deux expressions arithmétiques closes. Dans le modèle de réalisabilité, on a :*

$$\|e_1 = e_2\| \ = \ \begin{cases} \{t \cdot \pi \ : \ (t \star \pi) \in \perp\!\!\!\perp\} & \text{si } \downarrow e_1 = \downarrow e_2 \\ \Lambda \cdot \Pi & \text{si } \downarrow e_1 \neq \downarrow e_2 \end{cases}$$

En termes d'équivalence de formules (au sens de la relation $A \approx B$), on a donc $(e_1 = e_2) \approx \mathbf{1} \equiv \forall X(X \Rightarrow X)$ si $e_1 = e_2$ est vraie, et $(e_1 = e_2) \approx (\top \Rightarrow \bot)$ sinon. Ce qui nous permet de démontrer que :

Lemme 11 (Réalisation des égalités) — *Soient deux expressions arithmétiques $e_1(x_1, \ldots, x_n)$ et $e_2(x_1, \ldots, x_k)$ dépendant des seules variables $x_1, \ldots, x_k$.*

1. *Si $\downarrow e_1(n_1, \ldots, n_k) = \downarrow e_2(n_1, \ldots, n_k)$ pour tous $n_1, \ldots, n_k \in \mathbb{N}$, alors :*

$$\lambda z \,.\, z \ \Vdash \ \forall x_1 \cdots \forall x_k \ e_1(x_1, \ldots, x_k) = e_2(x_1, \ldots, x_k)$$

2. *Si $\downarrow e_1(n_1, \ldots, n_k) \neq \downarrow e_2(n_1, \ldots, n_k)$ pour tous $n_1, \ldots, n_k \in \mathbb{N}$, alors :*

$$\lambda z \,.\, zu \ \Vdash \ \forall x_1 \cdots \forall x_k \ \neg(e_1(x_1, \ldots, x_k) = e_2(x_1, \ldots, x_k))$$

(où u est un terme tel que $FV(u) \subseteq \{x\}$).

Preuve. Car $\lambda x \,.\, x \in |\mathbf{1}|$ et $\lambda x \,.\, xu \in |(\top \Rightarrow \bot) \Rightarrow \bot|$. □

Le premier point entraîne que le terme identité $\lambda z\,.\,z$ réalise toutes les égalités définitionnelles des symboles de fonctions primitives récursives définis dans la signature. Le second point entraîne que le terme $\lambda z\,.\,zz$ (ou tout autre terme clos de la forme $\lambda z\,.\,zu$) réalise le 4e axiome de Peano : $\forall x\ \neg(s(x) = 0)$. On peut généraliser le premier point de la manière suivante :

Lemme 12 (Réalisation des égalités conditionnelles) — *Étant données des égalités $E_1(\vec{x}), \ldots, E_n(\vec{x})$ et $E_{n+1}(\vec{x})$ entre des expressions arithmétiques ne dépendant que des variables $\vec{x} = x_1, \ldots, x_k$, et telles que l'implication*

$$E_1(\vec{x}) \Rightarrow \cdots \Rightarrow E_n(\vec{x}) \Rightarrow E_{n+1}(\vec{x})$$

est uniformément vraie dans $\mathbb{N}^k$, on a

$$\lambda z\,.\,z \ \Vdash\ \forall \vec{x}\ (E_1(\vec{x}) \Rightarrow \cdots \Rightarrow E_n(\vec{x}) \Rightarrow E_{n+1}(\vec{x}))$$

Preuve. On raisonne par sous-typage, en démontrant successivement que

1. $(\top \Rightarrow \bot)\ \leq\ E_1(\vec{x}) \Rightarrow \cdots \Rightarrow E_n(\vec{x}) \Rightarrow E_{n+1}(\vec{x})$ (par récurrence sur n) ;
2. $\mathbf{1}\ \leq\ E_1(\vec{p}) \Rightarrow \cdots \Rightarrow E_n(\vec{p}) \Rightarrow E_{n+1}(\vec{p})$ pour tout $\vec{p} \in \mathbb{N}^k$ tel que cette implication est vraie dans le modèle (par récurrence sur n). □

Ce qui entraîne en particulier que le terme $\lambda z\,.\,z$ réalise également le 3e axiome de Peano : $\forall x\ \forall y\ (s(x) = s(y) \Rightarrow x = y)$.

4.1.7 Réalisation des théorèmes de PA2

La réalisation des théorèmes de l'arithmétique classique du second ordre se heurte à une première difficulté, qui est que le principe de récurrence — qui au second ordre s'écrit $\forall x\ \mathsf{Nat}(x)$ — n'a pas de réalisateur universel, du moins dans le cas où la relation d'évaluation est déterministe[7] :

Proposition 28 — *Si la relation d'évaluation $p \succ p'$ est déterministe, alors il n'existe aucun terme clos t tel que $t \Vdash \forall x\ \mathsf{Nat}(x)$.*

Preuve. Voir [62].

Il est intéressant de préciser que dans ce résultat l'hypothèse de déterminisme est absolument essentielle. En effet, si on étend le calcul des réalisateurs avec une instruction $\pitchfork$ (intuitivement : le booléen non déterministe) munie des règles d'évaluation non déterministes

$$\pitchfork \star u_1 \cdot u_2 \cdot \pi\ \succ\ u_1 \star \pi \qquad \text{et} \qquad \pitchfork \star u_1 \cdot u_2 \cdot \pi\ \succ\ u_2 \star \pi\,,$$

il est tout à fait possible de réaliser universellement la formule $\forall x\ \mathsf{Nat}(x)$, par exemple avec la quasi-preuve suivante, où Y est un combinateur de point fixe :

$$Y\,(\lambda z\,.\pitchfork\,(\lambda xy\,.\,x)\ (\lambda xy\,.\,y\,(z\,x\,y)))\,.$$

(Intuitivement, le programme ci-dessus produit de manière non déterministe tous les entiers de Church.) Cependant, si la présence d'instructions non déterministes telles que $\pitchfork$ ne remet pas en cause les principaux théorèmes de

[7] C'est-à-dire telle que $p \succ p_1$ et $p \succ p_2$ (en une étape) entraîne $p_1 \equiv p_2$.

réalisabilité classique[8], elle amoindrit considérablement l'intérêt pratique des mécanismes d'extraction de témoin existentiel présentés à la section 4.2[9].

Pour pouvoir réaliser les théorèmes de l'arithmétique classique du second ordre sans avoir à renoncer à l'utilisation d'une relation d'évaluation déterministe, il est nécessaire de recourir à l'astuce habituelle (cf [44, 62]) qui consiste à ne travailler qu'avec des formules *arithmétiques*, c'est-à-dire des formules dont toutes les quantifications du premier ordre sont relativisées au prédicat $\mathsf{Nat}(x)$ défini à la section 4.1.3.

Formellement, on procède de la manière suivante. Étant donnée une formule A de la logique du second ordre, on note A^{Nat} la formule obtenue à partir de A en relativisant toutes les quantifications universelles du premier ordre avec le prédicat $\mathsf{Nat}(x)$. Par définition, on a donc $(\forall x\, A(x))^{\mathsf{Nat}} \equiv \forall^{\mathsf{N}} x\, A^{\mathsf{Nat}}(x)$ et $(\exists x\, A(x))^{\mathsf{Nat}} \equiv \exists^{\mathsf{N}} x\, A^{\mathsf{Nat}}(x)$ (en utilisant les abréviations introduites à la section 4.1.3 et la définition usuelle du quantificateur existentiel du premier ordre). Cette opération de relativisation n'affecte ni les quantifications du second ordre ni les connecteurs logiques qu'elle se contente simplement de traverser.

On démontre alors que :

Théorème 5 (Réalisation des théorèmes de PA2) — *Si une formule A (close) est un théorème de l'arithmétique classique du second ordre (avec le schéma de récurrence sur les entiers), alors la formule A^{Nat} admet un réalisateur universel qui est une quasi-preuve.*

Preuve. Voir [62].

Remarque 1 — Ce résultat repose de manière essentielle sur le fait que tous les symboles de fonction définis dans la signature sont associés à une défini-

[8] L'hypothèse de déterminisme n'entre en jeu que dans le cas où on cherche à analyser finement le comportement calculatoire des réalisateurs de certaines formules. On trouvera des exemples de telles analyses dans [62] et [47].

[9] Malgré ses inconvénients en ce qui concerne l'extraction de témoin (cf section 4.2), l'ajout de l'instruction non déterministe $\pitchfork$ permet de réaliser un grand nombre de formules qu'on ne sait pas réaliser dans le calcul de base (et qui sont sans doute non réalisables dans ce cadre). Parmi ces formules, on peut citer l'axiome du choix sous la forme du théorème de Zermelo, qui affirme l'existence d'un bon ordre sur les objets du second ordre. (Il s'agit d'un résultat non publié, sans doute connu par Krivine.) La technique est relativement simple : on introduit dans le langage un symbole du troisième ordre $W(X,Y)$ (où X et Y sont des variables du second ordre), qu'on réalise de la manière suivante : $\|W(\dot{F},\dot{G})\| = \|\mathbf{1}\| = \|\forall X(X \Rightarrow X)\|$ si $F < G$, et $\|W(\dot{F},\dot{G})\| = \|\top \Rightarrow \bot\|$ sinon, où $F < G$ est une relation de bon ordre strict sur les valeurs de fausseté dans le modèle. Avec cette définition, il est facile de réaliser le principe d'induction bien fondée sur les objets du second ordre (exprimée à l'aide du symbole $W(X,Y)$) avec un combinateur de point fixe approprié (un λ-terme pur). Toute la difficulté réside dans la réalisation de la propriété de totalité, qui repose sur l'utilisation de l'instruction $\pitchfork$. Intuitivement, on utilise cette instruction pour lancer en parallèle trois processus correspondant aux cas où $F < G$, $F = G$ et $F > G$. La définition de la réalisabilité classique nous assure alors qu'un des trois programmes lancés en parallèle sera correct, les deux autres ayant toutes les chances de faire n'importe quoi puisqu'il sont « mal typés ». Ici, l'utilisation de l'instruction $\pitchfork$ sert donc à suppléer au fait que le programme n'a aucun moyen de savoir en cours d'exécution laquelle des trois alternatives est correcte — d'où la bifurcation. (La méthode fonctionne également à l'ordre supérieur, et sans doute aussi en théorie des ensembles de Zermelo-Fraenkel [60].) Cet exemple montre la puissance de l'instruction $\pitchfork$, dont je conjecture qu'elle permet plus généralement de réaliser n'importe quelle formule de la logique du second ordre, pourvu que cette formule soit vraie dans le modèle standard. (Si c'est le cas, alors les modèles induits par la réalisabilité classique [62] sont élémentairement équivalents au modèle initial, ce qui leur fait perdre tout intérêt dans ce cadre.)

tion primitive récursive (ou plus généralement, à une définition récursive dont la totalité est prouvable en logique du second ordre). En effet, pour pouvoir éliminer une quantification universelle numérique $\forall^{\mathsf{N}} x\, A(x)$ avec une expression arithmétique e, il est nécessaire de disposer d'un réalisateur universel de la formule $\mathsf{Nat}(e)$ qui soit une quasi-preuve. Ceci n'est possible que si on dispose pour chaque symbole de fonction f figurant dans l'expression e d'un réalisateur universel de la formule $\forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k\ \mathsf{Nat}(f(x_1, \ldots, x_k))$. En pratique, on construit ce réalisateur à partir d'une preuve (en logique du second ordre) de la totalité de la fonction f, qui s'effectue à l'aide de la théorie équationnelle de f. Comme d'un point de vue calculatoire, ce réalisateur n'est rien d'autre qu'un λ_c-terme calculant la fonction f, on peut construire directement le réalisateur à partir d'une implémentation de la fonction f en $\lambda_{(c)}$-calcul.

4.1.8 Conclusion

Dans les pages qui précèdent, nous avons distingué trois niveaux de « vérité », à savoir : la prouvabilité dans PA2, l'existence d'un réalisateur universel (sous forme de quasi-preuve), et enfin la notion de validité dans le modèle standard. Ces trois niveaux sont reliés par les implications suivantes :

$$\text{Prouvable} \;\Rightarrow\; \text{Universellement réalisable} \;\Rightarrow\; \text{Valide (modèle standard)}.$$

Comme à plusieurs reprises nous avons montré que certaines formules vraies dans le modèle standard disposent d'un réalisateur universel, il est légitime de se demander dans quelle mesure ce résultat peut s'étendre à une plus large classe de formules. Dans ce cadre, Krivine a montré le théorème suivant :

Théorème 6 (Réalisabilité des formules de PA) — *Si une formule A de l'arithmétique du premier ordre est vraie dans le modèle standard, alors la formule A^{Nat} admet un réalisateur universel qui est une quasi-preuve.*

Preuve. Voir [62].

Plus récemment, Krivine a étendu ce résultat aux formules Π^1_1, c'est-à-dire aux formules de PA avec un unique paramètre du second ordre. La question de savoir jusqu'où ce résultat s'étend dans la hiérarchie Π^1_n/Σ^1_n est ouverte.

4.2 Extraction de témoin en logique classique

Dans la section 4.1 nous avons introduit les bases de la réalisabilité classique, et montré comment il est possible de construire un réalisateur universel d'une formule à partir d'une preuve de cette formule en arithmétique classique du second ordre. Dans cette section, nous allons maintenant nous intéresser à l'exploitation des réalisateurs classiques (construits par exemple à l'aide du Théorème 5), notamment pour extraire un témoin à partir d'un réalisateur (universel) d'une formule existentielle numérique $\exists^{\mathsf{N}} x\, A(x)$.

Certains résultats présentés ici font partie du « folklore » de la réalisabilité classique. C'est le cas notamment de la théorie des opérateurs de mise en mémoire [58] (rappelée à la section 4.2.1) et de la méthode d'extraction de témoin à partir d'un réalisateur d'une formule Σ^0_1 (section 4.2.3), dont on trouve des

utilisations dans [62] et [47]. La méthode d'extraction « du kamikaze » (section 4.2.4) est à ma connaissance originale, de même que l'utilisation des entiers primitifs (section 4.2.5) à la place des entiers de Krivine.[10]

Mon intérêt pour la réalisabilité classique a été au départ suscitée par un problème posé par le philosophe Karl Popper — le problème du *modus tollens expérimental* — dont je présente une solution (techniquement évidente, mais à mon avis épistémologiquement profonde) à la section 4.2.6. Je conclurai cette section en précisant les liens entre la méthode d'extraction décrite à la section 4.2.3 (pour les formules Σ^0_1) et la méthode, plus connue dans le monde de la théorie de la démonstration, qui consiste à passer en réalisabilité intuitionniste à travers une A-traduction (suivant la méthode due à Kreisel et Friedman).

4.2.1 Entiers de Krivine et opérateurs de mise en mémoire

Entiers de Krivine En réalisabilité classique, les entiers naturels ne sont pas représentés par des entiers de Church mais par des entiers de Krivine, qui sont définis par $\bar{n} = \bar{s}^n\bar{0}$ pour tout $n \in \mathbb{N}$, avec $\bar{0} \equiv \lambda xy\,.\,x$ et $\bar{s} \equiv \lambda nxy\,.\,y(nxy)$. Bien entendu, l'entier de Krivine $\bar{n}$ est β-convertible avec l'entier de Church correspondant, et agit donc calculatoirement de la même façon, c'est-à-dire comme un itérateur. Le changement de représentation (cosmétique) n'est ici dû qu'à l'utilisation des opérateurs de mise en mémoire — qui servent à récupérer l'entier n à partir d'un réalisateur classique de $\mathsf{Nat}(n)$ — dont on ne dispose pas de version capable de produire des entiers de Church.[11]

D'après la définition du prédicat $\mathsf{Nat}(x)$ et du jugement de typage de la section 4.1.5, on a $\vdash \bar{0} : \mathsf{Nat}(0)$, $\vdash \bar{s} : \forall^{\mathsf{N}} x\, \mathsf{Nat}(s(x))$ et finalement $\vdash \bar{n} : \mathsf{Nat}(n)$ pour tout entier $n \in \mathbb{N}$. (D'après la Prop. 26, ces termes de preuves sont donc aussi des réalisateurs universels des formules correspondantes.)

La construction $\{e\} \Rightarrow B$ Dans ce qui suit, on étend le langage des formules avec une nouvelle construction notée $\{e\} \Rightarrow B$ (où e est une expression arithmétique et B une formule)

Formules $\qquad A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow B$

qui représente le type des termes produisant un réalisateur de la formule B dès qu'on les applique à l'entier de Krivine associé à l'expression e. Dans le modèle de réalisabilité, la valeur de fausseté de la formule $\{e\} \Rightarrow B$ (dans le cas où cette formule est close) est définie par

$$\|\{e\} \Rightarrow B\| \;=\; \{\bar{n} \cdot \pi \;:\; n = \downarrow e,\; \pi \in \|B\|\}\,.$$

D'après cette définition, il est clair que la formule $\mathsf{Nat}(e) \Rightarrow B$ est un sous-type de la formule $\{e\} \Rightarrow B$ (voir section 4.1.4). Intuitivement, cette relation

[10] J'utilise ici la terminologie d'« entier de Krivine » pour insister sur le fait qu'en réalisabilité classique, il est nécessaire de modifier légèrement la représentation des entiers pour pouvoir bénéficier des opérateurs de mise en mémoire [58, 62]. Cependant, la représentation de Krivine reste β-convertible avec la représentation de Church, et Krivine utilise l'expression « entier de Church » pour désigner ses entiers. Il est à noter que ces problèmes de représentation disparaissent complètement dès qu'on passe à des entiers primitifs, et que dans ce cadre, les opérateurs de mise en mémoire sont réduits à leur plus simple expression.

[11] Guillermo a en effet démontré [47] qu'il n'existe pas de version des opérateurs de mise en mémoire pour les entiers de Church.

de sous-typage exprime que toute fonction capable de traiter un réalisateur classique arbitraire de la formule $\mathsf{Nat}(e)$ est également capable de traiter l'entier de Krivine représentant e (celui-ci constituant un réalisateur particulier de la formule $\mathsf{Nat}(e)$). On a donc

$$\lambda x\,.\,x \;\; \Vdash \;\; \forall x\; \forall X\; ((\mathsf{Nat}(x) \Rightarrow X) \Rightarrow (\{x\} \Rightarrow X))$$

Opérateurs de mise en mémoire Par définition, on appelle un *opérateur de mise en mémoire* toute quasi-preuve réalisant (universellement) l'implication réciproque, c'est-à-dire toute quasi-preuve M telle que

$$M \;\; \Vdash \;\; \forall x\; \forall X\; ((\{x\} \Rightarrow X) \Rightarrow (\mathsf{Nat}(x) \Rightarrow X))$$

Intuitivement, un opérateur de mise en mémoire sert à étendre toute fonction définie sur les seuls entiers de Krivine (qui jouent ici le rôle des valeurs) en une fonction définie sur tous les entiers classiques, c'est-à-dire sur tous les réalisateurs classiques de la formule $\mathsf{Nat}(x)$ (indépendamment de la valeur de x). C'est grâce aux opérateurs de mise en mémoire qu'il est possible d'utiliser des primitives en appel par valeur dans un cadre où l'évaluation est en appel par nom.

D'un point de vue calculatoire, on peut montrer qu'un opérateur de mise en mémoire évalue son deuxième argument (le réalisateur classique de $\mathsf{Nat}(x)$) dans le contexte courant (ici la pile π) avant de passer le résultat (sous la forme d'un entier de Krivine) à son premier argument :

Lemme 13 — *Si M est un opérateur de mise en mémoire, alors pour tout entier $n \in \mathbb{N}$, pour tout réalisateur universel $t \Vdash \mathsf{Nat}(n)$, pour tout terme u et pour toute pile π on a :* $M \star u \cdot t \cdot \pi \;\; \succ^* \;\; u \star \bar{n} \cdot \pi$.

Preuve. Il suffit de considérer le pôle $\bot\!\!\!\bot = \{p \;:\; p \succ^* u \cdot \bar{n} \cdot \pi\}$ et de poser $S = \{\pi\}$. On vérifie alors que $u \Vdash \{n\} \Rightarrow \dot{S}$, d'où $M \star u \cdot t \cdot \pi \in \bot\!\!\!\bot$. □

Il est facile de construire des opérateurs de mise en mémoire, comme par exemple le terme $M \equiv \lambda f n\,.\,n\,f\,(\lambda h x\,.\,h\,(\bar{s}\,x))\,\bar{0}$ [58, 62].

4.2.2 Une méthode d'extraction naïve

Considérons à présent un réalisateur universel d'une formule existentielle numérique close :

$$t_0 \;\; \Vdash \;\; \exists^{\mathsf{N}} x\, A(x) \;\; \equiv \;\; \forall Z\,(\forall^{\mathsf{N}} x\,(\mathsf{Nat}(x) \Rightarrow A(x) \Rightarrow Z) \;\; \Rightarrow \;\; Z)\,.$$

Pour extraire le témoin implicitement contenu dans le réalisateur universel t_0, il suffit d'appliquer ce réalisateur au terme $\lambda x y\,.\,M\,u\,x$ qui passe à la continuation u (ici : un terme clos arbitraire) la première projection du couple t_0 après l'avoir évaluée avec l'opérateur de mise en mémoire M. Cette méthode « marche » effectivement dans la mesure où :

Lemme 14 — *Si $t_0 \Vdash \exists^{\mathsf{N}} x\, A(x)$, alors pour tout terme u et pour toute pile π, il existe un entier $n \in \mathbb{N}$ tel que :* $t_0 \star (\lambda x y\,.\,M\,u\,x) \cdot \pi \succ^* u \star \bar{n} \cdot \pi$.

Preuve. Il suffit de considérer le pôle $\bot\!\!\!\bot \equiv \{p \;:\; \exists n\; (p \succ^* u \star \bar{n} \cdot \pi)\}$ et de poser $S = \{\pi\}$. On vérifie alors que $u \Vdash \forall x\,(\{x\} \Rightarrow \dot{S})$, puis $M\,u \Vdash \forall x\,(\mathsf{Nat}(x) \Rightarrow \dot{S})$ et finalement $\lambda x y\,.\,M\,u\,x \Vdash \forall x\,(\mathsf{Nat}(x) \Rightarrow A(x) \Rightarrow \dot{S})$, d'où il ressort que le processus $t_0 \star (\lambda x y\,.\,M\,u\,x) \cdot \pi$ appartient au pôle $\bot\!\!\!\bot$. □

Malheureusement, cette méthode d'extraction naïve ne nous donne absolument aucune garantie sur l'entier n qu'elle produit. Celui-ci n'a en effet aucune raison de satisfaire la propriété $A(n)$, puisque le réalisateur de la formule $A(n)$ qui l'accompagne (c'est-à-dire la deuxième composante du couple, effacée par la deuxième abstraction de l'éliminateur $\lambda xy\,.\,M\,u\,x$) ne certifie en rien que l'entier n satisfait la propriété désirée. (Voir la discussion au sujet des parapreuves à la fin de la section 4.1.1.)[12] Il n'y a d'ailleurs rien d'étonnant à ce qu'un mécanisme aussi naïf échoue la plupart du temps, dans la mesure où la logique classique permet de prouver — et donc de réaliser — la formule

$$\exists^{\mathsf{N}} x\ ((x = 1 \wedge C) \vee (x = 0 \wedge \neg C))$$

où C désigne n'importe quelle formule close de PA2 (par exemple : la conjecture de Goldbach ou l'hypothèse de Riemann).[13] Dans un cadre aussi général, un mécanisme d'extraction de témoin fiable nous donnerait automatiquement une procédure de décision pour tous les problèmes exprimables par une formule de PA2, laquelle n'existe bien évidemment pas.

L'échec de la méthode d'extraction naïve ne signifie évidemment pas qu'il faille renoncer à extraire un témoin fiable dans des cas particuliers. Il y a en effet un ingrédient que nous n'avons pas encore utilisé, à savoir le réalisateur $u \Vdash A(n)$ qui accompagne le témoin n proposé par le réalisateur t_0. À défaut de constituer un « certificat » de la formule $A(n)$ (ainsi que la logique intuitionniste nous y a habitués), ce réalisateur peut être utilisé comme une « police d'assurance » contre laquelle on pourra toujours se retourner au moment opportun.

Ce qui nous amène à considérer le cas suivant.

4.2.3 La méthode d'extraction dans le cas décidable

On suppose à présent que la formule $A(x)$ (qui ne dépend que de la variable x) vient avec les trois ingrédients suivants :

1. Un réalisateur universel $t_0 \Vdash \exists^{\mathsf{N}} x\, A(x)$ (par exemple : une preuve)
2. Une fonction de *décision* du prédicat $A(x)$, c'est-à-dire un terme $d_A \in \Lambda$ tel que pour tout $n \in \mathbb{N}$ et pour tous $u_1, u_2 \in \Lambda$, $\pi \in \Pi$ on ait

$$d_A \star \bar{n} \cdot u_1 \cdot u_2 \cdot \pi \ \succ^* \ \begin{cases} u_1 \star \pi & \text{si } A(n) \text{ est vrai} \\ u_2 \star \pi & \text{si } A(n) \text{ est faux} \end{cases}$$

   (au sens du modèle standard décrit à la section 4.1.4).
3. Une fonction de *réfutation conditionnelle* du prédicat $A(x)$, c'est-à-dire un terme $r_A \in \Lambda$ satisfaisant la condition que pour tout entier $n \in \mathbb{N}$, si $A(n)$ est faux (dans le modèle standard), alors $r\,\bar{n} \Vdash \neg A(n)$. Intuitivement, ce terme sert à construire la réfutation de $A(n)$ que l'on opposera à la « justification » de $A(n)$ qui accompagne le témoin proposé, dans le cas où ce témoin se révèle être un faux-témoin. (Nous verrons un peu plus loin dans quelles conditions il est possible de construire un tel terme.)

[12] Il existe cependant un cas où n est un témoin correct, qui est celui où le réalisateur t_0 provient d'une preuve intuitionniste de $\exists^{\mathsf{N}} x\, A(x)$. Dans ce cas, on peut évidemment justifier la correction du témoin en construisant un modèle de réalisabilité intuitionniste approprié.

[13] Dans le cas où t_0 est la preuve de $\exists^{\mathsf{N}} x\ ((x = 1 \wedge C) \vee (x = 0 \wedge \neg C))$ par disjonction de cas sur $C \vee \neg C$, la méthode d'extraction naïve retourne systématiquement l'entier 0, reflétant ainsi la stratégie du tiers-exclu (implémenté à l'aide de $\mathfrak{cc}$) qui consiste à parier d'abord sur $\neg A$ avant de se retourner sur A en cas de *backtrack*.

Considérons alors le processus

$$t_0 \star (M\, \lambda xy\,.\, d_A\, x\, (u\, x)\, (r_A\, x\, y)) \cdot \pi$$

où u est un terme quelconque (destiné à récupérer le témoin), π une pile quelconque, et où M est un opérateur de mise en mémoire.

Intuitivement, ce processus extrait le témoin et la justification proposés par le réalisateur t_0, en évaluant le témoin (avec l'opérateur de mise en mémoire) qui est alors passé à la fonction de décision d_A. Dans le cas où la fonction de décision accepte le témoin, celui-ci est passé à la continuation de sortie u. Dans le cas contraire, on applique la fonction de réfutation conditionnelle r_A au faux-témoin puis à la justification qui l'accompagne, pour produire un réalisateur de la formule fausse. En pratique, la parapreuve ainsi construite déclenche un mécanisme de *backtrack* qui conduit tôt ou tard le réalisateur t_0 à proposer un nouveau témoin et une nouvelle justification. Grâce à ce mécanisme, on a ainsi défini une *boucle de rétroaction* qui permet d'examiner non pas un témoin, mais une succession de témoins (tous produits par le réalisateur t_0 en conjonction avec la fonction de réfutation conditionnelle r_A), dans l'espoir que le processus finisse par trouver un témoin correct en un nombre fini d'étapes de réduction.

Bien que la description qui précède ne soit au départ qu'une simple intuition de programmeur, la réalisabilité classique permet de lui donner très facilement un contenu formel qui est le suivant :

Proposition 29 (Convergence du processus) — *Soient t_0 un réalisateur universel de la formule $\exists^{\mathsf{N}} x\, A(x)$, d_A une fonction de décision et r_A une fonction de réfutation conditionnelle du prédicat $A(x)$. Alors pour tous $u \in \Lambda$ et $\pi \in \Pi$, il existe un entier naturel $n \in \mathbb{N}$ tel que*

$$t_0 \star (M\, \lambda xy\,.\, d_A\, x\, (u\, x)\, (r_A\, x\, y)) \cdot \pi \quad \succ^* \quad u \star \bar{n} \cdot \pi\,,$$

et tel que la formule $A(n)$ est vraie.

Preuve. On considère le pôle défini par

$$\perp\!\!\!\perp \;=\; \{p \;:\; \exists n \in \mathbb{N}\ (A(n) \text{ vrai } \wedge\ p \succ^* u \cdot \bar{n} \cdot \pi\} \qquad (u, \pi \text{ fixés})$$

et on pose $S = \{\pi\}$. On montre alors que pour tous $n \in \mathbb{N}$ et $v \Vdash A(n)$ on a $d_A\, \bar{n}\, (u\, \bar{n})\, (r_A\, \bar{n}\, v) \star \pi \in \perp\!\!\!\perp$ (en distinguant les cas suivant que la formule $A(n)$ est vraie ou fausse dans le modèle standard), d'où il ressort que

$$\lambda xy\,.\, d_A\, x\, (u\, x)\, (r_A\, x\, y) \quad \Vdash \quad \forall x\, (\{x\} \Rightarrow A(x) \Rightarrow \dot{S})$$

puis
$$M\, \lambda xy\,.\, d_A\, x\, (u\, x)\, (r_A\, x\, y) \quad \Vdash \quad \forall x\, (\mathsf{Nat}(x) \Rightarrow A(x) \Rightarrow \dot{S})$$

et finalement : $t_0 \star (M\, \lambda xy\,.\, d_A\, x\, (u\, x)\, (r_A\, x\, y)) \cdot \pi \in \perp\!\!\!\perp$. □

Extraction dans le cas Σ_1^0 Le résultat précédent s'applique naturellement au cas où $A(x)$ est de la forme $A(x) \equiv (f(x) = 0)$, où f est un symbole de fonction primitive récursive. Dans ce cas, il est facile de construire une fonction de décision d_A à partir d'une implémentation de la fonction f en λ-calcul. En ce qui concerne la fonction de réfutation conditionnelle, il suffit de prendre la fonction constante $r_A \equiv \lambda\_\, x\,.\, x\,?$ (où ? désigne un terme arbitraire) puisque

le terme $\lambda x\,.\,x\,?$ réfute (universellement) toutes les égalités fausses d'après le lemme 11 p. 86.

En pratique, on peut même *inliner* le code de la fonction r_A dans le processus de la Prop. 29, ce qui nous permet d'aboutir au processus

$$t_0 \star (M\ \lambda xy\,.\,d_A\,x\,(u\,x)\,(y\,?))) \cdot \pi$$

dont la convergence s'établit de la même façon qu'à la Prop. 29.

4.2.4 La méthode du kamikaze

Il est important de noter que dans la méthode d'extraction qui précède, le composant critique n'est pas tant la fonction de réfutation conditionnelle r_A que la fonction de décision d_A, qui impose que le prédicat $A(x)$ soit décidable. La fonction de réfutation r_A n'impose rien de tel, et son existence peut même être démontrée pour une très large classe de formules, et en particulier pour toutes les formules de l'arithmétique du premier ordre :

Proposition 30 (Existence d'une réfutation conditionnelle) — *Si $A(x)$ est une formule de l'arithmétique du premier ordre (sans paramètre) ne dépendant que de la variable x, alors il existe une quasi-preuve r_A telle que pour tout $n \in \mathbb{N}$, si $A(n)$ est vrai dans le modèle standard, alors $r_A\,\bar{n} \Vdash A^{\mathsf{Nat}}(n)$.*

Preuve. L'existence d'une telle quasi-preuve est une conséquence immédiate du Théorème 21 (p. 14) dans [62]. □

Dans le cas où on ne dispose pas d'une fonction de décision, on peut néanmoins faire tourner indéfiniment la boucle de rétroaction décrite ci-dessus, en utilisant la fonction de réfutation conditionnelle r_A pour répudier systématiquement les témoins produits (indépendamment de leur correction).

Pour conserver la trace de tous ces témoins, il nous faut d'abord introduire un dispositif d'affichage (ou de stockage), qu'on modélisera ici sous la forme d'une instruction print avec la règle d'évaluation

$$\mathsf{print} \star \bar{n} \cdot t \cdot \pi \ \succ\ t \star \pi \qquad (n \in \mathbb{N})$$

On suppose également (de manière informelle) qu'au cours de cette étape d'évaluation, l'entier n est affiché à l'écran ou stocké dans un fichier quelconque.[14]

Ceci nous permet de considérer le processus

$$t_0 \star (M\ \lambda xy\,.\,\mathsf{print}\,x\,(r_A\,x\,y)) \cdot \pi\,,$$

où π est une pile quelconque. Intuitivement, ce processus examine un à un tous les témoins produits par le réalisateur universel t_0, les évalue, les affiche puis les répudie en invoquant la fonction de réfutation conditionnelle r_A. Pour chaque témoin examiné, il n'y a que deux cas possibles :

- Soit le témoin examiné est incorrect. Dans ce cas, la fonction de réfutation conditionnelle est invoquée dans les conditions de sa spécification, ce qui permet de *backtracker* pour passer à la production du témoin suivant.

[14] Il est important de distinguer la spécification formelle (i.e. la règle d'évaluation) de l'instruction print — qui permet de raisonner sur cette instruction — de sa spécification informelle — qui lui donne son intérêt pratique.

– Soit le témoin examiné est correct. Dans ce cas, la fonction de réfutation est utilisée en dehors des conditions qui la spécifient (le terme $r_A\,x\,y$ est « mal typé »), ce qui signifie que tout peut arriver. Le processus peut entrer en boucle infinie (en continuant à afficher éventuellement d'autres entiers) tout comme il peut simplement « planter »[15].

Évidemment, tout l'intérêt du processus en question réside dans le fait qu'il ne peut pas se comporter de manière incorrecte avant d'avoir affiché un témoin répondant à la question ! (D'où le nom de la méthode.) Il est même facile de démontrer que dans tous les cas, le processus $t_0 \star (M\,\lambda xy\,.\,\mathsf{print}\,x\,(r_A\,x\,y)) \cdot \pi$ finit par afficher un témoin correct (en temps fini) :

Proposition 31 — *Si t_0 est un réalisateur universel de $\exists^{\mathsf{N}} x\,A(x)$ et si r_A est une fonction de réfutation conditionnelle du prédicat $A(x)$, alors pour toute pile $\pi \in \Pi$, il existe un entier naturel $n \in \mathbb{N}$ et un terme $u \in \Lambda$ tels que $A(n)$ est vrai (dans le modèle standard) et*

$$t_0 \star (M\,\lambda xy\,.\,\mathsf{print}\,x\,(r_A\,x\,y)) \cdot \pi \quad \succ^* \quad \mathsf{print} \star \bar{n} \cdot u \cdot \pi\,.$$

Preuve. On considère le pôle

$$\perp\!\!\!\perp \;=\; \{p \;:\; \exists n\;\exists u\;(A(n) \text{ vrai} \;\wedge\; p \succ^* \mathsf{print} \star \bar{n} \cdot u \cdot \pi)\} \qquad (\pi \text{ fixé})$$

et on pose $S = \{\pi\}$. On montre alors que $\mathsf{print}\,\bar{n}\,(r_A\,\bar{n}\,v) \Vdash S$ pour tous $n \in \mathbb{N}$ et $v \Vdash A(n)$ (en distinguant les cas suivant que la formule $A(n)$ est vraie ou fausse), d'où il ressort que

$$\lambda xy\,.\,\mathsf{print}\,x\,(r_A\,x\,y) \quad \Vdash \quad \forall x\,(\{x\} \Rightarrow A(x) \Rightarrow \dot{S})$$

puis
$$M\,\lambda xy\,.\,\mathsf{print}\,x\,(r_A\,x\,y) \quad \Vdash \quad \forall x\,(\mathsf{Nat}(x) \Rightarrow A(x) \Rightarrow \dot{S})$$

et finalement : $t_0 \star (M\,\lambda xy\,.\,\mathsf{print}\,x\,(r_A\,x\,y)) \cdot \pi \in \perp\!\!\!\perp$. □

On notera que la preuve ci-dessus repose de manière essentielle sur l'utilisation d'un pôle qui n'est *pas* clos par évaluation. (En effet, tous les processus de la forme $\mathsf{print} \star \bar{n} \cdot u \cdot \pi$ continuent à s'évaluer.) Dès qu'on a dépassé le point où un témoin correct a été affiché, le modèle de réalisabilité n'assure donc plus aucun invariant de correction sur le processus en cours d'évaluation.

4.2.5 Utilisation des entiers primitifs

Le lecteur aura sans doute remarqué que les différentes méthodes d'extraction que nous venons de décrire reposent sur une représentation des entiers naturels qui est particulièrement inefficace. Ici, le problème ne vient pas tant du choix de la représentation unaire des entiers (qu'on pourrait facilement remplacer par une représentation binaire) que du fait de *définir* les entiers en utilisant les seules constructions du langage de base (ici : l'abstraction et l'application). Dans les langages de programmation réalistes, on choisit plutôt d'introduire les entiers comme des constantes primitives (c'est-à-dire : comme des boîtes noires

[15] Dans le langage λ_c, un processus « plante » lorsque l'instruction en position d'évaluation se retrouve face à une pile manifestement « mal typée », i.e. qui ne permet pas de continuer l'évaluation. Par exemple, une abstraction (ou l'une des constantes cc, k_π) face à une pile vide, ou encore l'instruction print face à une pile dont le premier argument n'est pas un entier.

sans structure particulière), et c'est à l'implémenteur du compilateur de choisir la représentation que ces entiers auront en machine, en fonction des possibilités offertes par le système considéré et des librairies qui sont à sa disposition.

Une des richesses de la réalisabilité classique est de permettre de faire exactement la même chose avec le langage λ_c. Plutôt que de définir les entiers d'une manière ou d'une autre (ce qui nous obligerait à privilégier une représentation particulière), on enrichit le langage λ_c avec les constantes suivantes :

- Pour chaque entier $n \in \mathbb{N}$, une constante $\hat{n}$ représentant l'entier n sous forme d'une donnée. (Le choix de la représentation étant laissé à la discrétion de l'implémenteur.) Ici, la constante $\hat{n}$ représente bien une *donnée* et non une *instruction* dans la mesure où on ne lui attache aucune règle de réduction spécifique, de la forme $\hat{n} \star \pi \succ \cdots$. À l'instar des langages de programmation habituels (où les entiers ne sont jamais *exécutés*), la constante $\hat{n}$ n'a ici aucun sens en position d'évaluation, et tout processus de la forme $\hat{n} \star \pi$ doit être considéré comme une erreur.[16]
- Pour chaque fonction arithmétique usuelle f d'arité k (successeur, prédécesseur, addition, multiplication, etc.) une instruction $\check{f}$ munie de la règle d'évaluation

$$\check{f} \star \hat{n}_1 \cdots \hat{n}_k \cdot u \cdot \pi \ \succ \ u \star \hat{m} \cdot \pi, \quad \text{avec } m = f(n_1, \cdots, n_k)$$

  pour tout $(m_1, \ldots, m_k) \in \mathrm{dom}(f)$ (le comportement de l'instruction $\check{f}$ n'étant pas spécifié quand la pile n'a pas la forme désirée). On notera que cette instruction attend $k+1$ arguments sur la pile, le dernier argument étant utilisé comme une continuation de retour.[17]
- Pour chaque opération de comparaison usuelle g d'arité k (test de nullité, test d'égalité, plus petit que, plus grand que, etc.) une instruction $\check{g}$ munie de la règle d'évaluation

$$\check{g} \star \hat{n}_1 \cdots \hat{n}_k \cdot u \cdot v \cdot \pi \ \succ \ \begin{cases} u \star \pi & \text{si } g(n_1, \ldots, n_k) \text{ vrai} \\ v \star \pi & \text{si } g(n_1, \ldots, n_k) \text{ faux} \end{cases}$$

  pour tout $(m_1, \ldots, m_k) \in \mathrm{dom}(g)$.

Interaction avec le modèle de réalisabilité Pour spécifier le comportement calculatoire des nouvelles constantes et instructions, on étend à nouveau le langage des formules avec une nouvelle construction

Formules $\qquad A, B \quad ::= \quad \cdots \quad | \quad [e] \Rightarrow B$

[16] Ce qui laisse toute latitude à la personne qui implémente un interpréteur ou un compilateur pour le langage λ_c de représenter ces constantes comme elle l'entend, sans se soucier de ce qui se produit lorsqu'elles arrivent en position d'évaluation. En pratique, l'utilisateur devra donc s'attendre à observer un comportement de la forme : $\hat{n} \star \pi \succ$ `segmentation fault`.

[17] Dans les langages en appel par valeur, cet argument supplémentaire est implicite dans le langage et n'apparaît de manière explicite que lors de la compilation : il s'agit du bloc de retour (posé généralement à la suite des arguments sur la pile), qui sert à indiquer ce que le programme doit faire au retour de l'appel de fonction. On voit donc qu'ici, le langage λ_c est bien un langage de plus bas niveau que les langages fonctionnels en appel par valeur, dans la mesure où il nécessite d'expliciter cet argument supplémentaire. Ce serait donc une erreur de vouloir compiler le langage λ_c vers un langage tel que Caml au moment de l'extraction, dans la mesure où Caml est un langage de plus haut niveau. S'il est nécessaire de traduire un langage vers l'autre, il est donc bien plus judicieux de procéder dans le sens inverse, c'est-à-dire de compiler Caml vers λ_c à travers une CPS chargée d'expliciter les blocs de retour.

qui représente cette fois-ci le type des termes produisant un réalisateur de la formule B dès qu'on les applique à l'entier-donnée associé à l'expression e. Dans le modèle de réalisabilité, la valeur de fausseté de la formule $[e] \Rightarrow B$ (dans le cas où cette formule est close) est définie par

$$\|[e] \Rightarrow B\| \;=\; \{\hat{n} \cdot \pi \;:\; n = {\downarrow}e,\; \pi \in \|B\|\}\,.$$

Bien entendu, cette construction est très similaire à la construction $\{x\} \Rightarrow B$ introduite à la section 4.2.1, la différence résidant uniquement dans la façon dont les valeurs entières sont représentées dans la pile. Cependant, la formule $\mathsf{Nat}(e) \Rightarrow B$ ne constitue pas un sous-type de la formule $[e] \Rightarrow B$, puisque l'entier-donnée correspondant à la valeur de e n'est pas un réalisateur de la formule classique $\mathsf{Nat}(e)$ (ce n'est pas un programme).

Pour remédier à ce problème, on introduit un nouveau prédicat de relativisation pour les entiers, noté $\mathsf{Nat}'(e)$ et défini par

$$\mathsf{Nat}'(e) \;\equiv\; \forall X\,((\![e]\!] \Rightarrow X) \Rightarrow X)$$

Intuitivement, la formule $\mathsf{Nat}'(e)$ désigne le type des *entiers-programmes*, ou encore des *entiers paresseux*. (On peut également voir la formule $\mathsf{Nat}'(e)$ comme le type des opérateurs de mise en mémoire spécialisés à l'entier e.)

Tout entier-donnée $\hat{n}$ peut être représenté sous la forme d'un entier-programme, à savoir le terme $\lambda x\,.\,x\,\hat{n} \Vdash \mathsf{Nat}'(n)$. Réciproquement, tout entier-programme a pour fonction de produire l'entier-donnée correspondant dès qu'on l'applique à une continuation de retour (c'est à ce moment-là que seront effectués tous les calculs nécessaires à la production de cet entier-donnée) :

Lemme 15 (Spécification de la formule $\mathsf{Nat}'(n)$) — *Si $t \Vdash \mathsf{Nat}'(n)$, alors pour tous $u \in \Lambda$ et $\pi \in \Pi$ on a*

$$t \star u \cdot \pi \;\succ^*\; u \star \hat{n} \cdot \pi\,.$$

Par ailleurs, le prédicat $\mathsf{Nat}'(x)$ permet de « typer » chaque instruction $\check{f}$ associée à un symbole de fonction primitive récursive de même nom :

$$\check{f} \;\Vdash\; \forall x_1 \cdots \forall x_k\;([x_1] \Rightarrow \cdots \Rightarrow [x_k] \Rightarrow \mathsf{Nat}'(f(x_1, \ldots, x_k))$$

Ce qui nous donne à peu de frais un réalisateur de la formule exprimant la totalité de la fonction f avec le prédicat $\mathsf{Nat}'(x)$:

$$\begin{aligned}&\lambda x_1 \cdots x_k z\,.\,x_1\,\lambda y_1\,.\,\ldots x_k\,\lambda y_k\,.\,\check{f}\,y_1 \cdots y_k\,z\\ &\quad\Vdash\; \forall x_1 \cdots \forall x_k\;(\mathsf{Nat}'(x_1) \Rightarrow \cdots \Rightarrow \mathsf{Nat}'(x_k) \Rightarrow \mathsf{Nat}'(f(x_1, \ldots, x_k)))\,.\end{aligned}$$

Bien évidemment, il n'y a aucun mystère dans le fait que le prédicat de relativisation $\mathsf{Nat}'(e)$ se comporte essentiellement de la même manière que le prédicat $\mathsf{Nat}(e)$ codé au second ordre. Les deux formules $\mathsf{Nat}'(e)$ et $\mathsf{Nat}(e)$ sont en effet équivalentes au sens de la réalisabilité, en ce sens qu'il existe des quasi-preuves réalisant (universellement) les deux implications permettant de passer d'une représentation à l'autre :

Proposition 32 — *Soient* succ *et* pred *les instructions associées aux fonctions successeur et prédécesseur,* null *l'instruction associée au test de nullité, et Y un combinateur de point fixe. On a alors :*

$$\begin{aligned}\lambda n\,.\,n\,(\lambda x\,.\,x\,\hat{0})\,(\lambda y\,.\,y\,\mathsf{succ}) \quad &\Vdash \quad \forall x\,(\mathsf{Nat}(x) \Rightarrow \mathsf{Nat}'(x))\\ Y\,(\lambda r x\,.\,x\,\lambda y\,.\,\mathsf{null}\,y\,\bar{0}\,(\bar{s}\,(r\,(\mathsf{pred}\,y)))) \quad &\Vdash \quad \forall x\,(\mathsf{Nat}'(x) \Rightarrow \mathsf{Nat}(x))\,.\end{aligned}$$

À partir du moment où il est clair que les prédicats de relativisation $\mathsf{Nat}(x)$ et $\mathsf{Nat}'(x)$ sont logiquement équivalents, on aura tout intérêt à n'utiliser que le second dans les formules afin de bénéficier de toute l'efficacité des entiers primitifs dans les programmes extraits. De fait, les diverses méthodes d'extraction de témoin évoquées dans les sections précédentes (4.2.2, 4.2.3 et 4.2.4) s'adaptent sans peine à cette nouvelle présentation des entiers, et sont même légèrement simplifiées par la disparition des opérateurs de mise en mémoire.

4.2.6 Le théorème d'effectivité expérimentale

Bien que la réalisabilité classique ait été introduite à l'origine pour analyser le comportement calculatoire des preuves mathématiques (classiques), le cadre qu'elle définit est en réalité suffisamment riche pour décrire également l'interaction entre le raisonnement mathématique pur et les hypothèses « extra-mathématiques » sur lesquelles reposent les sciences expérimentales.[18]

Le problème du modus tollens expérimental Pour cela, nous allons nous intéresser à un problème soulevé par le philosophe Karl Popper [88] concernant l'utilisation du principe du *modus tollens*

$$\frac{A \Rightarrow B \qquad \neg B}{\neg A}$$

dans un cadre expérimental.

L'épistémologie de Popper est en effet basée sur l'idée que les théories scientifiques (qui sont au départ de simples descriptions linguistiques) communiquent avec le monde réel par l'intermédiaire d'une classe particulière d'énoncés, à savoir les *énoncés empiriques*, c'est-à-dire les énoncés universels dont toutes les instances sont testables au moyen d'un dispositif expérimental approprié, au

[18] On touche ici évidemment à la question de la « déraisonnable efficacité des mathématiques » dans les sciences expérimentales, pour reprendre l'expression popularisée par le physicien Eugene Wigner dans son célèbre essai [105]. Cette question recouvre à mon avis (au moins) trois problèmes distincts. Le premier problème est celui de l'*expressivité* du langage mathématique pour formuler les différentes théories scientifiques. Le développement considérable des sciences au cours des deux derniers siècles nous a montré que le langage mathématique est très adapté à la formulation des problèmes des sciences « dures ». C'est beaucoup moins vrai dans les sciences « molles », pour lesquelles les constructions de la logique traditionnelle sont trop grossières pour exprimer des énoncés tels que « A malgré B », « A au lieu de B » ou encore « le comportement X est typique au sein de la population Y ». Le deuxième problème (purement subjectif) est l'aptitude du raisonnement mathématique à s'intégrer dans les structures de pensée humaines (qui tiennent sans doute autant à notre nature qu'à notre histoire), c'est-à-dire : dans quelle mesure le raisonnement mathématique est compatible avec nos « habitudes intellectuelles » et en quoi il est susceptible de stimuler notre imagination en faisant appel à des intuitions, des images, des représentations mentales en dehors des mathématiques. On notera que cette aptitude à stimuler l'intellect n'est pas l'apanage des mathématiques : c'est un trait qu'on retrouve aussi dans la mythologie, dans les religions, en philosophie ou même en psychanalyse. Et ce n'est sans doute pas un hasard si les mathématiques empruntent si souvent leur vocabulaire au vocabulaire philosophico-religieux : « canonique », « transcendant », « fidèle », « universel » ($\kappa\alpha\theta o\lambda\iota\kappa\acute{o}\varsigma$ en grec) et plus récemment : « catégorie ». Enfin, le troisième problème, que je me propose de traiter ici, concerne l'*objectivité* du raisonnement mathématique, c'est-à-dire son aptitude à interagir avec le monde des *objets* non mathématiques. C'est pourquoi je me place ici dans le cadre épistémologique développé par Popper, qui est précisément centré sur la question de l'objectivité de la démarche scientifique. (Ce qui n'exclut évidemment pas d'autres approches possibles.)

moins potentiellement. (On parlera aussi d'*énoncé universel falsifiable.*) La notion d'« énoncé empirique » au sens de Popper est donc très proche de la notion de formule Π^0_1 en logique, mais avec une différence essentielle, qui est que la testabilité doit s'entendre au sens expérimental et non au sens mathématique du terme. Ainsi on dira qu'un prédicat $p(x_1, \ldots, x_n)$ est testable si pour chaque jeu de paramètres $x_1, \ldots, x_n$ (dans un ensemble de valeurs données) il est possible de déterminer la vérité de $p(x_1, \ldots, x_n)$ à l'aide d'une expérience particulière[19] (au moins de manière potentielle). La notion de testabilité englobe la notion de *décidabilité* mathématique (basée sur la théorie des fonctions récursives) — qu'on peut voir comme la notion de testabilité interne aux mathématiques — mais ne s'y réduit évidemment pas[20].

Popper remarque qu'une théorie empirique A (c'est-à-dire : une conjonction d'énoncés universels falsifiables[21]) est rarement falsifiée de manière directe (i.e. par une expérience falsifiant une instance particulière d'un énoncé donné de la théorie A), mais qu'elle est le plus souvent falsifiée de manière indirecte à travers la falsification expérimentale d'un énoncé universel falsifiable B qui est une conséquence logique de A (c'est-à-dire : un énoncé pour lequel on dispose d'une preuve de $A \Rightarrow B$). Tout le problème réside dans le fait que la théorie A est alors falsifiée globalement par voie de conséquence logique (et non ponctuellement par une expérience bien choisie) en combinant deux objets de nature très différente, à savoir une falsification expérimentale de B (donnée par un jeu de paramètres et une mesure) et une preuve mathématique de $A \Rightarrow B$ (susceptible

[19] Un contresens courant consiste à croire que Popper n'accorde un statut de scientificité qu'aux seuls énoncés empiriques (universels falsifiables). Ce qui n'est pas le cas dans la mesure où Popper saisit très bien l'importance d'utiliser d'autres formes d'énoncés dans le discours scientifique, surtout dans l'expression des principes de la théorie. (Typiquement : les notions de force ou de potentiel, qu'on ne mesure pas directement, mais à travers leurs effets suivant d'autres principes édictés par la théorie.) De fait, la scientificité d'une théorie ne se mesure pas à l'échelle d'un énoncé, mais à l'échelle d'une théorie toute entière. Une théorie est d'autant plus scientifique qu'elle contient, explicitement (dans l'énoncé des principes) ou implicitement (par voie de conséquence logique) une plus large classe d'énoncés empiriques, c'est-à-dire une plus grande emprise à l'expérience pour falsifier la théorie. Si on s'en tient à ce principe (et contrairement à ce que semble penser Popper), les mathématiques constituent bel et bien une théorie scientifique, dans la mesure où elles produisent une large classe de théorèmes testables (à savoir : les théorèmes Π^0_1), qui jusqu'à présent ont tous été corroborés par l'expérience — c'est-à-dire par le calcul. (Rappelons que la falsification expérimentale d'un seul théorème Π^0_1 entraînerait *ipso facto* l'incohérence des mathématiques.)

[20] À moins de souscrire à la thèse selon laquelle toute fonction calculable par le monde physique est également calculable par une machine de Turing (une thèse connue sous le nom de « thèse de Church physique » [30]), ce que je me garderai bien de faire ici ! L'intérêt de la démarche que je présente ici réside précisément dans le fait qu'elle permet de combiner le calcul au sens mathématique (ici : avec des λ-termes) avec le calcul au sens physique, c'est-à-dire avec des primitives données sous la forme de « boîtes noires », qui peuvent d'ailleurs correspondre à des fonctions récursives dans certains cas. (De fait, le travail que je présente ne dépend ni de la thèse de Church physique, ni de sa négation.) Tout ceci pour dire qu'il n'est pas nécessaire de remplacer les phénomènes physiques par des machines de Turing équivalentes pour pouvoir calculer avec. À l'inverse, il est intéressant de noter que même les défenseurs les plus acharnés de la thèse de Church physique sont en général bien incapables de fournir le code de la machine de Turing calculant la température moyenne à Paris (à 1 degré près) en fonction de la date. La croyance en la thèse de Church physique (qui n'est à mon avis qu'une version modernisée de la cosmologie de Ptolémée) repose fondamentalement sur une confusion entre la notion de *calcul* et la notion de *fonction récursive*, une confusion qui n'a de sens que tant qu'on reste dans le monde des seuls objets mathématiques.

[21] Popper considère également la présence d'énoncés singuliers (testables) dans la théorie A, mais il est clair que du point de vue de la logique formelle, ceux-ci constituent un cas particulier des énoncés universels falsifiables (i.e. ceux dont le degré d'universalité est réduit à zéro).

d'être formalisée dans un système quelconque). Ce qui soulève naturellement la question de déterminer le ou les énoncés mis en cause dans la théorie A — en commençant par écarter les énoncés dont B ne dépend pas. D'après Popper :

> Nous ne pouvons donc pas savoir d'emblée lequel parmi les divers énoncés restants du sous-système A' (dont B n'est pas indépendant) est responsable de la fausseté de B ; nous ne pouvons pas savoir lequel nous devons modifier, lequel nous devons retenir. (Je ne parle pas ici des énoncés interchangeables.) C'est souvent le seul instinct scientifique du chercheur (influencé naturellement par les résultats de tests répétés) qui lui fait deviner quels énoncés de A' doivent être considérés comme inoffensifs et quels autres nécessitent une modification. (...) [88, chap. 3, section 18, p. 74 note 2]

Nous allons à présent montrer que la réalisabilité classique offre une solution simple et élégante à ce problème en donnant un algorithme qui, à partir d'un jeu de paramètres falsifiant B et d'une preuve formelle de $A \Rightarrow B$, effectue une série d'expériences sur A conduisant en temps fini à la falsification d'une instance particulière d'un énoncé de A :

$$\frac{\vdash A \Rightarrow B \qquad \not\models B}{\not\models A} \qquad \text{(Modus tollens expérimental)}$$

(en désignant ici par $\not\models E$ la falsification expérimentale de l'énoncé E).

Le principe d'effectivité expérimentale Pour cela, on se ramène à un problème similaire (en fait : équivalent) qui consiste à déterminer à partir d'une preuve d'incohérence de la théorie A (où A est comme d'habitude une conjonction d'énoncés universels falsifiables) une série d'expériences sur A conduisant en temps fini à la falsification d'une instance particulière d'un énoncé de A :

$$\frac{A \vdash \bot}{\not\models A} \qquad \text{(Effectivité expérimentale)}$$

Toute solution au problème de l'effectivité expérimentale induit immédiatement une solution au problème du modus tollens expérimental : il suffit pour cela de transformer le problème

$$\frac{\vdash A \Rightarrow B \qquad \not\models B}{\not\models A} \qquad \text{en le problème} \qquad \frac{A \wedge B_0 \vdash \bot}{\not\models (A \wedge \neg B_0)}$$

où B_0 désigne l'instance particulière de B qui a été falsifiée expérimentalement, en déduisant la preuve de $A \wedge \neg B_0 \vdash \bot$ des preuves de $\vdash A \Rightarrow B$ (qui nous est donnée par hypothèse) et $\vdash B \Rightarrow B_0$ (élimination du/des $\forall$).

Formalisation du problème On enrichit à présent le langage de la logique du second ordre avec un ensemble de *symboles de prédicats expérimentaux*, notés p, q, r, etc. On suppose que chacun de ces symboles de prédicat p (d'arité k) est accompagné d'une fonction de vérité

$$\mathrm{Val}(p) \ : \ \mathbb{N}^k \to \{0;1\}$$

qui modélise la fonction physique correspondante. Mathématiquement, on ne suppose rien d'autre sur cette fonction que sa totalité. (En particulier, on ne

suppose pas qu'elle est récursive.) Expérimentalement, on suppose que cette fonction est attachée à un dispositif susceptible de donner une réponse de type « oui/non » à chaque jeu de paramètres dans $\mathbb{N}^k$. (On notera qu'ici, les entrées comme la sortie sont de nature discrète.) Bien entendu, il n'est pas raisonnable d'exiger que l'« expérience » attachée à la formule atomique $p(n_1, \ldots, n_k)$ soit toujours réalisable en pratique[22], et nous verrons un peu plus loin comment contourner ce problème.

On définit ensuite trois sous-classes de formules (dans le langage de la logique du second ordre ainsi étendu) : les *formules élémentaires*, les *formules testables* et les *formules universelles falsifiables*. Ces trois sous-catégories syntaxiques sont données par la BNF suivante :

F. élémentaires	E	::=	$e_1 = e_2 \quad \mid \quad p(e_1, \ldots, e_k)$
F. testables	T	::=	$\bot \quad \mid \quad E \quad \mid \quad E \Rightarrow T \quad \mid \quad \neg E \Rightarrow T$
F. universelles falsifiables	U	::=	$\forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k T(x_1, \ldots, x_k)$

(en ne considérant dans la dernière catégorie que des formules closes).

On notera que les formules élémentaires sont soit des formules atomiques construites à partir des symboles de prédicats expérimentaux, soit des égalités entre expressions arithmétiques (égalités dont les instances sont facilement testables). Les formules testables sont quant à elles simplement des clauses de formules élémentaires. Enfin, les formules universelles falsifiables sont les clôtures universelles (relativisées aux entiers) des formules testables, les formules élémentaires ou testables closes constituant un cas particulier (correspondant aux énoncés singuliers chez Popper). On peut remarquer également que toute formule sans quantificateurs construite à partir des formules élémentaires est logiquement équivalente à une conjonction finie de formules testables.

À chaque formule élémentaire E on associe une valeur de vérité notée $\mathrm{Val}(E)$ et définie par :

$$\begin{aligned} \mathrm{Val}(p(e_1, \ldots, e_k)) &= \mathrm{Val}(p)(\downarrow e_1, \ldots, \downarrow e_k) \\ \mathrm{Val}(e_1 = e_2) &= \begin{cases} 1 & \text{si } \downarrow e_1 = \downarrow e_2 \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

On notera que la valeur de vérité d'une formule élémentaire close est expérimentalement calculable : il suffit d'effectuer l'expérience ou le test d'égalité correspondant sur les paramètres obtenus en évaluant les expressions arithmétiques figurant dans la formule. Cette notion est ensuite étendue à toutes les formules testables closes (notation : $\mathrm{Val}(T)$) en suivant la table de vérité de l'implication et en associant à la formule $\bot$ l'expérience qui échoue toujours. (Là encore, la valeur de vérité d'une formule testable close est expérimentalement calculable.)

Enfin, on se donne une *théorie expérimentale* constituée d'une suite finie de formules universelles falsifiables $U_1, \ldots, U_\ell$. Dans ce qui suit, on ne supposera pas que ces formules sont vraies ou fausses *a priori*.

Les instructions test_i et stop On enrichit le langage des réalisateurs avec une instruction stop sans règle d'évaluation, qui sert à arrêter le calcul.

[22] Il serait en effet peu réaliste d'espérer pouvoir mesurer expérimentalement la température moyenne au centre du Soleil au cours de l'année 2800 avant J. C.

Enfin, pour chaque formule $U_i \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k T(x_1, \ldots, x_k)$ $(1 \leq i \leq \ell)$ de la théorie expérimentale dont le corps est de la forme

$$T(\vec{x}) \quad \equiv \quad L_1(\vec{x}) \Rightarrow \cdots \Rightarrow L_n(\vec{x}) \Rightarrow R(\vec{x})$$

(où $L_i(\vec{x})$ est soit une formule élémentaire soit la négation d'une formule élémentaire, et où $R(\vec{x})$ est soit la formule fausse, soit une formule élémentaire) on introduit une instruction test_i munie des règles d'évaluation suivantes pour tous $\nu_1, \ldots, \nu_k \in \mathbb{N}$, $\pi \in \Pi$ et pour tout $j \in [1..n]$:

$$\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \quad \succ \quad \begin{cases} \lambda z_1 \cdots z_n \,.\, z_j \, \mathbf{I} \star \pi & \text{si } \mathrm{Val}(L_j(\vec{\nu})) = 0 \\ \lambda z_1 \cdots z_n \,.\, \mathbf{I} \star \pi & \text{si } \mathrm{Val}(R(\vec{\nu})) = 1 \\ \mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi & \text{si } \mathrm{Val}(T(\vec{\nu})) = 0 \end{cases}$$

Intuitivement, la première règle traite le cas où l'une des prémisses $L_j(\vec{\nu})$ de l'implication $T(\vec{\nu}) \equiv L_1(\vec{\nu}) \Rightarrow \cdots \Rightarrow L_n(\vec{\nu}) \Rightarrow R(\vec{\nu})$ est fausse, la deuxième règle traite le cas où la conclusion $R(\vec{\nu})$ est vraie, tandis que la troisième règle traite le cas où l'implication $T(\vec{\nu})$ est fausse. (La signification des membres droits deviendra claire quand nous aurons donné l'interprétation des symboles de prédicats expérimentaux dans le modèle de réalisabilité.)

On notera que telles que nous venons de les écrire, ces règles ne sont pas déterministes : les conditions d'application des deux premières règles ne s'excluent pas, et la première règle peut même s'appliquer pour plusieurs valeurs de $j \in [1..n]$. (Cependant, les conditions qui permettent d'appliquer chacune des règles sont toutes expérimentalement testables.) Pour conserver le déterminisme de l'évaluation, on fixera donc un ordre d'application des règles, par exemple en donnant priorité à la première règle sur la deuxième, et en considérant (dans la première règle) le plus petit indice j tel que $\mathrm{Val}(L_j(\vec{\nu})) = 0$.

Le modèle de réalisabilité Dans le modèle de réalisabilité on interprète chaque prédicat expérimental p d'arité k exactement de la même manière qu'une égalité (section 4.1.6) en posant :

$$\|p(e_1, \ldots, e_k)\| \;=\; \begin{cases} \{t \cdot \pi \;:\; (t \star \pi) \in \perp\!\!\!\perp\} & \text{si } \mathrm{Val}(p)(\downarrow e_1, \ldots, \downarrow e_k) = 1 \\ \Lambda \cdot \Pi & \text{sinon} \end{cases}$$

(La valeur de fausseté d'une formule atomique dépend donc du pôle $\perp\!\!\!\perp$.)

On remarque alors que pour chaque formule de la théorie expérimentale $U_i \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k T(x_1, \ldots, x_k)$ $(1 \leq i \leq \ell)$, pour chaque jeu de paramètres $\vec{\nu} = \nu_1, \ldots, \nu_k$ tel que $\mathrm{Val}(F(\vec{\nu})) = 1$ et pour chaque pile $\pi \in \Pi$, le processus $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ s'évalue sur un processus de la forme $t \star \pi$, où t est une quasi-preuve qui est un réalisateur universel de la formule $F(\vec{\nu})$. (C'est précisément la raison du choix des règles d'évaluation de l'instruction test_i.)

D'où il ressort que :

Fait 1 — *Pour tout jeu de paramètres $\vec{\nu} = \nu_1, \ldots, \nu_k$ tel que* $\mathrm{Val}(F(\vec{\nu})) = 1$, *le terme $\mathsf{test}_i \, \bar{\nu}_1 \cdots \bar{\nu}_k$ est un réalisateur universel de la formule $F(\vec{\nu})$.*

Bien entendu, ce résultat est faux dans le cas complémentaire (i.e. lorsque $\mathrm{Val}(F(\vec{\nu})) = 0$), puisqu'une formule fausse ne peut pas admettre de réalisateur

universel. Pour « forcer » ce résultat dans tous les cas, on se restreint à un pôle particulier, à savoir le pôle $\perp\!\!\!\perp_0$ constitué de tous les processus s'évaluant (en zéro, une ou plusieurs étapes) sur une falsification de la théorie expérimentale, c'est-à-dire sur un processus de la forme

$$\mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \,,$$

où $i \in [1..\ell]$ est l'indice d'une formule $U_i \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k T(x_1, \ldots, x_k)$ de la théorie expérimentale, et où $\vec{\nu} = \nu_1, \ldots, \nu_k$ est un jeu de paramètres falsifiant cette formule. On vérifie alors immédiatement que :

Fait 2 — *Dans le modèle de réalisabilité induit par le pôle $\perp\!\!\!\perp_0$, on a pour chaque indice $i \in [1..\ell]$ associé à une formule $U_i \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k T(x_1, \ldots, x_k)$:*

$$\mathsf{test}_i \;\Vdash\; \forall x_1 \cdots \forall x_k \; (\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow T(x_1, \ldots, x_k)) \,.$$

En posant $\mathsf{test}'_i \equiv M_k \, U_i$, où M_k est un opérateur de mise en mémoire d'arité k (correspondant au nombre de paramètres de la formule U_i), il vient :

Fait 3 — *Dans le modèle de réalisabilité induit par le pôle $\perp\!\!\!\perp_0$, pour chaque indice $i \in [1..\ell]$ on a :* $\mathsf{test}'_i \Vdash U_i$.

D'où l'on déduit immédiatement le théorème suivant :

Théorème 7 (Effectivité expérimentale) — *Si t est un réalisateur universel de la formule $U_1 \Rightarrow \cdots \Rightarrow U_\ell \Rightarrow \bot$ (éventuellement issu d'une preuve dans PA2), alors pour toute pile $\pi \in \Pi$, il existe un indice $i \in [1..\ell]$, un jeu de paramètres $\vec{\nu} = \nu_1, \ldots, \nu_k$ et une pile π' tels que*

$$t \, \mathsf{test}'_1 \cdots \mathsf{test}'_\ell \star \pi \;\succ^*\; \mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi' \,,$$

et tels que le jeu de paramètres $\vec{\nu} = \nu_1, \ldots, \nu_k$ falsifie la formule universelle U_i.

Il y a essentiellement deux façons d'envisager l'exécution du processus de falsification donné par le théorème 7 : le *mode séquentiel* et le *mode arborescent*.

Exécution séquentielle En mode séquentiel, le processus $t \, \mathsf{test}'_1 \cdots \mathsf{test}'_\ell \star \pi$ est évalué pas à pas en faisant appel à l'expérimentateur à chaque fois qu'une instruction test_i propose un jeu de paramètres $\vec{\nu} = \nu_1, \ldots, \nu_k$ pour tester l'hypothèse $U_i \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k T(x_1, \ldots, x_k)$ à laquelle cette instruction est associée. L'expérimentateur réalise alors toutes les expériences nécessaires pour déterminer les valeurs de vérité des sous-formules de la formule $T(\vec{\nu})$ et, en fonction du résultat, déclenche la règle d'évaluation appropriée. Le théorème 7 nous garantit alors que cet algorithme termine en temps fini[23] sur un processus terminal dont la pile contient tous les paramètres de l'instance falsifiée.

Ce mode d'exécution repose cependant sur une hypothèse très forte, qui est que toutes les expériences sur lesquelles reposent l'évaluation du processus $t \, \mathsf{test}'_1 \cdots \mathsf{test}'_\ell \star \pi$ peuvent être réalisées concrètement, ce qui est peu réaliste d'un point de vue expérimental. Pour contourner ce problème (sans le résoudre totalement), il est possible de se passer totalement de l'expérimentateur en adoptant un autre mode d'exécution qui est le suivant.

[23] Sous l'hypothèse que la théorie dans laquelle la preuve de $U_1 \Rightarrow \cdots \Rightarrow U_\ell \Rightarrow \bot$ a été formalisée (ici l'arithmétique du second ordre) est 1-cohérente. Il s'agit en réalité de la seule hypothèse « non empirique » sur laquelle repose la correction de l'algorithme.

Exécution arborescente et extraction d'un arbre de Herbrand Dans le mode d'exécution arborescent, l'expérimentateur est remplacé à chaque étape d'évaluation par une base de connaissances associant une valeur de vérité (0 ou 1) à un ensemble fini de formules atomiques de la forme $p(\vec{\nu})$. L'exécution du processus $t\,\mathsf{test}'_1 \cdots \mathsf{test}'_\ell \star \pi$ démarre avec la base de connaissances vide.

À chaque fois que l'évaluation d'une instruction de test requiert la connaissance de la valeur de vérité d'une formule atomique $p(\vec{\nu})$ qui n'est pas déjà dans la base, on duplique le contexte d'exécution en posant $\mathrm{Val}(p(\vec{\nu})) = 0$ dans la première branche et $\mathrm{Val}(p(\vec{\nu})) = 1$ dans la seconde, et on continue séparément l'évaluation du processus courant dans chacune des deux branches (correspondant à deux bases de connaissances différentes).

Ce qui permet ainsi de construire un arbre de décision binaire (a priori infini) dont les nœuds sont des formules atomiques de la forme $p(\vec{\nu})$, et dont les feuilles sont des processus terminaux contenant une instance falsifiante de la théorie expérimentale, relativement à la base de connaissances courante (qui se déduit facilement du chemin conduisant de la feuille à la racine de l'arbre). Là encore, le théorème 7 permet de démontrer en utilisant un argument classique que cet arbre de décision binaire n'a pas de branche infinie, et qu'il est donc fini d'après le lemme de Koenig.

Contrairement au mode d'exécution séquentiel, la falsification n'est pas donnée ici sous la forme d'une instance falsifiante, mais sous la forme d'un arbre de Herbrand falsifiant la théorie expérimentale $U_1, \ldots, U_\ell$ indépendamment de toute interprétation des symboles de prédicats expérimentaux.[24]

Il est important de remarquer que le théorème de Herbrand aurait pu nous donner le même résultat sans passer par la théorie de la réalisabilité classique. Mais l'algorithme extrait aurait sans doute été beaucoup moins efficace, puisque dans la preuve usuelle du théorème de Herbrand, les expériences successives sont données par une énumération des formules atomiques fixée à l'avance, indépendamment du réalisateur t. Dans la méthode présentée ici, c'est au contraire le réalisateur t qui décide à chaque étape quelle sera la prochaine expérience effectuée, et on peut raisonnablement augurer qu'un choix guidé par un raisonnement mathématique (représenté sous la forme d'un réalisateur universel) sera en général plus judicieux qu'un simple choix à l'aveuglette.

Enfin, il est utile de préciser que la méthode que nous venons de présenter n'est pas limitée à l'arithmétique classique du second ordre, mais qu'elle s'étend naturellement à l'arithmétique d'ordre supérieur avec choix dépendant (réalisé avec les instructions décrites dans [61]), et même à la théorie des ensembles de Zermelo-Fraenkel avec choix dépendant.

4.2.7 Extraction de témoin et traduction négative

La méthode d'extraction que nous avons présentée à la section 4.2.3 (basée sur la réalisabilité classique) n'est pas la seule méthode permettant d'obtenir un témoin fiable à partir d'une preuve classique d'une formule Σ^0_1. Une autre méthode, plus standard en théorie de la démonstration, repose sur le fait que l'arithmétique classique et ses extensions imprédicatives (PA, PA2, PAω, Z, ZF)

[24] On notera qu'ici, l'algorithme qui permet de construire un arbre de Herbrand à partir du réalisateur t définit une fonction récursive, puisqu'il ne repose sur aucune primitive de calcul externe.

sont toutes des extensions conservatives de leur fragment intuitionniste (HA, HA2, HAω, IZ, IZF$_C$) sur la classe des formules Π^0_2.

Techniquement, cette méthode consiste à transformer une preuve classique de la formule $\exists^{\mathsf{N}} x\, f(x) = 0$ en une preuve intuitionniste de cette même formule à l'aide d'une traduction négative bien choisie (par exemple : la A-traduction de Friedman). On extrait alors facilement le témoin désiré à partir de la preuve intuitionniste en appliquant les techniques d'extraction habituelles.

Il est naturel de se demander si la méthode d'extraction de Krivine diffère essentiellement de la méthode d'extraction de Kreisel et Friedman. Sans surprise, il est possible de démontrer [79] que ces deux méthodes sont essentiellement les mêmes à travers une traduction négative bien choisie, inspirée de [84].

Le choix des formalismes La principale difficulté de ce travail réside dans le choix des formalismes source (classique) et cible (intuitionniste). Car il ne s'agit pas simplement de traduire le calcul propositionnel ni même la logique du second ordre dans un cadre intuitionniste, mais toute l'arithmétique classique du second ordre, et cela en respectant l'esprit de la réalisabilité classique. Ce qui soulève naturellement plusieurs difficultés.

La première difficulté vient du traitement des quantifications universelles. En réalisabilité classique, celles-ci sont définies par orthogonalité comme des types union infinitaires au niveau des ensembles de piles. Pour refléter cette construction dans un cadre intuitionniste, on doit donc introduire dans le calcul cible (à l'instar de [84]) des quantifications existentielles primitives au premier et au second ordre, que l'on traitera comme des types union infinitaires.

Le seconde difficulté vient du traitement des entiers naturels. En réalisabilité classique, ceux-ci sont traditionnellement définis au second ordre suivant le codage de Church ou de Krivine, ce qui laisse peu d'espoir de pouvoir suivre raisonnablement la partie purement arithmétique du calcul à travers une traduction négative. Pour résoudre ce problème, on utilisera donc les entiers primitifs introduits à la section 4.2.5 dans le calcul source, et des entiers primitifs présentés de manière plus traditionnelle dans le calcul cible.

Le calcul classique Le langage des expressions arithmétiques et des formules du calcul classique (voir Fig. 4.2) est essentiellement celui qui a été introduit à la section 4.1.3. On y a ajouté la construction $[e] \Rightarrow B$ introduite à la section 4.2.5, ainsi qu'un prédicat primitif $\mathsf{null}(e)$, avec l'abréviation $\top \equiv \mathsf{null}(0)$. Le prédicat de relativisation $\mathsf{Nat}(e)$ est défini comme le prédicat $\mathsf{Nat}'(e)$ à la section 4.2.5 en posant $\mathsf{Nat}(e) \equiv \forall Z\, (([e] \Rightarrow Z) \Rightarrow Z)$ (c'est-à-dire comme un type des entiers paresseux) tandis que les quantifications numériques sont représentées par

$$\begin{array}{rcl} \forall^{\mathsf{N}} x\, A(x) & \equiv & \forall x\, ([x] \Rightarrow A(x)) \\ \exists^{\mathsf{N}} x\, A(x) & \equiv & \forall Z\, (\forall x\, ([x] \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z) \end{array}$$

(c'est-à-dire : en utilisant les entiers primitifs en appel par valeur).

Enfin, on introduit une congruence sur les expressions arithmétiques et sur les formules, notées respectivement $e \cong e'$ et $A \cong A'$, pour traiter les axiomes de l'arithmétique directement au niveau du typage. Au niveau des expressions arithmétiques, la congruence $e \cong e'$ est simplement engendrée par la théorie équationnelle des symboles de fonctions primitives récursives, que nous ne détaillerons pas ici. À travers la règle de conversion du système de types présenté

Le langage des formules et des réalisateurs

Expr. arithmétiques	e	$::=$	$x \mid f(e_1, \ldots, e_k)$
Formules	A, B	$::=$	$X(e_1, \ldots, e_k) \mid \mathsf{null}(e)$
			$\mid A \Rightarrow B \mid [e] \Rightarrow B$
			$\mid \forall x\, A \mid \forall X\, A$
Réalisateurs	t, u	$::=$	$x \mid \lambda x\,.\,t \mid tu \mid \mathsf{cc}$
			$\mid \mathsf{s} \mid \mathsf{rec} \mid \hat{n} \mid \mathsf{k}_\pi$
Piles	π	$::=$	$\diamond \mid t \cdot \pi \quad (t \text{ clos})$
Contextes	Γ	$::=$	$\emptyset \mid \Gamma, x : A \mid \Gamma, x : [e]$
Abréviations	$\bot$	$\equiv$	$\forall Z\, Z$
	$\top$	$\equiv$	$\mathsf{null}(0)$
	$e = e'$	$\equiv$	$\forall Z\, (Z(e) \Rightarrow Z(e'))$
	$\mathsf{Nat}(e)$	$\equiv$	$\forall Z\, (([e] \Rightarrow Z) \Rightarrow Z)$
	$\forall^{\mathsf{N}} x\, A(x)$	$\equiv$	$\forall x\, ([x] \Rightarrow A(x))$
	$\exists^{\mathsf{N}} x\, A(x)$	$\equiv$	$\forall Z\, (\forall x\, ([x] \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z)$

Règles de typage

$$\frac{}{\Gamma \vdash_{\mathrm{NK}} x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{}{\Gamma \vdash_{\mathrm{NK}} t : \top}\ {\scriptstyle FV(t)\subseteq \mathrm{dom}(\Gamma)}$$

$$\frac{\Gamma, x : A \vdash_{\mathrm{NK}} t : B}{\Gamma \vdash_{\mathrm{NK}} \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : A \Rightarrow B \qquad \Gamma \vdash_{\mathrm{NK}} u : A}{\Gamma \vdash_{\mathrm{NK}} tu : B}$$

$$\frac{\Gamma, x : [e] \vdash_{\mathrm{NK}} t : B}{\Gamma \vdash_{\mathrm{NK}} \lambda x\,.\,t : [e] \Rightarrow B} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : [e] \Rightarrow B}{\Gamma \vdash_{\mathrm{NK}} tx : B}\ {\scriptstyle (x:[e])\in\Gamma}$$

$$\frac{\Gamma \vdash_{\mathrm{NK}} t : [s^n 0] \Rightarrow B}{\Gamma \vdash_{\mathrm{NK}} t\hat{n} : B} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} t : A'}\ {\scriptstyle A\cong A'}$$

$$\frac{\Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} t : \forall x\, A}\ {\scriptstyle x\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : \forall x\, A}{\Gamma \vdash_{\mathrm{NK}} t : A\{x := e\}}$$

$$\frac{\Gamma \vdash_{\mathrm{NK}} t : A}{\Gamma \vdash_{\mathrm{NK}} t : \forall X\, A}\ {\scriptstyle X\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NK}} t : \forall X\, A}{\Gamma \vdash_{\mathrm{NK}} t : A\{X := \hat{x}_1 \cdots \hat{x}_k.B\}}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NK}} \mathsf{cc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A} \qquad \frac{}{\Gamma \vdash_{\mathrm{NK}} \mathsf{s} : \forall^{\mathsf{N}} x\ \mathsf{Nat}(s(x))}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NK}} \mathsf{rec} : \forall X\, (X(0) \Rightarrow \forall^{\mathsf{N}} x\, (X(x) \Rightarrow X(s(x))) \Rightarrow \forall^{\mathsf{N}} x\, X(x))}$$

FIG. 4.2 – Une arithmétique classique du second ordre

à la Fig. 4.2, ces équations permettent de typer les réalisateurs des égalités définitionnelles des symboles de fonction définis dans la signature, ainsi que le réalisateur du 3e axiome de Peano (en utilisant les équations de la fonction prédécesseur). Au niveau des formules, la congruence $A \cong A'$ reprend les équations de la congruence $e \cong e'$ (par passage au contexte) et y ajoute l'équation $\mathsf{null}(s(e)) \cong \bot$, qui permet de typer le réalisateur du 4e axiome de Peano.

Le calcul des réalisateurs est le langage λ_c, utilisé ici avec l'instruction $\mathfrak{cc}$ (pour réaliser la loi de Peirce), des entiers primitifs $\hat{n}$ (introduits à la section 4.2.5), et deux instructions s et rec munies des règles d'évaluation :

$$\begin{array}{rcccl}
\mathsf{s} & \star & \hat{n} \cdot u \cdot \pi & \succ & u \star \widehat{n+1} \cdot \pi \\
\mathsf{rec} & \star & u_0 \cdot u_1 \cdot \hat{0} \cdot \pi & \succ & u_0 \star \pi \\
\mathsf{rec} & \star & u_0 \cdot u_1 \cdot \widehat{n+1} \cdot \pi & \succ & u_1 \star \hat{n} \cdot (\mathsf{rec}\, u_0\, u_1\, \hat{n}) \cdot \pi
\end{array}$$

(On aurait pu implémenter l'instruction rec avec un combinateur de point fixe et des instructions pour le prédécesseur et le test de nullité, mais il est plus commode de la traiter ici comme une construction primitive.) Les piles sont définies comme dans le langage λ_c, ici avec un unique fond de pile $\diamond$.

Le système de types de la Fig. 4.2 reprend les règles de déduction introduites à la section 4.1.5 (Fig. 4.1 p. 85), avec une règle de conversion dans l'esprit de la théorie des types. Sont ajoutées des règles pour traiter l'implication en appel par valeur $[e] \Rightarrow B$, qui suffisent à dériver les jugements de la forme $\Gamma \vdash_{\mathrm{NK}} t : \mathsf{Nat}(e)$ pour chaque expression arithmétique e (en supposant que le contexte contient une hypothèse de type $[x]$ ou de type $\mathsf{Nat}(x)$ pour chaque variable libre de e). On y trouve également des règles pour les constantes s et rec.

Le lemme d'adéquation de la réalisabilité classique (Prop. 26) s'étend évidemment au calcul de la Fig. 4.2 en interprétant le nouveau symbole de prédicat null par $\|\mathsf{null}(0)\| = \varnothing$ et $\|\mathsf{null}(n)\| = \Pi$ pour tout $n \geq 1$.

Le calcul intuitionniste Le langage des formules du calcul intuitionniste (voir Fig. 4.3) reprend l'essentiel des constructions du calcul classique, en y ajoutant une conjonction primitive ainsi que des quantifications existentielles primitives (i.e. à la Curry) du premier et du second ordre. L'implication en appel par valeur disparaît et est remplacée par un prédicat $\mathsf{Nat}(e)$ primitif (et non défini comme dans le calcul classique). On notera que dans ce cadre, les quantifications numériques deviennent :

$$\begin{array}{rcl}
\forall^{\mathsf{N}} x\, A(x) & \equiv & \forall x\, (\mathsf{Nat}(x) \Rightarrow A(x)) \\
\exists^{\mathsf{N}} x\, A(x) & \equiv & \exists x\, (\mathsf{Nat}(x) \wedge A(x))
\end{array}$$

La formule $\top$ est alors définie par $\top \equiv \exists Z Z$.

La congruence sur les formules est définie à partir de la congruence sur les expressions arithmétiques (définie comme dans le calcul classique) en ajoutant les équations $\mathsf{null}(0) \cong \top$, $\mathsf{null}(s(e)) \cong \bot$ ainsi que les nouvelles règles :

$$\begin{array}{rcll}
(\exists x\, A(x)) \Rightarrow B & \cong & \forall x\, (A(x) \Rightarrow B) & (\text{si } x \notin FV(B)) \\
(\exists X\, A(X)) \Rightarrow B & \cong & \forall X\, (A(X) \Rightarrow B) & (\text{si } X \notin FV(B))
\end{array}$$

Ces deux règles permettent d'implémenter les règles d'élimination des quantifications existentielles primitives et jouent un rôle crucial dans la preuve de

Le langage des formules et des réalisateurs

Expr. arithmétiques	e	$::=$	$x \mid f(e_1, \ldots, e_k)$
Formules	A, B	$::=$	$X(e_1, \ldots, e_k) \mid A \Rightarrow B$
			$\mid \mathsf{null}(e) \mid \mathsf{Nat}(e)$
			$\mid \forall x\, A \mid \forall X\, A \mid \exists x\, A \mid \exists X\, A$
Réalisateurs	t, u	$::=$	$x \mid \lambda x\,.\,t \mid tu \mid \mathsf{pair}$
			$\mid \mathsf{fst} \mid \mathsf{snd} \mid \mathsf{0} \mid \mathsf{s} \mid \mathsf{rec}$
Contextes	Γ	$::=$	$\emptyset \mid \Gamma, x : A$
Abréviations	$\bot$	$\equiv$	$\forall Z\, Z$
	$\top$	$\equiv$	$\exists Z\, Z$
	$e = e'$	$\equiv$	$\forall Z\, (Z(e) \Rightarrow Z(e'))$
	$\forall^{\mathsf{N}} x\, A(x)$	$\equiv$	$\forall x\, (\mathsf{Nat}(x) \Rightarrow A(x))$
	$\exists^{\mathsf{N}} x\, A(x)$	$\equiv$	$\exists x\, (\mathsf{Nat}(x) \wedge A(x))$

Règles de typage

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{}{\Gamma \vdash_{\mathrm{NJ}} t : \top}\ {\scriptstyle FV(t)\subseteq \mathrm{dom}(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} t : A'}\ {\scriptstyle A\cong A'}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{pair} : A \Rightarrow B \Rightarrow A \wedge B}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{fst} : A \wedge B \Rightarrow A} \qquad \frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{snd} : A \wedge B \Rightarrow B}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{0} : \mathsf{Nat}(0)} \qquad \frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{s} : \forall^{\mathsf{N}} x\ \mathsf{Nat}(s(x))}$$

$$\frac{}{\Gamma \vdash_{\mathrm{NJ}} \mathsf{rec} : \forall X\, (X(0) \Rightarrow \forall^{\mathsf{N}} x\, (X(x) \Rightarrow X(s(x))) \Rightarrow \forall^{\mathsf{N}} x\, X(x))}$$

$$\frac{\Gamma, x : A \vdash_{\mathrm{NJ}} t : B}{\Gamma \vdash_{\mathrm{NJ}} \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : A \Rightarrow B \qquad \Gamma \vdash_{\mathrm{NJ}} u : A}{\Gamma \vdash_{\mathrm{NJ}} tu : B}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} t : \forall x\, A}\ {\scriptstyle x\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : \forall x\, A}{\Gamma \vdash_{\mathrm{NJ}} t : A\{x := e\}}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} t : A}{\Gamma \vdash_{\mathrm{NJ}} t : \forall X\, A}\ {\scriptstyle X\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : \forall X\, A}{\Gamma \vdash_{\mathrm{NJ}} t : A\{X := \hat{x}_1 \cdots \hat{x}_k.B\}}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} t : A\{x := e\}}{\Gamma \vdash_{\mathrm{NJ}} t : \exists x\, A} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} t : A\{X := \hat{x}_1 \cdots \hat{x}_k.B\}}{\Gamma \vdash_{\mathrm{NJ}} t : \exists X\, A}$$

FIG. 4.3 – Une arithmétique intuitionniste du second ordre

correction de la traduction négative vis-à-vis des règles d'élimination des quantifications universelles du premier et du second ordre (Prop. 1). On notera que dans un modèle de réalisabilité à la Kleene, ces deux règles de conversion sont valides du moment qu'on interprète l'implication par la flèche de la réalisabilité et les quantifications existentielles primitives comme des unions.

Le calcul des réalisateurs intuitionnistes est un λ-calcul traditionnel étendu avec des paires, des projections et toutes les constructions nécessaires pour travailler avec les entiers de Peano. La réduction est ici décomposée en deux relations : une relation de *réduction faible* $t \succ_w u$ qui ne passe pas sous les abstractions, et une relation de *réduction interne* $t \succ_i u$, qui décrit au contraire les étapes de réduction sous les abstractions :

$$\frac{}{(\lambda x\,.\,t)u \succ_w t\{x := u\}} \qquad \frac{}{\mathsf{rec}\,u_0\,u_1\,0 \succ_w u_0} \qquad \frac{}{\mathsf{rec}\,u_0\,u_1\,(\mathsf{s}\,t) \succ_w u_1\,t\,(\mathsf{rec}\,u_0\,u_1\,t)}$$

$$\frac{}{\mathsf{fst}\,\langle t_1; t_2\rangle \succ_w t_1} \qquad \frac{}{\mathsf{snd}\,\langle t_1; t_2\rangle \succ_w t_2} \qquad \frac{t \succ_w t'}{tu \succ_w t'u} \qquad \frac{u \succ_w u'}{tu \succ_w tu'}$$

$$\frac{t \succ_w t'}{\lambda x\,.\,t \succ_i \lambda x\,.\,t'} \qquad \frac{t \succ_i t'}{tu \succ_i t'u} \qquad \frac{u \succ_i u'}{tu \succ_i tu'} \qquad \frac{t \succ_i t'}{\lambda x\,.\,t \succ_i \lambda x\,.\,t'}$$

(On notera que la réduction faible est possible à gauche et à droite dans les applications.) La relation de réduction en une étape (notée $t \to u$) est alors définie comme l'union de ces deux relations.

Le système de types de la Fig. 4.3 contient les règles attendues pour la logique intuitionniste du second ordre avec une conjonction et des entiers primitifs. On notera que les quantifications existentielles primitives ont chacune leur règle d'introduction, mais aucune règle d'élimination : celle-ci est ici prise en charge par la congruence sur les formules à travers la règle de conversion.

Le modèle de réalisabilité intuitionniste Le calcul décrit à la Fig. 4.3 a un modèle de réalisabilité intuitionniste très simple dans lequel les formules sont interprétées par des ensembles de termes clos stables par antiréduction faible (au sens de la relation $\succ_w$). Dans ce modèle, l'implication est interprétée par la flèche de la réalisabilité, les quantifications universelles par des intersections, les quantifications existentielles par des unions et la formule $\mathsf{null}(n)$ par l'ensemble de tous les termes clos si $n = 0$, et par l'ensemble vide sinon. (Voir [79] pour les détails de la construction.) Comme dans le cas classique, on dispose d'un lemme d'adéquation exprimant que si $\vdash_{\mathrm{NJ}} t : A$, alors $t \Vdash_{\mathrm{NJ}} A$.

La traduction négative Suivant l'esprit de la réalisabilité classique, chaque formule A du calcul classique est traduite en deux formules du calcul intuitionniste : une formule $A^{\perp}$ qui représente un type de piles, et une formule $A^{\neg\neg}$ qui représente un type de termes. La formule $A^{\neg\neg}$ est définie à partir de la formule $A^{\perp}$ par l'équation $A^{\neg\neg} \equiv \neg_R A^{\perp} \equiv A^{\perp} \Rightarrow R$, où R est une formule fixée qui paramétrise la traduction. (Intuitivement : $R = \perp\!\!\!\perp$.) La formule $A^{\perp}$ est définie par induction sur la structure de A par les équations

$$\begin{array}{rcl@{\qquad}rcl}
(X(e_1,\ldots,e_k))^{\perp} & \equiv & X(e_1,\ldots,e_k) & (\mathsf{null}(e))^{\perp} & \equiv & \mathsf{null}(h(e)) \\
(A \Rightarrow B)^{\perp} & \equiv & A^{\neg\neg} \wedge B^{\perp} & (\forall x\,A)^{\perp} & \equiv & \exists x\,A^{\perp} \\
(\{e\} \Rightarrow B)^{\perp} & \equiv & \mathsf{Nat}(e) \wedge B^{\perp} & (\forall X\,A)^{\perp} & \equiv & \exists X\,A^{\perp}
\end{array}$$

où h est un symbole de fonction défini par $h(0) = 1$ et $h(s(x)) = 0$.

Lemme 16 — *Si $A \cong A'$, alors $A^\perp \cong A'^\perp$ et $A^{\neg\neg} \cong A'^{\neg\neg}$.*

Notons également que dans le calcul cible on a

$$(\forall v\, A(v))^{\neg\neg} \;\equiv\; \exists v\, A(v)^\perp \Rightarrow R \;\cong\; \forall v\, (A(v)^\perp \Rightarrow R) \;\equiv\; \forall v\, (A(v)^{\neg\neg})$$

(où v est une variable du premier ou du second ordre).

La traduction CPS On définit ensuite par induction mutuelle deux traductions $t \mapsto t^*$ et $\pi \mapsto \pi^*$ des termes et des piles du calcul source vers les termes du calcul cible. La traduction des piles est donnée par

$$(\diamond)^* \;\equiv\; \alpha^* \qquad \text{et} \qquad (t \cdot \pi)^* \;\equiv\; \langle t^*;\ \pi^* \rangle$$

où $\diamond^*$ est un terme quelconque (dont nous ne nous servirons pas dans la suite), tandis que la traduction des termes est donnée par

$$\begin{array}{rcl}
x^* & \equiv & x \\
(tu)^* & \equiv & \lambda k\,.\, t^*\,\langle u^*; k\rangle \\
(\lambda x\,.\,t)^* & \equiv & \lambda k\,.\,(\lambda x\,.\,t^*)\,(\mathsf{fst}\,k)\,(\mathsf{snd}\,k) \\
(\mathsf{k}_\pi)^* & \equiv & \lambda k\,.\,\mathsf{fst}\,k\,\pi^* \\
(\mathsf{cc})^* & \equiv & \lambda k\,.\,\mathsf{fst}\,k\,\langle(\lambda z k'\,.\,\mathsf{fst}\,k'\,z)\,(\mathsf{snd}\,k);\ \mathsf{snd}\,k\rangle \\
(\hat{n})^* & \equiv & \mathsf{s}^n\,\mathsf{0} \\
(\mathsf{s})^* & \equiv & \lambda k\,.\,\mathsf{fst}\,(\mathsf{snd}\,k)\,\langle \mathsf{s}\,(\mathsf{fst}\,k);\ \mathsf{snd}\,(\mathsf{snd}\,k)\rangle \\
(\mathsf{rec})^* & \equiv & \lambda k\,.\,\mathsf{rec}\ (\mathsf{fst}_1\,k)\ (\lambda p y k'\,.\,\mathsf{fst}\,(\mathsf{snd}\,k)\,\langle p; \langle y; k'\rangle\rangle) \\
 & & \qquad (\mathsf{fst}\,(\mathsf{snd}\,(\mathsf{snd}\,k)))\ (\mathsf{snd}\,(\mathsf{snd}\,(\mathsf{snd}\,k)))
\end{array}$$

(où $\langle t; u\rangle$ est une abréviation pour $\mathsf{pair}\,t\,u$).

Proposition 1 (Correction vis-à-vis du typage)
— *Si $\Gamma \vdash_{\mathrm{NK}} t : A$ (Fig. 4.2), alors $\Gamma^{\neg\neg} \vdash_{\mathrm{NJ}} t^* : A^{\neg\neg}$ (Fig. 4.3).*

On pourrait s'attendre à ce que toute étape d'évaluation $t_1 \star \pi_1 \succ t_2 \star \pi_2$ dans le calcul source se traduise en une ou plusieurs étapes de réduction faible $t_1^*\,\pi_1^* \succ_w^+ t_2^*\,\pi_2^*$ dans le calcul cible. C'est vrai pour toutes les règles d'évaluation du calcul classique, sauf pour la seconde règle d'évaluation du récurseur rec. On doit donc formuler le résultat de simulation de la manière suivante :

Proposition 33 (Simulation en un pas) — *Si $t_1 \star \pi_1 \succ t_2 \star \pi_2$ (en un pas), alors $t_1^*\,\pi_1^* \succ_w^+ t_2^*\,u$ (en un ou plusieurs pas) pour un certain terme $u =_i \pi_2^*$.*

Autrement dit, la relation d'évaluation du calcul source est bien simulée dans le calcul cible, modulo certaines réductions administratives sous les λ.

Corollaire 5 (Simulation de l'évaluation) — *Si $t_1 \star \pi_1 \succ^* t_2 \star \pi_2$, alors $t_1^*\,\pi_1^* \succ_w^* u$ pour un certain terme $u =_i t_2^*\,\pi_2^*$.*

Interprétation de la méthode d'extraction dans le cas Σ_1^0 Il s'agit à présent d'interpréter à travers la traduction négative que nous venons de définir la méthode d'extraction présentée à la section 4.2.3.

Pour cela, on se place dans un cadre un tout petit peu différent, en partant non pas d'un réalisateur universel de la formule $\exists^{\mathsf{N}} x\ (f(x) = 0)$, mais d'un réalisateur universel de la formule $\exists^{\mathsf{N}} \mathsf{null}(f(x))$, en supposant que :

– le prédicat null est interprété par la fonction de fausseté définie par

$$\|\mathsf{null}(n)\| \quad = \quad \begin{cases} \varnothing & \text{si } n = 0, \\ \Pi & \text{sinon} \end{cases}$$

– la quantification numérique est définie ici par

$$\exists^{\mathsf{N}} x\, A(x) \quad \equiv \quad \forall Z\, (\forall x\, ([x] \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z)$$

Dans ce nouveau cadre, la proposition 29 s'adapte de la manière suivante :

Proposition 34 — *Si t_0 est un réalisateur universel de la formule $\exists^{\mathsf{N}}\mathsf{null}(f(x))$, si d_f est une fonction de décision de la nullité de f et si stop est une instruction d'arrêt, alors il existe un entier $n \in \mathbb{N}$ tel que $f(n) = 0$ et*

$$t_0 \star (\lambda xy\,.\, d_A\, x\, (\mathsf{stop}\, x)\, y) \cdot \diamond \quad \succ^* \quad \mathsf{stop} \star \bar{n} \cdot \diamond\,.$$

Pour pouvoir analyser le comportement du processus

$$p_0 \quad \equiv \quad t_0 \star (\lambda xy\,.\, d_A\, x\, (\mathsf{stop}\, x)\, y) \cdot \diamond$$

à travers la traduction-CPS, on doit faire les hypothèses de typage suivantes :

1. $\vdash_{\mathrm{NK}} t_0 : \exists^{\mathsf{N}}\mathsf{null}(f(x))$ (au sens de la Fig. 4.2) ;
2. $\vdash_{\mathrm{NK}} d_f : \forall Z\, \forall x\, ([x] \Rightarrow Z(0) \Rightarrow \forall y\, Z(s(y)) \Rightarrow Z(f(x)))$

La condition de typage sur d_f implique que d_f est un réalisateur universel de la formule $\forall Z\, \forall x\, ([x] \Rightarrow Z(0) \Rightarrow \forall y\, Z(s(y)) \Rightarrow Z(f(x)))$, et il est facile de vérifier que cette dernière condition est équivalente (en réalisabilité classique) au fait que d_f est une fonction de décision de la nullité de f. Dans le système de types de la Fig. 4.2, on construit aisément une telle fonction de décision à partir d'un terme $\check{f} : \forall^{\mathsf{N}}\, \mathsf{Nat}(f(x))$ en posant $d_f \equiv \lambda xyz\,.\, \check{f}\, x\, (\mathsf{rec}\, y\, (\lambda\_\,.\, z))$.

La condition de typage sur d_f entraîne que

$$\vdash_{\mathrm{NK}} d_f : \forall Z\, \forall x\, \big([x] \Rightarrow Z(x,0) \Rightarrow \forall y\, Z(x, s(y)) \Rightarrow Z(x, f(x))\big)$$

c'est-à-dire, à travers la traduction négative :

$$\vdash_{\mathrm{NJ}} d_f^* : \forall Z\, \forall x\, \big(\mathsf{Nat}(x) \wedge \neg_R Z(x,0) \wedge \forall y\, \neg_R Z(x, s(y)) \wedge Z(x, f(x)) \Rightarrow R\big)$$

En instanciant Z dans le modèle de réalisabilité intuitionniste de telle sorte que $Z(x,y) = \top$ si $f(x) = y$ et $Z(x,y) = \bot$ sinon, on remarque que :

$$d_f^* \Vdash_{\mathrm{NJ}} \forall x\, (\mathsf{Nat}(x) \wedge \neg_R \mathsf{null}(f(x)) \wedge \neg_R \mathsf{null}(f(h(x))) \wedge \top \Rightarrow R)$$

(en utilisant le fait que $Z(x,0) = \mathsf{null}(f(x))$ et $Z(x, s(y)) = \mathsf{null}(h(f(x)))$ dans le modèle). On pose alors $R \equiv \exists^{\mathsf{N}} x\, \mathsf{null}(f(x))$ et $\mathsf{stop}^* \equiv \lambda x\,.\, x$. Ce qui nous donne

$$\vdash_{\mathrm{NJ}} \mathsf{stop} : \forall x\, (\mathsf{Nat}(x) \Rightarrow \mathsf{null}(f(x)) \Rightarrow R)$$

puis

$$(\lambda xy\,.\, d_f\, x\, (\mathsf{stop}\, x)\, y)^* \;\Vdash_{\mathrm{NJ}}\; \forall x\, (\mathsf{Nat}(x) \wedge (\mathsf{null}(h(s(x))) \Rightarrow R) \wedge \top \Rightarrow R)$$

et finalement

$$p_0^* \;\Vdash_{\mathrm{NJ}}\; R \;\equiv\; \exists^{\mathsf{N}} x\, \mathsf{null}(f(x))\,.$$

Autrement dit, nous avons montré que le mécanisme d'extraction de témoin de la Prop. 34 revient à transformer une preuve classique t_0 de la formule $\exists^{\mathsf{N}} x\, \mathsf{null}(f(x))$ en un réalisateur intuitionniste p_0^* de cette même formule à travers la traduction négative $A \mapsto A^{\neg\neg}$.

4.3 Extension au calcul des constructions

Nous allons maintenant montrer comment le modèle de réalisabilité classique (défini à l'origine pour l'arithmétique du second ordre) peut être étendu de manière conservative (en un sens qui sera précisé à la section 4.3.6) au calcul des constructions avec univers (CC_ω), le noyau du système Coq. Cette extension théorique est en effet un préalable indispensable pour pouvoir utiliser les méthodes d'extraction décrites à la section 4.2 dans Coq.

La richesse du calcul des constructions fait apparaître des difficultés qui n'existent pas dans l'arithmétique du second ordre. Le premier problème vient du fait que dans ce formalisme, les types (dont les propositions) et plus généralement les objets construits dans les univers supérieurs sont susceptibles de dépendre des preuves. Il est à noter que ce trait du langage — bien utile pour définir la notion de type sous-ensemble — pose déjà des problèmes dans le cadre de la formalisation des mathématiques sur machine. Par exemple, le type des nombres premiers est défini dans Coq comme le type des *paires dépendantes* (p, π) formées d'un entier p et d'une preuve π de la proposition « p est premier ». Pour que deux nombres premiers (p, π) et (p', π') soient égaux dans ce système, il ne suffit pas que les entiers p et p' soient égaux ; il faut encore que les preuves π et π' qui les accompagnent le soient également.

La façon habituelle de résoudre ce problème est d'introduire un axiome de *proof-irrelevance*[25] (ou de modifier la règle de conversion en conséquence) pour faire en sorte que toutes les preuves d'une même proposition soient égales. Pendant longtemps on a pensé[26] que le principe de la *proof-irrelevance* était incompatible avec une extraction dans le style de AF2 [57], où les programmes ne sont pas extraits à partir du squelette prédicatif des objets définis dans les univers supérieurs (comme c'est le cas pour le mécanisme d'extraction constructive de Coq [65]), mais à partir des termes de preuves. Ainsi que nous allons le voir, ce n'est pas le cas. La *proof-irrelevance* est non seulement compatible avec l'extraction de programmes en logique classique dans la sorte Prop, mais en plus elle nous dispense d'introduire des règles de réduction pour les constantes « classiques » (telles que *call*/*cc*) au niveau de la théorie des types source, simplement parce que ces constantes deviennent transparentes dans la règle de conversion. Dans la version du calcul des constructions que nous allons présenter, l'opérateur *call*/*cc*[27] peut être défini comme une constante inerte sans que cela pose le moindre problème au niveau du test de conversion.

4.3.1 Un calcul des constructions *proof-irrelevant*

Le calcul des constructions *proof-irrelevant* avec univers ($\mathrm{CC}_\omega^{\mathrm{irr}}$) est construit sur la même syntaxe que le calcul des constructions avec univers (CC_ω) :

$$\begin{array}{lllll} \textbf{Sortes} & s \in \mathscr{S} & ::= & * \mid \Box_i & (i \geq 1) \\ \textbf{Termes} & M, N, T, U & ::= & x \mid s \mid \Pi x\!:\!T\,.\,U \mid \lambda x\!:\!T\,.\,M \mid MN & \end{array}$$

[25] Une expression qu'on peut traduire par : « indiscernabilité des preuves »

[26] Et notamment l'auteur de ces lignes.

[27] Pour laquelle on serait bien en peine de définir des règles de réduction appropriés. De fait, la théorie des types semble totalement rétive à l'introduction des opérateurs de contrôle, qui font très mauvais ménage avec les types dépendants. De manière surprenante, ce problème disparaît totalement dans le modèle de réalisabilité, dans la mesure où les dénotations et les réalisateurs vivent dans des mondes bien cloisonnés.

Ici, $*$ désigne la sorte des *propositions* tandis que $\Box_i$ ($i \geq 1$) désigne le $i$ème *univers prédicatif*. L'ensemble des sortes $\mathscr{S}$ est muni d'un ensemble d'axiomes $\mathscr{A} \subset \mathscr{S}^2$ et d'un ensemble de règles $\mathscr{R} \subset \mathscr{S}^3$ définis par :

$$\begin{aligned} \mathscr{A} &= \{(* : \Box_1);\ (\Box_i : \Box_{i+1}) \mid i \geq 1\} \\ \mathscr{R} &= \{(*,*,*);\ (\Box_i,*,*);(*,\Box_i,\Box_i);\ (\Box_i,\Box_i,\Box_i) \mid i \geq 1\} \end{aligned}$$

ainsi que d'un ordre (total) sur les sortes — l'*ordre cumulatif* — engendré par les relations $* \leq \Box_1$ et $\Box_i \leq \Box_{i+1}$ ($i \geq 1$).[28] Le système de types de $\mathrm{CC}_\omega^{\mathrm{irr}}$ est défini à partir de quatre formes de jugement :

$\vdash \Gamma$ **ctx**	« Γ est un contexte bien formé »
$\Gamma \vdash M : T$	« dans le contexte Γ, le terme M a pour type T »
$\Gamma \vdash T_1 \leq T_2$	« dans le contexte Γ, T_1 est un sous-type de T_2 »
$\Gamma \vdash M_1 = M_2 : T$	« dans Γ, M_1 et M_2 sont des termes égaux de type T »

Les règles d'inférence correspondantes sont données à la Fig. 4.4. (Voir [77] pour les propriétés de ce système.)

On notera que dans la règle de conversion de la Fig. 4.4, la notion de β-conversion non typée (sur laquelle est basée la règle de conversion des PTS) est remplacée par une notion d'égalité typée qui procède ici à deux formes d'identification : l'identification des β-radicaux bien typés avec leurs β-contractés, mais aussi l'identification des preuves d'une même proposition.

4.3.2 La structure de Π-ensemble

Le modèle de réalisabilité que nous allons présenter est directement inspiré du modèle de réalisabilité intuitionniste du calcul des constructions [66, 97] basé sur la notion de ω-ensemble [53] et ses variantes [97, 3, 72]. Ici, le passage de la logique intuitionniste à la logique classique conduit naturellement à remplacer la notion de ω-ensemble (construite sur l'idée qu'un terme t peut défendre une dénotation) par la notion de Π-ensemble (construite sur l'idée qu'une pile π peut s'opposer à une dénotation). Ce qui nous amène à la définition suivante.

Soit $\perp\!\!\!\perp$ un pôle fixé. Un Π-ensemble est un couple $X = (|X|, \perp_X)$ formé par un *support* $|X|$ et une relation d'*opposition locale* $\perp_X \subseteq |X| \times \Pi$. Étant donné un élément $x \in |X|$, on note $x^{\perp_X} = \{\pi \in \Pi\ :\ x \perp_X \pi\}$ l'ensemble des *opposants* de x dans le Π-ensemble X. Ce qui permet de définir la relation de *réalisabilité locale à* X en posant

$$\begin{aligned} t \Vdash_X x \quad &\equiv \quad \forall \pi\ (x \perp_X \pi \Rightarrow t \star \pi \in \perp\!\!\!\perp) \\ &\equiv \quad t \in (x^{\perp_X})^{\perp\!\!\!\perp} \end{aligned}$$

pour tous $t \in \Lambda$ et $x \in |X|$.

Remarque Contrairement aux ω-ensembles, on ne demande pas que chaque élément de X soit réalisé par au moins une quasi-preuve, ni même par quelque terme que ce soit. Cependant, tous les éléments de X sont réalisés par au moins un terme dès que le pôle $\perp\!\!\!\perp$ est non vide (cf section 4.1.4).

[28] On notera qu'ici, l'interprétation de la cumulativité $* \leq \Box_1$ ne pose pas de problème dans le modèle de réalisabilité, car le formalisme source ($\mathrm{CC}_\omega^{\mathrm{irr}}$) est présenté avec un jugement d'égalité dans le style de Martin-Löf. Voir à ce sujet la discussion dans [80].

Règles de formation de contexte

$$\frac{}{\vdash \emptyset\ \mathbf{ctx}} \qquad \frac{\Gamma \vdash T : s \qquad x \notin \mathrm{dom}(\Gamma)}{\vdash \Gamma, x : T\ \mathbf{ctx}}$$

Règles de typage

$$\frac{\vdash \Gamma\ \mathbf{ctx}}{\Gamma \vdash x : T}\ {\scriptstyle (x:T)\in\Gamma} \qquad \frac{\vdash \Gamma\ \mathbf{ctx}}{\Gamma \vdash s_1 : s_2}\ {\scriptstyle (s_1,s_2)\in\mathscr{A}} \qquad \frac{\Gamma \vdash M : T \qquad \Gamma \vdash T \leq T'}{\Gamma \vdash M : T'}$$

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash \Pi x : T\,.\,U : s_3}\ {\scriptstyle (s_1,s_2,s_3)\in\mathscr{R}}$$

$$\frac{\Gamma \vdash \Pi x : T\,.\,U : s \qquad \Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T\,.\,M : \Pi x : T\,.\,U} \qquad \frac{\Gamma \vdash M : \Pi x : T\,.\,U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{x := N\}}$$

Règles de sous-typage

$$\frac{\Gamma \vdash T = T' : s}{\Gamma \vdash T \leq T'} \qquad \frac{\Gamma \vdash T \leq T' \qquad \Gamma \vdash T' \leq T''}{\Gamma \vdash T \leq T''}$$

$$\frac{\vdash \Gamma\ \mathbf{ctx} \qquad s_1 \leq s_2}{\Gamma \vdash s_1 \leq s_2} \qquad \frac{\Gamma \vdash T = T' : s \qquad \Gamma \vdash U \leq U'}{\Gamma \vdash \Pi x : T\,.\,U \leq \Pi x : T'\,.\,U'}$$

Règles d'égalité

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash M = M : T} \qquad \frac{\Gamma \vdash M = M' : T \qquad \Gamma \vdash M' = M'' : T}{\Gamma \vdash M = M'' : T}$$

$$\frac{\Gamma \vdash M = M' : T}{\Gamma \vdash M' = M : T} \qquad \frac{\Gamma \vdash M = M' : T \qquad \Gamma \vdash T \leq T'}{\Gamma \vdash M = M' : T'}$$

$$\frac{\Gamma \vdash T = T' : s_1 \qquad \Gamma, x : T \vdash U = U' : s_2 \qquad (s_1, s_2, s_3) \in \mathscr{R}}{\Gamma \vdash \Pi x : T\,.\,U = \Pi x : T'\,.\,U' : s_3}$$

$$\frac{\Gamma \vdash T = T' : s \qquad \Gamma, x : T \vdash M = M' : U}{\Gamma \vdash \lambda x : T\,.\,M = \lambda x : T'\,.\,M' : \Pi x : T\,.\,U}$$

$$\frac{\Gamma \vdash M = M' : \Pi x : T\,.\,U \qquad \Gamma \vdash N = N' : T}{\Gamma \vdash MN = M'N' : U\{x := N\}}$$

$$\frac{\Gamma \vdash \Pi x : T\,.\,U : s \qquad \Gamma, x : T \vdash M : U \qquad \Gamma \vdash N : T}{\Gamma \vdash (\lambda x : T\,.\,M)N = M\{x := N\} : U\{x := N\}}$$

$$\frac{\Gamma \vdash T : * \qquad \Gamma \vdash M : T \qquad \Gamma \vdash M' : T}{\Gamma \vdash M = M' : T}\ {\scriptstyle \text{(proof-irrelevance)}}$$

FIG. 4.4 – Règles de typage de $\mathrm{CC}^{\mathrm{irr}}_{\omega}$

Π-ensembles grossiers On dit qu'un Π-ensemble X est *grossier* si $\bot_X = \varnothing$. Par orthogonalité, on a alors $t \Vdash_X x$ pour tous $t \in \Lambda$ et $x \in |X|$. (Pour la même raison que celle nous avons indiqué à la fin de la section 4.1.5 p. 86.) On notera que tout ensemble X peut être muni d'une structure de Π-ensemble grossier en posant $|X| = X$ et $\bot_X = \varnothing$.

Produit cartésien d'une famille de Π-ensembles Étant donnée une famille de Π-ensembles $(Y_x)_{x\in|X|}$ indicée par le support d'un Π-ensemble X, on définit le Π-ensemble produit $\Pi x : X \,.\, Y_x$ par

$$|\Pi x : X \,.\, Y_x| = \prod_{x\in|X|} |Y_x| \quad \text{et} \quad f^{\bot_{\Pi x:X\,.\,Y_x}} = \bigcup_{x\in|X|} \Big((x^{\bot_X})^{\perp\!\!\!\perp} \cdot (f(x)^{\bot_{Y_x}}) \Big)$$

pour toute fonction $f \in |\Pi x : X \,.\, Y_x|$.

Π-ensembles dégénérés On dit qu'un Π-ensemble X est *dégénéré* si son support est un singleton. Un Π-ensemble dégénéré est caractérisé par son unique élément x_0 et par l'ensemble de piles $x_0^{\bot_X}$, que l'on note plus simplement $X^\bot$. Intuitivement, un Π-ensemble dégénéré n'est rien d'autre qu'une valeur de fausseté attachée à un point, et c'est par cette structure qu'on interprétera naturellement les propositions. On vérifie alors que :

Proposition 35 — *Soit X un Π-ensemble quelconque et $(Y_x)_{x\in|X|}$ une famille de Π-ensembles dégénérés indicée par le support de X. Alors le produit de la famille $(Y_x)_{x\in|X|}$ est un Π-ensemble dégénéré, et*

$$(\Pi x : X \,.\, Y_x)^\bot \;=\; \bigcup_{x\in|X|} \Big((x^{\bot_X})^{\perp\!\!\!\perp} \cdot Y_x^\bot \Big)$$

Si de plus le Π-ensemble X est dégénéré (cas de l'implication), on a alors

$$(\Pi x : X \,.\, Y_x)^\bot \;=\; (X \to Y)^\bot \;=\; (X^\bot)^{\perp\!\!\!\perp} \cdot Y^\bot \,.$$

La proposition ci-dessus exprime plusieurs choses. Au niveau des supports on retrouve la sémantique attendue de la *proof-irrelevance*, qui sera précieuse pour interpréter le caractère imprédicatif de la sorte $*$. Au niveau des valeurs de fausseté, on retrouve cette idée que le produit dépendant est un mélange de quantification universelle (reflétée par une union du point de vue de l'opposant) et d'implication (reflétée par le produit cartésien $(x^{\bot_X})^{\perp\!\!\!\perp} \cdot Y_x^\bot$ du point de vue de l'opposant). Et dans le cas d'une implication, on retrouve exactement la valeur de fausseté de la réalisabilité classique.

Sous-typage Étant donnés deux Π-ensembles X et Y, on dit que X est un *sous-type* de Y et on note $X \leq Y$ si $|X| \subseteq |Y|$ et si $x^{\bot_X} \supseteq x^{\bot_Y}$ pour tout $x \in X$. D'après cette dernière condition, il est clair que $t \Vdash_X x$ implique $t \Vdash_Y x$ pour tous $t \in \Lambda$ et $x \in |X|$.

4.3.3 Construction du modèle

Dans ce qui suit, on travaille dans ZF étendu avec un axiome exprimant l'existence d'une infinité de cardinaux inaccessibles (pour interpréter la hiérarchie d'univers prédicatifs), et on se fixe un pôle $\perp\!\!\!\perp$.

Une représentation alternative des fonctions On suppose que les fonctions sont représentées en théorie des ensembles non pas par des graphes mais par des traces :

$$\begin{array}{rcl} f \text{ fonction} & \equiv & \forall p \in f\ \exists x\ \exists y\ \ p = (x,y) \\ (x \in D \mapsto E_x) & = & \{(x,z)\ :\ x \in D\ \wedge\ z \in E_x\} \\ f(x) & = & \{z\ :\ \exists x\ (x,z) \in f\} \end{array}$$

(Voir [2, 80] pour les détails de cette représentation.) Grâce à ce changement de représentation, toute fonction constante retournant l'ensemble vide est elle-même représentée par l'ensemble vide, ce qui nous permettra d'interpréter toutes les preuves par $\bullet = \varnothing$ et toutes les propositions par des Π-ensembles dégénérés de support le singleton $\{\bullet\}$.[29]

Interprétation des sortes Soit $(\lambda_i)_{i\geq 1}$ une suite croissante de cardinaux inaccessibles. On pose :

$$\begin{array}{rcll} \mathcal{U}_0 & = & \{(\{\bullet\},\ \{\bullet\} \times S)\ :\ S \subseteq \Pi\} & \\ \mathcal{U}_i & = & \{(E,\ R)\ :\ E \in (V_{\lambda_i} \setminus \{\varnothing\}),\ R \subseteq E \times \Pi\} & (i \geq 1) \end{array}$$

Chaque ensemble (de Π-ensembles) $\mathcal{U}_i$ est lui-même muni d'une structure de Π-ensemble grossier en posant $\mathcal{U}'_i = (\mathcal{U}_i, \varnothing)$ $(i \in \omega)$. On définit alors le modèle $\mathcal{M}$ par $\mathcal{M} = \bigcup_{i\in\omega} \bigcup_{X\in\mathcal{U}_i} |X|$.

La fonction d'interprétation On note $\mathrm{Val}_{\mathcal{M}}$ l'ensemble des valuations à valeurs dans $\mathcal{M}$. à chaque terme M on associe une fonction partielle $[\![M]\!]_{\_} : \mathrm{Val}_{\mathcal{M}} \rightharpoonup \mathcal{M}$ qui est définie par induction sur M par les équations

$$\begin{array}{rclcrcl} [\![x]\!]_\rho & = & \rho(x) & \qquad & [\![\Pi x : T\,.\,U]\!]_\rho & = & \Pi v : [\![T]\!]_\rho\,.\,[\![U]\!]_{\rho,x\leftarrow v} \\ [\![*]\!]_\rho & = & \mathcal{U}'_0 & & [\![\lambda x : T\,.\,M]\!]_\rho & = & (v \in |[\![T]\!]_\rho| \mapsto [\![M]\!]_{\rho,x\leftarrow v}) \\ [\![\Box_i]\!]_\rho & = & \mathcal{U}'_i & & [\![MN]\!]_\rho & = & [\![M]\!]_\rho([\![N]\!]_\rho) \end{array}$$

La fonction d'interprétation est étendue aux contextes en posant

$$[\![\Gamma]\!] \quad = \quad \left\{\rho \in \mathrm{Val}_{\mathcal{M}} \mid \forall (x : T) \in \Gamma \rho(x) \in |[\![T]\!]_\rho|\right\}$$

Correction Les critères de correction (vis-à-vis du modèle $\mathcal{M}$) pour les jugements de typage, de sous-typage et d'égalité sont les suivants :

1. Un jugement de typage $\Gamma \vdash M : T$ est correct si pour tout $\rho \in [\![\Gamma]\!]$:
   - Les dénotations $[\![M]\!]_\rho$ et $[\![T]\!]_\rho$ sont définies ;
   - $[\![T]\!]_\rho$ est un Π-ensemble ; et
   - $[\![M]\!]_\rho \in |[\![T]\!]_\rho|$.
2. Un jugement de sous-typage $\Gamma \vdash T \leq T'$ est correct si pour tout $\rho \in [\![\Gamma]\!]$:
   - Les dénotations $[\![T]\!]_\rho$ et $[\![T']\!]_\rho$ sont définies ;
   - $[\![T]\!]_\rho$ et $[\![T']\!]_\rho$ sont des Π-ensembles ;
   - $[\![T]\!]_\rho \leq [\![T']\!]_\rho$.

[29] Sans ce changement de représentation des fonctions (qui perd l'information du domaine) l'ensemble des propositions ne serait techniquement pas clos par produit dépendant indicé par un Π-ensemble arbitraire.

3. Un jugement d'égalité $\Gamma \vdash M = M' : T$ est correct si pour tout $\rho \in [\![\Gamma]\!]$:
   - Les dénotations $[\![M]\!]_\rho$, $[\![M']\!]_\rho$ et $[\![T]\!]_\rho$ sont définies ;
   - $[\![T]\!]_\rho$ est un Π-ensemble ; et
   - $[\![M]\!]_\rho = [\![M']\!]_\rho \in |[\![T']\!]_\rho|$.

Proposition 36 (Correction) — *Tout jugement dérivable suivant les règles de la Fig. 4.4 est correct au sens défini ci-dessus.*

4.3.4 Adéquation

Le schéma d'extraction de base À chaque terme (non typé) de $\mathrm{CC}^{\mathrm{irr}}_\omega$ on associe un terme M^* de λ_c défini récursivement par :

$$x^* = x \qquad s^* = (\Pi x : T \,.\, U)^* = \text{quasi-preuve quelconque}$$
$$(\lambda x : T \,.\, M)^* = \lambda x \,.\, M^* \qquad (MN)^* = M^* N^*$$

Intuitivement, la fonction d'extraction $M \mapsto M^*$ efface tout ce qui ne contribue pas au calcul dans les termes : les types sont effacés et remplacés par une quasi-preuve quelconque, qui n'a pas vocation à être exécutée.

Substitutions Une *substitution* est une suite finie $\sigma = [x_1 := u_1; \ldots; x_n : u_n]$ où $x_1, \ldots, x_n$ sont des variables deux à deux distinctes et où $u_1, \ldots, u_n$ sont des λ_c-termes quelconque. On dit qu'une substitution σ *réalise* une valuation ρ dans un contexte Γ (notation : $\sigma \Vdash_\Gamma \rho$) si $\mathrm{dom}(\sigma) = \mathrm{dom}(\Gamma)$ et si pour tout $(x : T) \in \Gamma$ on a $\sigma(x) \Vdash_{[\![T]\!]_\rho} \rho(x)$.

Adéquation Nous pouvons maintenant étendre la propriété d'adéquation de la section 4.1.5 (Prop. 26) à $\mathrm{CC}^{\mathrm{irr}}_\omega$ de la manière suivante :

Proposition 37 (Adéquation) — *Si $\Gamma \vdash M : T$, alors pour toute valuation $\rho \in [\![\Gamma]\!]$ et pour toute substitution σ telle que $\sigma \Vdash_\Gamma \rho$, on a*

$$M^*[\sigma] \;\Vdash_{[\![T]\!]_\rho}\; [\![M]\!]_\rho \,.$$

4.3.5 Extension du formalisme par ajout de constantes

Le modèle de réalisabilité que nous venons de présenter ne concerne pour l'instant qu'un formalisme purement intuitionniste. Il contient néanmoins tout le matériel nécessaire pour interpréter les principes de raisonnement classiques et les types inductifs, qu'on introduit dans le système de base ($\mathrm{CC}^{\mathrm{irr}}_\omega$) sous la forme de constantes typées éventuellement accompagnées d'une théorie équationnelle.

Supposons qu'on désire étendre le système avec une constante fraîche $c : T$ dont le type T ne mentionne que des constantes déjà introduites auparavant. La constante c peut éventuellement venir avec des règles d'inférence pour le jugement d'égalité qui décrivent la théorie équationnelle de cette constante. Pour interpréter le formalisme étendu avec la constante c dans le modèle de réalisabilité, on procède en deux temps :

1. D'abord on choisit dans le support du Π-ensemble $[\![T]\!]$ une dénotation $v \in |[\![T]\!]|$ satisfaisant la théorie équationnelle de c, et on pose $[\![c]\!] = v$. Ce qui permet d'étendre le résultat de correction (Prop. 36) au formalisme étendu avec la constante $c : T$. Dans le cas où T est une proposition (la constante c correspondant à un axiome), il suffit de poser $[\![c]\!] = \bullet$.

2. Ensuite on choisit une quasi-preuve $t \Vdash_{[\![T]\!]} v$ (en supposant qu'il en existe une) et on étend le schéma d'extraction avec $c^* \equiv t$. Ce qui permet d'étendre le résultat d'adéquation (Prop. 37) au formalisme étendu avec la constante $c : T$.

La loi de Peirce La constante

$$\mathsf{peirce} \ : \ \Pi A, B : * \,.\, ((A \to B) \to A) \to A$$

(sans théorie équationnelle) correspondant à l'axiome de Peirce s'interprète dans le modèle de réalisabilité classique par $[\![\mathsf{peirce}]\!] = \bullet$ et $\mathsf{peirce}^* \equiv \lambda\_ \,.\, \lambda\_ \,.\, \mathfrak{cc}$. Les deux abstractions effaçantes sont ici nécessaires pour garantir l'adéquation du λ_c-terme peirce^* vis-à-vis de l'interprétation du produit dépendant dans le modèle de réalisabilité.

Les entiers de Peano Les entiers de Peano sont introduits dans $\mathrm{CC}^{\mathrm{irr}}_{\omega}$ au moyen de trois constantes $\mathsf{nat} : \Box_1$, $\mathsf{0} : \mathsf{nat}$ et $\mathsf{s} : \mathsf{nat} \to \mathsf{nat}$. La constante nat est interprétée dans le modèle de réalisabilité par le Π-ensemble $[\![\mathsf{nat}]\!]$ défini par

$$|[\![\mathsf{nat}]\!]| = \mathbb{N} \qquad \text{et} \qquad n^{\perp_{[\![\mathsf{nat}]\!]}} = \|\mathsf{Nat}(n)\| \qquad (n \in \mathbb{N})$$

où $\|\mathsf{Nat}(n)\|$ désigne ici la valeur de fausseté de la formule $\mathsf{Nat}(n)$ au sens de la réalisabilité classique en logique du second ordre.

Les constantes $\mathsf{0}$ et s sont interprétées de la manière évidente, et la fonction d'extraction est étendue en posant :

$$\mathsf{0}^* \ \equiv \ \bar{0} \ \equiv \ \lambda xy \,.\, x \qquad \text{et} \qquad \mathsf{s}^* \ \equiv \ \bar{s} \ \equiv \ \lambda nxy \,.\, y(nxy)$$

(la constante de type nat étant traduite par n'importe quelle quasi-preuve). Là encore, cette extension est complète et adéquate au sens des propositions 36 et 37. Pour que la définition des entiers naturels soit complète, il faut également introduire les récurseurs

$$\Pi X : \mathsf{nat} \to \Box_i \,.\, X\mathsf{0} \to (\Pi y : \mathsf{nat} \,.\, X\, y \to X\,(\mathsf{s}\, y)) \to \Pi x : \mathsf{nat} \,.\, X\, x \qquad (i \geq 0)$$

(en notant $\Box_0 \equiv *$) accompagnés de leurs égalités définitionnelles. Ces constantes s'interprètent sans difficulté dans le modèle de réalisabilité classique de $\mathrm{CC}^{\mathrm{irr}}_{\omega}$. (On se reportera à [77] pour les détails.)

Il est important de souligner que l'interprétation des entiers que nous venons de donner n'est pas la seule possible (du moins en ce qui concerne la composante « réalisabilité »). On peut remplacer (au niveau de l'extraction) les entiers de Krivine par les entiers paresseux introduits à la section 4.2.5 en posant

$$|[\![\mathsf{nat}]\!]| = \mathbb{N} \qquad \text{et} \qquad n^{\perp_{[\![\mathsf{nat}]\!]}} = \|\mathsf{Nat}'(n)\| \qquad (n \in \mathbb{N})$$

où $\mathsf{Nat}'(n) \equiv \forall Z \,((\![n] \Rightarrow Z) \Rightarrow Z)$. L'interprétation des constantes $\mathsf{0}$ et s reste la même, mais le schéma d'extraction est modifié en posant :

$$\mathsf{0}^* \ \equiv \ \lambda x \,.\, x\,\hat{0} \qquad \text{et} \qquad \mathsf{s}^* \ \equiv \ \lambda n \,.\, n\,\check{s}$$

où $\check{s}$ est l'instruction associée à la fonction successeur (voir section 4.2.5). Avec ce nouveau schéma d'extraction, les entiers de Peano de la théorie des types sont tout simplement remplacés par des entiers paresseux au cours de l'extraction. C'est ce schéma d'extraction qui a été retenu dans le module d'extraction classique que nous présenterons à la section 4.4.

4.3.6 Le fragment commun

Pour relier le modèle de réalisabilité classique de Krivine avec celui que nous venons de présenter pour le calcul des constructions, il est nécessaire d'isoler un fragment commun, c'est-à-dire une classe de formules qui s'expriment indifféremment dans les deux systèmes. Ce fragment est le suivant :

		PA2	$\mathrm{CC}^{\mathrm{irr}}_{\omega} + \mathsf{nat}$
A, B	$::=$	$X(e_1, \ldots, e_k)$	$X\, e_1 \cdots e_k$
	$\mid$	$A \Rightarrow B$	$A \to B$
	$\mid$	$\forall x\, (\mathsf{Nat}(x) \Rightarrow A)$	$\Pi x : \mathsf{nat}\,.\, A$
	$\mid$	$\forall X\, (\top \Rightarrow A)$	$\Pi X : (\mathsf{nat}^k \to *)\,.\, A$

On notera que le produit dépendant $\Pi x : \mathsf{Nat}\,.\, A$ est associé à la quantification numérique $\forall^{\mathsf{N}} x\, A$ de l'arithmétique du second ordre tandis que le produit dépendant $\Pi X : (\mathsf{nat}^k \to *)\,.\, A$ est associé à la quantification du second ordre préconditionnée par la formule trivialement vraie : $\forall X\, (\top \Rightarrow A)$, afin de respecter l'arité imposée par l'interprétation du produit dépendant.

En supposant que les entiers de Peano de la théorie des types $\mathrm{CC}^{\mathrm{irr}}_{\omega} + \mathsf{nat}$ sont interprétés par des entiers de Krivine (suivant la première méthode d'interprétation que nous avons présentée), on démontre alors que :

Proposition 38 (Conservativité) — *Les deux interprétations coïncident sur le fragment commun défini ci-dessus, en ce sens que pour toute formule close A qui appartient à ce fragment on a* $[\![A]\!]^{\perp}_{\mathrm{CC}^{\mathrm{irr}}_{\omega}} = \|A\|_{\mathrm{PA2}}$.

Preuve. Par induction sur la formule A, en étendant l'égalité ci-dessus aux formules ouvertes et en utilisant des valuations appropriées. □

(Il est possible d'étendre la Prop. 38 à toutes les formules de la logique du second ordre en enrichissant le langage de $\mathrm{CC}^{\mathrm{irr}}_{\omega}$ avec des constructions supplémentaires dont le détail est donné dans [77].)

En pratique, ce résultat de conservativité sémantique[30] permet d'importer tous les résultats de réalisabilité classique en arithmétique du second ordre (notamment ceux qui concernent la réalisation de l'axiome du choix) dans le modèle de réalisabilité classique du calcul des constructions.

4.4 Extraction classique en Coq

Grâce aux idées développées dans la section 4.3, j'ai développé un module d'extraction classique en Coq[31], qui permet d'extraire des λ_c-termes à partir de preuves effectuées en Coq. Ce module met aussi en œuvre la méthode d'extraction de témoin décrite à la section 4.2.3 dans un cadre un peu plus général.

L'extraction classique en Coq repose sur deux composants logiciels distincts écrits en Objective Caml (v. 3.11 [34]) :

[30] Qui n'est pas un résultat de conservativité syntaxique, dans la mesure où il est clair que le calcul des constructions avec univers (et entiers de Peano) est très loin de constituer une extension conservative de l'arithmétique du second ordre.

[31] `http://perso.ens-lyon.fr/alexandre.miquel/kextraction/kextraction.html`

Le module `kextraction` Il s'agit du module d'extraction classique proprement dit, qui ajoute au système Coq un ensemble de commandes dédiées à l'extraction classique. Ce module est fourni sous la forme d'un greffon qu'on peut charger dynamiquement dans le *toplevel* de Coq en utilisant les possibilités de chargement dynamique en mode natif qui sont présentes dans Coq (depuis la v8.2 [98]) et dans Caml (depuis la v3.11 [34]).

L'évaluateur `jivaro` Il s'agit de l'interpréteur[32] qui permet d'exécuter et de tracer les programmes extraits. Cet interpréteur (basé sur la réduction de tête) traite toutes les constructions du langage λ_c et dispose de primitives d'affichage et de calcul sur les entiers et les chaînes de caractères[33].

Le module d'extraction classique est basé sur une extension du schéma d'extraction classique défini à la section 4.3.4 à toutes les constructions de Coq : les types (co)inductifs et leurs constructeurs, les définitions (co)récursives, les définitions et les axiomes. Comme le cadre théorique qui permet de justifier toutes ces extensions n'a pas encore été entièrement écrit, l'extracteur classique doit être considéré pour l'instant comme un prototype au stade expérimental.

4.4.1 Les commandes d'extraction

Le module d'extraction classique enrichit le système Coq avec des commandes permettant d'extraire le contenu calculatoire (au sens de la réalisabilité classique) des preuves effectuées en Coq. Le résultat de l'extraction est stocké dans un fichier-objet λ_c (de suffixe `.lco`, pour *lambda-calculus object*) dans lequel sont déclarés un certain nombre de symboles (c'est-à-dire : des instructions au sens de λ_c) accompagnés de leur définition. Ce fichier-objet peut ensuite être directement utilisé par l'interpréteur `jivaro`.

L'extraction de témoin est effectuée par la commande

`Classical Extract Witness` *exproof* `using` *decproof* `.`

où *exproof* et *decproof* sont des constantes Coq dont le type est de la forme

$$\begin{array}{lcl} \textit{exproof} & : & \texttt{forall}\ \vec{x} : \vec{T}\texttt{,}\ \texttt{exists}\ y : U\texttt{,}\ P\,\vec{x}\,y \\ \textit{decproof} & : & \texttt{forall}\ \vec{x} : \vec{T}\texttt{,}\ \texttt{forall}\ y : U\texttt{,}\ \texttt{\{}P\,\vec{x}\,y\texttt{\}+\{}\tilde{}P\,\vec{x}\,y\texttt{\}} \end{array}$$

Le résultat est stocké dans le fichier-objet produit par l'extracteur classique sous la forme d'une instruction nommée *exproof*`%witness`, qui attend autant d'arguments que de paramètres $\vec{x} : \vec{T}$ et retourne la représentation (dans λ_c) d'un objet de type U satisfaisant la propriété P.

Remarque On notera que la preuve de décidabilité du prédicat P doit être fournie sous la forme d'un objet de type `forall` $y : U$`, {`$P\,\vec{x}\,y$`}+{`$\tilde{}P\,\vec{x}\,y$`}`, qui en Coq désigne le type des fonctions de décision du prédicat P (ici paramétré par les objets $\vec{x} : \vec{T}$). Dans le modèle de réalisabilité décrit à la section 4.3, il est facile d'interpréter l'union disjointe $A + B$ dans les univers prédicatifs $\Box_i$, et on

[32] `http://perso.ens-lyon.fr/alexandre.miquel/jivaro/jivaro.html`

[33] Dans λ_c, les entiers (relatifs) et les chaînes de caractères sont traités exactement de la même manière que les entiers (naturels) primitifs décrits à la section 4.2.5. Dans l'évaluateur `jivaro`, les entiers sont représentés à l'aide de la librairie `Big_int` de Caml.

vérifie aisément que celle-ci conserve un caractère intuitionniste, contrairement à la disjonction $A \vee B$ au niveau propositionnel.[34]

4.4.2 Le schéma de base

D'un point de vue technique, le schéma d'extraction utilisé par le module d'extraction classique est une projection des termes de pCIC[35] sur le λ_c-calcul où tous les types (et les prédicats) sont écrasés sur une constante `.type` sans contenu calculatoire. (Rappelons que de tels objets ne sont pas censés arriver en position de tête lors de l'exécution des termes extraits.)

La traduction $M \mapsto M^*$ de pCIC vers λ_c est définie sur le PTS sous-jacent par les équations

$$\begin{array}{lcl}
x^* & \equiv & x \\
(\texttt{fun}\ (x : T)\ \Rightarrow\ M)^* & \equiv & \lambda x\,.\,M^* \\
(M\ N)^* & \equiv & M^*\ N^* \\
(\texttt{forall}\ (x : T),\ U)^* & \equiv & \texttt{.type} \\
(\texttt{Prop})^* & \equiv & \texttt{.type} \\
(\texttt{Set})^* & \equiv & \texttt{.type} \\
(\texttt{Type})^* & \equiv & \texttt{.type}
\end{array}$$

Contrairement au mécanisme d'extraction constructive tel qu'il est implémenté en Coq [65], l'extraction classique traite les termes de sorte `Prop` de la même manière que les termes de sorte `Set` ou `Type`.

Objets définis L'environnement Coq permet à l'utilisateur d'introduire des constantes pour représenter les familles de types définies (co)inductivement, les constructeurs, les définitions et les axiomes. Chaque constante c de pCIC est traduite en une instruction du langage λ_c avec le même nom long. Par exemple, le constructeur `S` dans la définition inductive du type `nat` est traduite en une instruction nommée `Coq.Init.Datatypes.S`.

Seule exception à ce principe, les constantes I représentant des types ou des familles de types définies (co)inductivement sont traduites en l'instruction `.type`, comme n'importe quel type : $I^* \equiv$ `.type`.

[34] De manière générale, le modèle de réalisabilité classique étendu le calcul des constructions confine la logique classique au seul niveau propositionnel. On peut démontrer que le type exprimant la loi de Peirce *dans les univers prédicatifs* n'est réalisé par aucune quasi-preuve.

[35] Ici, pCIC désigne la version du calcul des constructions inductives (CIC) qui est actuellement utilisée dans le système Coq. Elle diffère de la présentation originelle [86] par le fait que la sorte `Set` est désormais prédicative, au même titre que les sortes $\texttt{Type}_i$ (pour $i \geq 1$). On peut donc voir la sorte `Set` comme un alias pour le premier univers prédicatif. Ce changement de statut est lié à la découverte en 2002 [18] d'une incompatibilité profonde entre l'imprédicativité de `Set` et la combinaison du tiers-exclu et de l'axiome du choix, même lorsque ces deux axiomes sont exprimés dans la sorte `Prop`. Pour restaurer la compatibilité avec les mathématiques « habituelles » (qui font un grand usage des deux principes), on a donc dû renoncer à l'imprédicativité de la sorte `Set`. (Le problème ne se pose pas pour l'autre sorte imprédicative, la sorte `Prop`, en raison de son caractère *proof-irrelevant*.)

Il est intéressant de mentionner que la construction de modèle présentée à la section 4.3 ne pourrait sans doute pas être adaptée à la version originelle du calcul des constructions inductives, à moins de renoncer définitivement à l'axiome du choix. (Un axiome que la réalisabilité sait par ailleurs traiter, du moins en ce qui concerne ses formes les plus faibles [61].) De fait, la *proof-irrelevance* semble être la seule manière raisonnable d'interpréter l'imprédicativité dans le cadre de la réalisabilité classique.

Filtrage Le filtrage d'un terme M appartenant à une famille I définie (co)inductivement à partir d'un ensemble de constructeurs $c_1, \ldots, c_n$ d'arités $k_1, \ldots k_n$ (en excluant les paramètres) est traduite de la manière suivante :

$$\left(\begin{array}{l}\texttt{match } M \texttt{ with}\\ \quad\vdots\\ \texttt{end}\end{array}\right)^* \equiv \begin{array}{l} I\texttt{\%case } M^* \ (\lambda x_1 \cdots x_{k_1} \,.\, N_1^*)\\ \qquad\vdots\\ \qquad\qquad (\lambda x_1 \cdots x_{k_n} \,.\, N_n^*)\,,\end{array}$$

où I`%case` est un combinateur de filtrage associé à la famille de types I, dont la définition dépendra de la représentation des données de type I dans le langage λ_c (voir sections 4.4.3 et 4.4.4). Le prédicat d'élimination et les paramètres associés à la construction `match` sont ignorés au cours de l'extraction.

Points fixes et co-points fixes Les constructions `fix` et `cofix` sont traduites de la même manière qu'en λ-calcul, sans tenir compte des contraintes de décroissance structurelle (pour les points fixes) ou de production (pour les co-points fixes).[36] Formellement, la traduction est définie par

$$\left(\begin{array}{l}(\texttt{co})\texttt{fix } f_1 : T_1 := M_1\\ \qquad\vdots\\ \texttt{with } f_n : T_n := M_n \texttt{ for } f_i\end{array}\right)^* \equiv \begin{array}{l}\texttt{.fix\_}n\texttt{\_}i \ (\lambda f_1 \cdots f_n \,.\, M_1^*)\\ \qquad\vdots\\ \qquad\qquad (\lambda f_1 \cdots f_n \,.\, M_n^*)\end{array}$$

où `.fix_`n`_`i (pour $1 \leq i \leq n$) est un combinateur de point fixe défini par la règle de réduction

$$\texttt{.fix\_}n\texttt{\_}i \ F_1 \ \cdots \ F_n \ \succ \ F_i \ (\texttt{.fix\_}n\texttt{\_1} \ F_1 \cdots F_n) \ \cdots \ (\texttt{.fix\_}n\texttt{\_}n \ F_1 \cdots F_n)$$

Au cours de l'extraction, ces combinateurs de point fixe sont automatiquement engendrés à la volée, et leur définition est stockée dans le fichier-objet produit.

4.4.3 Représentation par défaut des (co)inductifs

Dans le langage λ_c, la représentation des structures de données appartenant à une famille I de types (co)inductifs est déterminée par l'implémentation des instructions $c_1, \ldots, c_k$ associées aux constructeurs de I et par l'implémentation du combinateur de filtrage I`%case` associée à cette famille. Dans la mesure où les structures de données appartenant à la famille de types I ne sont manipulées dans le langage cible qu'à travers les instructions mentionnées ci-dessus, il suffit de changer l'implémentation de ces instructions pour changer la représentation des données de type I dans tout le code extrait.

Par défaut, les structures de données de pCIC sont projetées sur le λ-calcul pur suivant les codages imprédicatifs standard. Formellement, si I désigne une famille de types (co)inductifs dépendant de p paramètres et définie à partir de n constructeurs $c_1, \ldots, c_n$ d'arités $k_1, \ldots, k_n$ (en excluant les paramètres) alors, par défaut, chaque instruction c_i ($1 \leq i \leq n$) associée au constructeur de même nom est définie dans le code extrait comme un alias pour le λ-terme

$$c_i \quad := \quad \lambda \underbrace{y_1 \cdots y_p}_{\text{paramètres}} \underbrace{x_1 \cdots x_{k_i}}_{\text{arg. réels}} \underbrace{e_1 \cdots e_n}_{\text{éliminateurs}} \,.\, e_i \, x_1 \cdots x_{k_i} \,.$$

[36] Ces contraintes ne se posent pas quand on travaille dans un langage en appel par nom.

Suivant cette représentation, le filtrage d'un objet de type I consiste à appliquer cet objet à toutes les branches de filtrage, de telle sorte que le combinateur de filtrage I`%case` associé à la famille I est simplement défini comme un alias pour la fonction identité :

$$I\texttt{\%case} \quad := \quad \lambda z\,.\,z\,.$$

4.4.4 Le mécanisme de remplacement

Pour certains types de données, il est préférable de remplacer la représentation par défaut (suivant le schéma décrit à la section 4.4.3) par une représentation plus efficace. L'exemple typique est celui du type `nat` dont la représentation par défaut à l'aide des instructions

$$\begin{array}{lcl}
\texttt{Coq.Init.Datatypes.O} & := & \lambda e_0 e_1\,.\,e_0 \\
\texttt{Coq.Init.Datatypes.S} & := & \lambda x e_0 e_1\,.\,e_1\,x \\
\texttt{Coq.Init.Datatypes.nat\%case} & := & \lambda z\,.\,z
\end{array}$$

est au moins aussi inefficace que le codage traditionnel de Church.[37]

Pour ce faire, le module d'extraction classique introduit un ensemble auxiliaire de λ_c-définitions destinées à primer sur les définitions qui seraient normalement produites au cours de l'extraction. Ces λ_c-définitions sont stockées dans un fichier-objet `override.lco` qui est automatiquement chargé à la première invocation d'une commande d'extraction classique. Avant de produire une nouvelle définition (correspondant à un constructeur, un combinateur de filtrage, une constante, une définition, un axiome, etc.) l'extracteur examine d'abord la liste des définitions du fichier « `override.lco` » pour voir si l'instruction correspondante y est définie. Si c'est le cas, c'est cette définition qui est placée dans le fichier-objet produit ; sinon, on utilise la définition par défaut.

Représentation efficace des entiers naturels Le fichier « `override.lco` » qui accompagne le module d'extraction classique pour Coq redéfinit la représentation des entiers naturels en posant :

$$\begin{array}{lcl}
\texttt{Coq.Init.Datatypes.O} & := & \lambda k\,.\,k\ \texttt{0} \\
\texttt{Coq.Init.Datatypes.S} & := & \lambda n k\,.\,n\ (\lambda n\,.\,\texttt{int_succ}\ n\ k) \\
\texttt{Coq.Init.Datatypes.nat\%case} & := & \lambda z e_0 e_1\,.\,z\ \lambda n\,.\,\texttt{int_null}\ n \\
 & & e_0\ (\texttt{int_pred}\ n\ (\lambda p\,.\,e_1\ (\lambda k\,.\,k\ p)))
\end{array}$$

Avec ces définitions, chaque entier naturel est représenté dans le code extrait comme un entier paresseux au sens qui a été défini à la section 4.2.5.

Pour accompagner ce gain en termes de complexité spatiale d'un gain en termes de complexité temporelle, le fichier « `override.lco` » redéfinit également les fonctions arithmétiques définies dans la librairie standard : l'addition

$$\texttt{Coq.Init.Peano.plus} \quad := \quad \lambda n m k\,.\,n\,(\lambda n\,.\,m\,(\lambda m\,.\,\texttt{int\_plus}\,n\,m\,k))$$

la multiplication, la fonction prédécesseur et la soustraction.

[37] Sauf en ce qui concerne le calcul du prédécesseur, qui se fait en temps constant.

Extraction des axiomes Par défaut, chaque axiome de pCIC est extrait sous la forme d'une instruction (avec le même nom long) déclarée dans le fichier-objet produit comme un symbole non défini (comme pour l'instruction `.type`). Là encore, il est possible de passer outre ce comportement par défaut en redéfinissant l'instruction correspondante dans le fichier « `override.lco` ».

Ainsi, le fichier distribué avec l'extracteur classique contient une redéfinition de l'instruction associée au principe du tiers-exclu (déclaré comme un axiome dans la librairie `Classical` de Coq) mais aussi des instructions associées à divers résultats classiques qui sont démontrés dans cette même librairie à partir du principe du tiers-exclu (notamment la loi de Peirce).

Réalisateurs de remplacement pour la librairie standard L'extraction d'une preuve d'un théorème formalisé en Coq déclenche automatiquement (et de manière récursive) l'extraction de toutes les preuves des lemmes utilisés dans la preuve du théorème. Parmi ces lemmes, on trouve de nombreux lemmes démontrés dans la librairie standard pour lesquels on dispose de réalisateurs bien plus efficaces que ceux qui sont extraits à partir de la preuve.

Pour améliorer l'efficacité du code extrait, le fichier « `override.lco` » propose également un certain nombre de réalisateurs simples en remplacement des réalisateurs qui seraient produits par l'extracteur à partir des démonstrations effectuées dans la librairie standard. Parmi ces lemmes, on peut citer l'exemple de la commutativité de l'addition dans $\mathbb{N}$

```
forall (n m : nat), n + m = m + n,
```

dont la preuve, qui repose sur trois récurrences sur les entiers, produit un réalisateur assez gros et très inefficace. En remplacement de ce réalisateur, le fichier « `override.lco` » propose le réalisateur $\lambda nmz\,.\,z$, qui a néanmoins le même comportement calculatoire que le réalisateur produit par défaut.

(Les programmes extraits reposent donc fortement sur cette combinaison de réalisateurs construits à la main et de réalisateurs issus de termes bien typés que nous avions évoquée à la section 1.2.2.)

Chapitre 5

Perspectives

Les sujets abordés dans les chapitres qui précèdent reflètent dans un ordre quasi-chronologique l'essentiel de mon parcours scientifique depuis la fin de ma thèse.[1] Mon programme de recherche initial, qui concernait les liens entre la théorie des ensembles et la théorie des types (chapitre 2), est aujourd'hui pratiquement achevé.[2] Au cours des deux dernières années, j'ai progressivement réorienté toute mon activité de recherche vers la réalisabilité classique. Il s'agit d'un sujet difficile dont je commence tout juste à comprendre les enjeux ; mais plus j'avance dans sa compréhension, et plus il me semble clair que le sujet est très vaste et extrêmement prometteur. C'est donc dans cette direction que je compte poursuivre ma recherche dans les années qui viennent, en suivant un certain nombre de pistes que je vais maintenant tâcher de présenter.

5.1 Extensions de la réalisabilité classique

5.1.1 Réalisabilité et forcing

La principale limitation de la notion de modèle qui nous a été léguée par une longue tradition qui remonte à Tarski (c'est-à-dire : les modèles à valeurs dans

[1] Afin de ne pas surcharger ce mémoire plus qu'il ne l'est déjà, j'ai mis de côté certains travaux qui ne s'inscrivaient pas directement dans le fil conducteur suggéré par le titre. Je n'ai parlé ni de mes travaux avec Stéphane Lengrand [64] autour du système $F\omega$ classique (que je vois aujourd'hui comme un « ballon d'essai » pour aborder la correspondance de Curry-Howard en logique classique à travers les λ-calculs non déterministes), ni de mes travaux avec Germain Faure sur la sémantique catégorique du λ-calcul parallèle.

[2] Seule exception : la force théorique *exacte* du calcul des constructions inductives (Coq), qui reste toujours pour moi une inconnue. Cette question me semble intimement liée à la conjecture de Friedman et Ščedrov [38], à savoir : le système IZF_R (cf section 2.2) est-il strictement plus faible que le système IZF_C ? Je conjecture à mon tour que si la conjecture de Friedman et Ščedrov [38] est vraie, c'est-à-dire si $\mathrm{IZF}_R < \mathrm{IZF}_C \approx \mathrm{ZF}$, alors la puissance théorique de Coq est strictement plus faible que celle de la théorie des ensembles de Zermelo-Fraenkel. Mon intuition est la suivante : le modèle ensembliste de Coq peut être entièrement décrit dans $\mathrm{IZF}_R^{<\omega}$, c'est-à-dire IZF_R étendu avec une hiérarchie infinie de IZF_R-univers emboîtés (une notion qui reste à définir formellement). Si $\mathrm{IZF}_R < \mathrm{ZF}$, c'est-à-dire : si ZF peut montrer l'existence d'un modèle de Kripke de IZF_R, on peut espérer itérer la construction pour obtenir un modèle de Kripke de $\mathrm{IZF}_R^{<\omega}$ dans ZF. D'où l'on pourrait tirer que $\mathrm{Coq} \leq \mathrm{IZF}_R^{<\omega} < \mathrm{ZF}$. On notera ici l'importance de raisonner en termes de modèles de Kripke (ou toute autre forme de modèle complète pour la logique intuitionniste) quand on travaille avec IZF_R, dans la mesure où la version classique de IZF_R est équiconsistante à ZF.

l'algèbre de Boole à deux points) vient du fait qu'elle convoie une information beaucoup trop faible sur les formules qu'elle se propose d'interpréter, c'est-à-dire une information de type « tout ou rien » (vrai/faux). Dans cette perspective, l'introduction par Cohen de la technique du *forcing* (à l'origine pour démontrer l'indépendance de l'hypothèse du continu [23, 24][3]) a constitué une avancée majeure, puisqu'elle permettait enfin de convoyer à travers les « tuyaux » de la logique une information considérablement plus riche que le vrai/faux, à savoir : un ensemble (généralement infini) de conditions de forcing. La technique du forcing a été considérablement affinée depuis, et elle constitue aujourd'hui l'un des outils de base pour construire des modèles de la théorie des ensembles. Le forcing s'est par ailleurs rapproché de la théorie des modèles usuelle à travers la notion de modèle booléen (développée peu après l'introduction du forcing par Scott, Solovay et Vopěnska), qui est une généralisation très naturelle de la notion de modèle dans laquelle les formules sont interprétées non pas par 0 et 1, mais par les éléments d'une algèbre de Boole arbitraire.

Plusieurs auteurs (notamment Scott et Krivine) ont remarqué la ressemblance frappante entre le forcing (où une formule est interprétée par un ensemble de conditions) et la réalisabilité de Kleene (où une formule est interprétée par un ensemble de programmes), l'application de programmes jouant dans le cadre de la réalisabilité le même rôle que le produit de conditions dans le cadre du forcing. De fait, le forcing a eu une influence significative sur le développement ultérieur de la théorie de la réalisabilité (intuitionniste), et les premiers modèles de réalisabilité de la théorie des ensembles intuitionniste [82, 37, 70] en sont directement inspirés, de même que la théorie des *topoi*.

Cependant, la réalisabilité a été longtemps handicapée par son confinement à la logique intuitionniste, ce qui explique sans doute pourquoi elle n'a eu qu'une faible influence sur le forcing. De ce point de vue, l'introduction de la théorie de la réalisabilité classique (dont le développement est depuis le début très influencé par le forcing) constitue un progrès majeur, qui permet enfin à la théorie de la réalisabilité de se mesurer pleinement au forcing.

Dans cette confrontation des deux approches, la réalisabilité dispose pourtant d'un atout majeur qui apparaît déjà dans le cas intuitionniste : son interprétation des constructions de la logique (suivant la correspondance de Curry-Howard) distingue très nettement les constructions uniformes des constructions non uniformes correspondantes (l'union et la disjonction, l'intersection et la conjonction, etc.), contrairement au forcing qui les identifie (par exemple, en interprétant la quantification universelle de la même manière qu'une conjonction infinitaire). Or il est important de rappeler qu'en théorie de la démonstration, la quantification universelle (non bornée) ne peut pas être assimilée à une conjonction infinitaire. Cela apparaît déjà dans sa règle d'introduction — qui ne comporte qu'une seule prémisse — mais aussi dans le premier théorème d'incomplétude de Gödel, qui montre précisément l'existence de formules $A(x)$ dont toutes les instances $A(n)$ sont prouvables (lorsque $n \in \mathbb{N}$), tandis que la clôture universelle $\forall x\, A(x)$ ne l'est pas. En théorie de la démonstration, la quantification universelle (non bornée) est bien plus proche d'une intersection infinitaire que d'une conjonction infinitaire.[4]

[3] Et plus précisément pour établir la non prouvabilité de l'hypothèse du continu dans ZF. La non contradiction avait déjà été établie par Gödel avec les ensembles constructibles.

[4] Cette caractérisation ne concerne pas la quantification universelle bornée (qui inclut la quantification universelle numérique $\forall^{\mathsf{N}} x\, A(x)$ dans PA2) qui est un mélange d'intersection

L'un des enjeux actuels de la réalisabilité classique est de développer des extensions de la réalisabilité suffisamment générales pour englober à la fois la réalisabilité classique et le forcing.[5] Krivine a récemment proposé [63] une notion de *structure de réalisabilité* très générale permettant de combiner la réalisabilité et le forcing dans l'esprit du forcing itéré. Dans la mesure où les travaux de Krivine laissaient nettement entrevoir un lien entre le forcing et les effets de bord, j'ai moi-même défini il y a quelques mois une notion de *réalisabilité classique avec états* afin de rendre plus explicite le lien entre le forcing et la programmation impérative. (On peut voir la réalisabilité avec états comme une forme extrêmement simplifiée des structures de réalisabilité de Krivine.)

5.1.2 La réalisabilité avec états

La réalisabilité classique avec états [78] est une extension de la réalisabilité classique dans laquelle les processus ne sont plus des entités binaires, mais des entités ternaires de la forme

Processus $\qquad p \quad ::= \quad t \star \pi \star s \qquad (s \in \mathscr{S})$

où s est un *état* appartenant à un ensemble $\mathscr{S}$ non vide muni d'un préordre noté $\leq$. D'un point de vue logique, un état $s \in \mathscr{S}$ représente une *condition de forcing*, et la relation $s \leq s'$ exprime que s' est une condition *plus forte* que la condition s. (On utilise ici la relation d'ordre dans le sens inverse où elle est utilisée habituellement en forcing, suivant la convention de Shelah.)

Comme en réalisabilité classique ordinaire (correspondant au cas où $\mathscr{S}$ est un singleton), on suppose que l'ensemble $\Lambda \star \Pi \star \mathscr{S}$ est muni d'une relation d'évaluation notée $p \succ p'$ comme précédemment. On suppose que la relation d'évaluation satisfait les axiomes

Grab	$\lambda x\,.\,t$	$\star$	$u \cdot \pi$	$\succ\!\!\!\succ$	$t\{x := u\}$	$\star$	π
Push	tu	$\star$	π	$\succ\!\!\!\succ$	t	$\star$	$u \cdot \pi$
Save	cc	$\star$	$t \cdot \pi$	$\succ\!\!\!\succ$	t	$\star$	$\mathsf{k}_\pi \cdot \pi$
Restore	k_π	$\star$	$t \cdot \pi'$	$\succ\!\!\!\succ$	t	$\star$	π

où $t \star \pi \succ\!\!\!\succ t' \star \pi'$ est une abréviation (sur les processus binaires) définie par :

$$t \star \pi \succ\!\!\!\succ t' \star \pi' \qquad \equiv \qquad \forall s \in \mathscr{S} \;\; \exists s' \geq s \quad t \star \pi \star s \succ^* t' \star \pi' \star s' \,.$$

Les quatre axiomes ci-dessus expriment que l'abstraction, l'application et les constantes cc et k_π s'évaluent comme en réalisabilité ordinaire, à cette différence près qu'elles peuvent à présent modifier l'état courant en le renforçant (ce qui inclut la possibilité de ne pas toucher à l'état courant). D'un point de vue informatique, il y a de nombreuses raisons de supposer que des opérations purement logiques puissent modifier l'état courant : par exemple pour y stocker

et de type flèche. La remarque vaut également pour le produit dépendant $\Pi x : A\,.\,B(x)$ en théorie des types, que l'on peut toujours décomposer en réalisabilité intuitionniste comme une intersection (infinitaire) de types flèche (voir [74, 77] par exemple).

[5] De telles extensions permettraient notamment de définir de nouveaux modèles de ZF (suivant les techniques de construction habituelles en réalisabilité et en forcing), avec l'espoir d'en tirer de nouveaux résultats d'indépendance. Rappelons que d'après un résultat de Schoenfield, les modèles construits par forcing ne peuvent pas établir l'indépendance de formules appartenant à l'une des classes Π^1_2 ou Σ^1_2. Un espoir suscité par le développement de modèles combinant la réalisabilité et le forcing est d'abaisser cette « barrière » de Schoenfield.

une trace des opérations effectuées, ou encore pour incrémenter un compteur contenant le nombre d'étapes d'évaluation effectuées depuis le début de l'exécution du processus. Par ailleurs, à l'instar des langages impératifs, la constante cc sauvegarde la pile courante, mais pas l'état courant. De même, la constante k_π ne se charge de restaurer que la pile π — tout en ayant la possibilité de modifier l'état courant (à condition de le renforcer).

On notera également que ces axiomes ne supposent pas que l'évaluation des constructions logiques (abstraction, application, cc et k_π) est atomique. En présence d'un état séparé des constituants purement fonctionnels du processus (le terme et la pile courante), il est tout-à-fait légitime de supposer — par exemple — qu'un processus de la forme $tu \star \pi \star s$ passe par un certain nombre d'étapes intermédiaires avant de s'évaluer sur $t \star u \cdot \pi \star s'$ (avec $s' \geq s$). Cette non-atomicité des opérations logiques permet d'envisager (par exemple) la mise en place d'un ordonnanceur dans lequel les processus dormants sont stockés dans l'état courant. (Ce point sera détaillé à la section 5.1.6.)

Il est utile de préciser qu'en dehors des opérations purement logiques (abstraction, application, cc, k_π), on ne suppose pas que l'état se renforce systématiquement au cours de l'évaluation, en ce sens que $t \star \pi \star s \succ t' \star \pi' \star s'$ n'implique pas nécessairement que $s \leq s'$. Même les opérations purement logiques (dont on ne suppose plus qu'elles sont atomiques) peuvent passer par des processus intermédiaires contenant des états arbitraires : la condition de renforcement ne porte que sur le processus d'arrivée, pas sur les processus intermédiaires.

Le modèle de réalisabilité avec états Comme en réalisabilité ordinaire, le modèle de réalisabilité avec états est paramétré par un ensemble de processus $\perp\!\!\!\perp \subseteq \Lambda \star \Pi \star \mathscr{S}$ clos par anti-évaluation. Dans ce cadre, une *valeur de fausseté* est donnée par un ensemble $S \subseteq \Pi \times \mathscr{S}$ quelconque. La valeur de vérité correspondante est donnée par l'ensemble $S^{\perp\!\!\!\perp} \subseteq \Lambda \times \mathscr{S}$ défini par

$$\mathscr{S}^{\perp\!\!\!\perp} \;=\; \{(t, s_1) \in \Lambda \times \mathscr{S} \;:\; \forall (\pi, s_2) \in S \;\; (t, s_1) \perp\!\!\!\perp (\pi, s_2)\}$$

où $(t, s_1) \perp\!\!\!\perp (\pi, s_2)$ est la relation définie par

$$(t, s_1) \perp\!\!\!\perp (\pi, s_2) \quad\equiv\quad \forall s \in \mathscr{S}\ (s \geq s_1,\ s \geq s_2 \;\Rightarrow\; t \star \pi \star s \in \perp\!\!\!\perp)\,.$$

On notera que l'orthogonal $S^{\perp\!\!\!\perp} \subseteq \Lambda \times \mathscr{S}$ d'une valeur de fausseté $S \subseteq \Pi \times \mathscr{S}$ (quelconque) est monotone par rapport à l'état, en ce sens que les conditions $(t, s) \in \mathscr{S}^{\perp\!\!\!\perp}$ et $s \leq s'$ entraînent que $(t, s') \in \mathscr{S}^{\perp\!\!\!\perp}$.

L'interprétation des formules de l'arithmétique du second ordre est définie de la même manière qu'auparavant (cf section 4.1.4), à cette différence près qu'on a à présent $\|A\| \subseteq \Pi \times \mathscr{S}$ et $|A| = \|A\|^{\perp\!\!\!\perp} \subseteq \Lambda \times \mathscr{S}$. Le seul changement notable apparaît dans l'interprétation de l'implication, qui devient :

$$\|A \Rightarrow B\| \;=\; \{(t \cdot \pi, s) \;:\; \exists s_0 \leq s\ ((t, s) \in |A| \;\wedge\; (\pi, s_0) \in \|B\|)\}\,.$$

Cas particuliers Le cas où $\mathscr{S}$ est un singleton correspond à la réalisabilité ordinaire (voir chapitre 4). Plus intéressant est le cas où $\perp\!\!\!\perp = \varnothing$, qui correspond très exactement au forcing au sens de Cohen. Dans ce cas particulier, la valeur de vérité d'une formule A est caractérisée par

$$|A| = \Lambda \times [\![A]\!]\,,$$

où $[\![A]\!]$ désigne la valeur de vérité de A dans l'algèbre de Boole engendrée par l'ensemble de conditions $(\mathscr{S}, \geq)$ [12, 59].

La situation est résumée dans la table ci-dessous :

	$\perp\!\!\!\perp \neq \varnothing$	$\perp\!\!\!\perp = \varnothing$
$\#\mathscr{S} > 1$	Réalisabilité avec états	Forcing de Cohen
$\#\mathscr{S} = 1$	Réalisabilité de Krivine	Modèle standard (de PA2)

Le lemme d'adéquation En réalisabilité avec états, on note $(t, s) \Vdash A$ si $(t, s) \in |A|$, et on écrit plus simplement $t \Vdash A$ dans le cas où $(t, s) \in |A|$ pour tout état $s \in \mathscr{S}$. Avec ces notations, on démontre un lemme d'adéquation très similaire à la Prop. 26 (p. 85). En particulier, les règles de typage utilisées sont exactement les mêmes que celles de de la Fig. 4.1 : la réalisabilité avec états modifie en profondeur la structure du modèle de réalisabilité, mais ne touche pas aux règles de typage ; aussi les termes issus des preuves de l'arithmétique classique du second ordre seront-ils exactement les mêmes qu'auparavant.

Opérateurs de mise en mémoire et extraction de témoin La théorie des opérateurs de mise en mémoire et les différentes procédures d'extraction de témoin décrites à la section 4.2 s'adaptent sans difficulté au nouveau cadre de la réalisabilité avec états. Dans ce cadre étendu, les réalisateurs peuvent à présent faire intervenir des instructions spécifiques pour lire et modifier l'état courant.

5.1.3 Application à l'axiome du choix dénombrable

Le cadre de la réalisabilité classique avec états (section 5.1.2) permet de reformuler de manière très élégante la méthode de réalisation de l'axiome du choix dénombrable à l'aide d'une *horloge* [61]. Pour cela on se place dans le cas où $\mathscr{S} = \mathbb{N}$ (muni de l'ordre usuel), et on considère une nouvelle instruction χ munie de la règle d'évaluation

$$\chi \star t \cdot \pi \star n \quad \succ \quad t \star \bar{n} \cdot \pi \star (n+1)\,.$$

Intuitivement, l'instruction χ récupère la valeur n stockée dans l'état et la passe (sous la forme d'un entier de Krivine) au terme t avant d'incrémenter l'état courant. Suivant le comportement des autres instructions (et notamment des instructions purement logiques) on peut interpréter indifféremment l'instruction χ comme une horloge ou comme un compteur :

L'horloge Dans le cas où toutes les instructions incrémentent l'état courant — y compris les instructions purement logiques (abstraction, application, cc, k_π) — celui-ci peut être vu comme une horloge permettant de mesurer le nombre d'étapes d'évaluation depuis le démarrage du processus. L'instruction χ permet alors d'accéder à la valeur courante de l'horloge, tout en incrémentant l'état comme les autres instructions.

Le compteur Dans le cas où toutes les instructions sauf l'instruction χ laissent inchangé l'état courant, celui-ci peut être vu comme un compteur qui est incrémenté à chaque appel de l'instruction χ (laquelle instruction retourne l'ancienne valeur du compteur).

Quelle que soit l'interprétation retenue parmi les deux interprétation proposées ci-dessus, l'instruction χ permet de construire une quasi-preuve (contenant χ) réalisant l'axiome du choix dénombrable dans PA2 :

$$\forall^{\mathsf{N}} x\ \exists Y\, A(x,Y)\ \Rightarrow\ \exists Z\ \forall^{\mathsf{N}} x\, A(x,Z(x))\big)$$

où Y et Z sont des variables du second ordre d'arité k et $k+1$ respectivement. (Les détails de la construction sont donnés dans [78].) Comme dans [61], ce résultat nécessite de poser un certain nombre de conditions raisonnables sur l'ensemble $\perp\!\!\!\perp$, qui sont en pratique suffisamment peu restrictives pour conserver les principaux résultats d'extraction de témoin (cf section 4.2).

Je conjecture qu'une construction similaire permet également de réaliser l'axiome du choix dépendant :

$$\forall X\ \exists Y\, A(X,Y)\ \Rightarrow\ \exists Z\ \forall^{\mathsf{N}} x\ A(Z(x),Z(s(x)))$$

où X et Y sont des variables du second ordre d'arité k, et où Z est une variable du second ordre d'arité $k+1$.

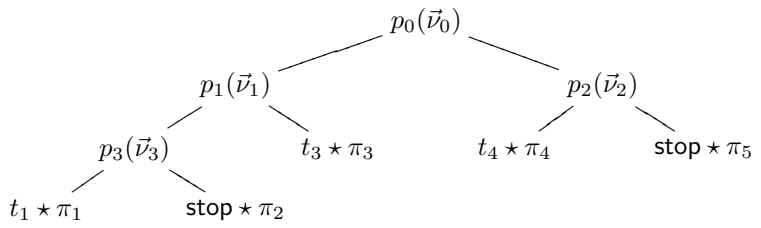
5.1.4 Autres applications envisagées

Le tiers-exclu sur les formules Π^0_1 Dans [6], Aschieri et Berardi proposent une forme de réalisabilité intuitionniste dans laquelle l'utilisation d'une base de connaissances auxiliaire permet de réaliser le principe du tiers-exclu sur les formules Π^0_1 :

$$\exists x\, p(x) \vee \forall x\, \neg p(x)\,.$$

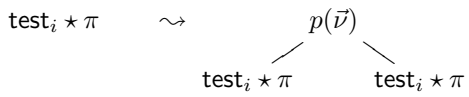
Il est naturel de se demander si cette forme particulière de réalisabilité intuitionniste — qui procède par la méthode essai/erreur — ne peut pas s'exprimer dans le cadre de la réalisabilité avec états qui nous avons présentée. Dans un tel cadre, $\mathscr{S}$ serait naturellement constitué par l'ensemble des bases de connaissances possibles (muni d'une relation de préordre assez naturelle), et le tiers-exclu sur les formules Π^0_1 serait réalisé à l'aide d'une instruction spécifique capable d'interroger (et éventuellement de modifier) la base de connaissances sans passer par l'instruction *call/cc*.

Extraction d'arbres de Herbrand Dans la section 4.2.6, j'ai montré comment il est possible d'extraire un arbre de Herbrand falsifiant une théorie expérimentale $U_1,\dots,U_\ell$ à partir d'un réalisateur classique du séquent $U_1,\dots,U_\ell \vdash \bot$. Telle que je l'ai décrite, cette procédure d'extraction consiste à exécuter (en parallèle ou de manière séquentielle) un ensemble fini de processus attachés au feuilles de l'arbre de Herbrand en cours de construction :



Ici, les nœuds sont étiquetés par des « expériences » de la forme $p(\nu_1,\dots,\nu_k)$ et les feuilles par des processus, vivants (de la forme $t \star \pi$ avec $t \neq \mathsf{stop}$) ou morts

(de la forme $\mathsf{stop} \star \pi$), ce dernier cas correspondant à une falsification de la branche correspondante. À chaque feuille étiquetée par un processus vivant, les instructions de test s'évaluent en utilisant la « base de connaissances » fournie par le chemin courant. Lorsqu'une instruction de test test_i a besoin de connaître la valeur de vérité d'une formule atomique $p(\vec{\nu})$ qui n'est pas donnée par le chemin courant, un nouveau nœud interne est créé qui pointe sur deux copies du processus courant :



Là encore, il est naturel de se demander si cette procédure d'extraction ne peut pas s'exprimer directement en λ_c-calcul (impératif !) avec une notion d'état contenant une forme de *zipper* de l'arbre de Herbrand en cours de construction, c'est-à-dire : à la fois les paramètres de falsification des branches mortes et les processus « endormis » attachés aux extrémités des branches vivantes. Dans ce cadre, on peut imaginer que les instructions de test se chargent de l'extension de l'arbre courant (en y insérant de nouveaux nœuds internes) tandis que l'instruction stop se charge de tuer le processus courant et de « réveiller » le processus suivant dans l'ensemble des branches actives.

Toute la difficulté d'un tel travail réside évidemment dans la définition de la « bonne » notion d'état munie d'un préordre approprié au problème.

5.1.5 Curry-Howard et effets de bord

Le cadre de la réalisabilité classique avec états (section 5.1.2) permet d'envisager l'intégration des traits de programmation impératifs au sein de la correspondance de Curry-Howard (en logique classique) suivant le slogan :

$$\text{Effet de bord} \quad = \quad \text{Modification d'une condition de forcing.}$$

Il s'agit bien évidemment d'un programme de recherche ambitieux (et donc à long terme) qui couvre le traitement des références, des tableaux, des tables de hachage (pour l'axiome du choix ou l'axiome de description ?), des entrées-sorties, etc. mais sans doute aussi le parallélisme et la concurrence.

Les références comme des entiers non standard ? Ainsi que Krivine l'a noté à plusieurs reprises dans ses exposés, il n'est pas nécessaire d'interpréter les individus (de la logique du second ordre) par les entiers naturels pour être en mesure de définir le modèle de réalisabilité classique décrit à la section 4.1.4. Plus généralement, on peut interpréter les individus par les éléments de n'importe quel ensemble $\mathcal{D}$ du moment que cet ensemble dispose :

- d'une fonction injective $\boldsymbol{s} : \mathcal{D} \to \mathcal{D}$ (pour interpréter le successeur) ;
- d'un élément $\mathbf{0} \in \mathcal{D}$ en dehors de l'image de la fonction $\boldsymbol{s}$ (pour interpréter le zéro) ; et plus généralement
- pour chaque définition d'une fonction primitive récursive f d'arité k, d'une fonction $\boldsymbol{f} : \mathcal{D}^k \to \mathcal{D}$ satisfaisant la théorie équationnelle de f.

Bien entendu, ces conditions impliquent que le domaine $\mathcal{D}$ contient un sous-ensemble isomorphe à $\mathbb{N}$, sur lequel les fonctions $\boldsymbol{f} : \mathcal{D}^k \to \mathcal{D}$ coïncident avec les fonctions primitives récursives qu'elles sont censées interpréter. (Le lecteur

vérifiera sans peine que les principaux résultats du chapitre 4 s'étendent au cadre plus général où $\mathbb{N}$ est remplacé par un tel ensemble $\mathcal{D}$.) L'intérêt d'une telle extension des individus réside dans le fait qu'on peut espérer réaliser, au moyen d'instructions appropriées, la formule $\mathsf{Nat}(d)$ pour des objets $d \in \mathcal{D}$ qui ne sont pas des entiers naturels — autrement dit : des entiers non standard.

Dans le cadre de la réalisabilité avec états (basée sur un ensemble $\mathscr{S}$), il est naturel de remplacer l'ensemble des entiers naturels $\mathbb{N}$ par l'ensemble $\mathcal{D}$ des *entiers dépendants* (de l'état courant), c'est-à-dire par l'ensemble des fonctions partielles $\boldsymbol{n} : \mathscr{S} \rightharpoonup \mathbb{N}$ qui sont monotones en ce sens que

$$s \in \mathrm{dom}(\boldsymbol{n}) \ \wedge \ s \leq s' \quad \Rightarrow \quad s' \in \mathrm{dom}(\boldsymbol{n}) \ \wedge \ \boldsymbol{n}(s') = \boldsymbol{n}(s)$$

pour tous $s, s' \in \mathscr{S}$. (Cet ensemble satisfait toutes les conditions mentionnées précédemment.) Il convient également d'adapter dans ce nouveau cadre la définition des deux formes d'implication en appel par valeur $\{e\} \Rightarrow B$ (section 4.2.1) et $[e] \Rightarrow B$ (section 4.2.5) correspondant aux deux représentations des entiers naturels en réalisabilité (les entiers de Krivine et les entiers primitifs).

Au niveau syntaxique, cet élargissement de l'ensemble des individus permet d'enrichir les expressions arithmétiques avec de nouveaux symboles de constante représentant des entiers naturels dont la valeur dépend de l'état : par exemple, le contenu d'une référence, ou la prochaine valeur (entière) lue sur l'entrée standard. Étant donné un entier dépendant $\boldsymbol{n} \in \mathcal{D}$ représenté dans la syntaxe des expressions arithmétiques par une constante c, on dit qu'une quasi-preuve $t \in \Lambda$ est un *accesseur* de l'entier dépendant $\boldsymbol{n}$ si pour tous $u \in \Lambda$, $\pi \in \Pi$ et $s \in \mathrm{dom}(\boldsymbol{n})$, il existe un état $s' \geq s$ tel que

$$t \star u \cdot \pi \star s \quad \succ^* \quad u \star \widehat{\boldsymbol{n}(s)} \cdot \pi \star s'$$

(en utilisant ici la représentation avec des entiers primitifs introduite à la section 4.2.5). Autrement dit : un accesseur de l'entier dépendant $\boldsymbol{n}$ est une quasi-preuve $t \in \Lambda$ permettant de récupérer la valeur de cet entier dépendant dans l'état courant et de la passer au bloc de retour sur le sommet de la pile. On vérifie alors aisément que tout accesseur de l'entier dépendant $\boldsymbol{n}$ est un réalisateur universel de la formule $\mathsf{Nat}'(c)$ (cf section 4.2.5), où c est la constante qui dans la syntaxe représente cet entier dépendant. Ce qui montre que tout entier dépendant $\boldsymbol{n}$ pour lequel on dispose d'un accesseur est effectivement un entier naturel au sens de la réalisabilité classique, c'est-à-dire : un individu c pour lequel on sait réaliser la formule $\mathsf{Nat}'(c)$ avec une quasi-preuve — ici : l'accesseur.

Cette constatation permet d'envisager l'intégration des références (et par extension : de toutes les primitives susceptibles de calculer un entier en fonction de l'état) en réalisabilité classique avec états sous la forme d'*entiers non standard* représentés par des constantes spécifiques dans les expressions arithmétiques. Si cette constatation permet de typer de manière élégante les primitives de « lecture » (d'un entier en fonction d'un état), elle ne donne pas encore la solution pour typer les primitives d'« écriture » correspondantes — un problème pour lequel un gros travail de recherche reste encore à faire.

5.1.6 Un exemple simple d'ordonnanceur

Comme je l'ai déjà évoqué plus haut, le cadre de la réalisabilité avec états n'exige pas que les opérations d'évaluation purement logiques soient atomiques.

On peut tirer parti de ce degré de liberté supplémentaire pour implémenter un petit ordonnanceur dans lequel les processus dormants sont stockés dans l'état courant. Voici comment on procède.

Un *processus simple* est un processus au sens de la réalisabilité classique ordinaire, c'est-à-dire un couple formé par un terme clos et une pile. L'ensemble des processus simples est muni de la relation d'évaluation (simple) engendrée par les seules règles logiques : Push, Grab, Save et Restore.

Un *état* est une liste finie de processus simples. L'ensemble des états, défini par $\mathscr{S} = (\Lambda \star \Pi)^*$, est muni de sa structure de monoïde libre qu'on note multiplicativement. La relation de préordre $s \leq s'$ sur l'ensemble des états est définie à partir de la relation d'évaluation simple avec les règles :

$$\frac{}{\varepsilon \leq \varepsilon} \qquad \frac{s \leq s' \qquad t \star \pi \succ^* t' \star \pi'}{s(t \star \pi) \leq s'(t' \star \pi')}$$

Dans ce cadre, un processus avec état (i.e. un élément de $\Lambda \star \Pi \star \mathscr{S}$) est essentiellement une liste non vide de processus simples dont le premier élément est en position active :

$$t_0 \star \pi_0 \star (t_1 \star \pi_1)(t_1 \star \pi_2) \cdots (t_n \star \pi_n) \qquad (n \geq 0)$$

La rotation des processus simples au sein d'un processus avec état est assurée au moyen d'une fonction $\mathsf{switch} : \Lambda \star \Pi \star \mathscr{S} \to \Lambda \star \Pi \star \mathscr{S}$ définie par :

$$\begin{aligned}\mathsf{switch}\big(t_0 \star \pi_0 \star (t_1 \star \pi_1)(t_1 \star \pi_2) \cdots (t_n \star \pi_n)\big) \;\equiv\\ t_1 \star \pi_1 \star (t_2 \star \pi_2) \cdots (t_n \star \pi_n)(t_0 \star \pi_0)\end{aligned}$$

L'ensemble des processus avec états est à son tour muni d'une relation d'évaluation qui simule le fonctionnement d'un ordonnanceur :

$$\begin{array}{rcll}
\lambda x\,.\,t \star u \cdot \pi \star s & \succ & \mathsf{switch}(t\{x := u\} \star \pi \star s) & \\
tu \star \pi \star s & \succ & \mathsf{switch}(t \star u \cdot \pi \star s) & \\
\mathsf{cc} \star u \cdot \pi \star s & \succ & \mathsf{switch}(u \star \mathsf{k}_\pi \cdot \pi \star s) & \\
\mathsf{k}_\pi \star u \cdot \pi' \star s & \succ & \mathsf{switch}(u \star \pi \star s) & \\
t \star \pi \star s & \succ & \mathsf{switch}(t \star \pi \star s) & \text{si } t \star \pi \not\succ
\end{array}$$

(La dernière règle est essentielle pour empêcher qu'un processus simple qui ne peut plus s'évaluer puisse bloquer le mécanisme d'ordonnancement.)

L'ordonnanceur que nous venons de définir est équitable en ce sens que si $t \star \pi \succ t' \star \pi'$ (évaluation simple), alors pour tout état $s \in \mathscr{S}$ il existe un état $s' \geq s$ tel que $t \star \pi \star s \succ^* t' \star \pi' \star s'$ (évaluation avec états). Cette propriété d'équité implique que la relation d'évaluation avec états que nous venons de définir satisfait toutes les conditions énoncées à la section 5.1.2 ; on pourra donc l'utiliser pour définir un modèle de réalisabilité. Il est par ailleurs facile d'enrichir la relation d'évaluation avec états (sans perdre la propriété d'équité) avec de nouvelles instructions permettant de tuer le processus actif (comme la primitive `kill` sous Unix) ou de lancer un nouveau processus (`fork`).

Cet exemple d'ordonnanceur n'est pour l'instant rien d'autre qu'un exemple jouet pour lequel je n'ai pas d'interprétation logique intéressante à proposer, mais il illustre de manière particulièrement frappante tout ce qu'il est possible d'effectuer dans le cadre de la réalisabilité avec états. Peut-être s'agit-il aussi d'une piste intéressante pour réaliser l'axiome du choix (sous sa forme la plus générale) dans un cadre déterministe (voir à ce sujet la note 9 p. 88).

5.2 Extraction classique en Coq

Lorsque j'ai commencé à apprendre la théorie de la réalisabilité classique, les rares personnes qui s'y intéressaient (toutes membres ou anciens membres du laboratoire PPS à Paris 7) ne l'abordaient que sous un angle purement théorique[6] : les λ-termes étaient extraits à partir des preuves (dans PA2) à la main, dans la plus pure tradition de l'arithmétique fonctionnelle du second ordre (AF2) [57]. Ceci explique sans doute pourquoi la réalisabilité classique a longtemps eu — et a encore — la réputation (largement injustifiée) de n'avoir qu'un intérêt purement théorique, faute d'un lien avec la programmation de tous les jours. Cette situation me semblait d'autant plus regrettable qu'à peine vingt kilomètres plus au sud de Paris était développé l'un des assistants à la démonstration les plus utilisés en Europe, qui proposait depuis longtemps des mécanismes d'extraction basés sur la seule logique intuitionniste. Il m'a donc semblé urgent de rapprocher les deux mondes, à la fois pour montrer que la réalisabilité classique pouvait avoir un intérêt pratique, mais aussi pour montrer que le calcul des constructions inductives et ses types dépendants pouvaient fort bien s'accommoder d'un mécanisme d'extraction classique similaire à celui de AF2. Aujourd'hui encore, je considère que le rapprochement de la réalisabilité classique avec le système Coq (ou avec tout autre assistant à la démonstration) est un élément essentiel dans la dissémination des idées développées autour de la correspondance de Curry-Howard en logique classique (et au-delà).

C'est dans cet esprit que j'ai étendu (section 4.3) le modèle de réalisabilité classique développé par Krivine au calcul des constructions avec univers, pour pouvoir ensuite développer — sur la base de cette extension de la théorie — un petit prototype d'extracteur classique capable de traduire n'importe quel terme de preuve de Coq en un λ_c-terme, notamment pour calculer un témoin d'une formule Σ_1^0 à partir d'une preuve classique de cette formule.

5.2.1 Extension du modèle aux types inductifs

Le module d'extraction classique (section 4.4) que j'ai implémenté est en avance sur la théorie : il est capable de traiter toutes les constructions du système Coq — y compris les types (co)inductifs et les définitions de fonctions par (co)point-fixe — alors que la théorie sur laquelle il est basé [77] ne sait interpréter que le fragment purement fonctionnel du système (c'est-à-dire le PTS sous-jacent). Pour cette raison, rien ne certifie que le schéma d'extraction que j'ai défini est correct, et l'extracteur classique ne peut être destiné pour le moment qu'à un usage purement expérimental.

Un modèle de réalisabilité classique complet pour Coq — qui sache traiter toutes les constructions du langage — reste donc à définir. Pour le moment je n'ai étendu la fonction d'interprétation (et vérifié la correction du schéma d'extraction correspondant) que pour quelques constructions de base telles que les entiers naturels ou les sommes disjointes, mais j'ai bon espoir de pouvoir un jour étendre le modèle de réalisabilité classique au formalisme tout entier. Ici, il est à noter que la difficulté est moins liée à l'aspect classique qu'à la nécessité d'interpréter dans toute sa généralité le schéma de définition de types

[6] À l'exception notable de Yves Legrandgérard, qui avait développé un évaluateur du λ_c-calcul pour tester les termes proposés par Jean-Louis Krivine.

(co)inductifs de Coq, dont la grande expressivité s'accompagne inévitablement d'une certaine lourdeur dès qu'il s'agit de traiter ses aspects sémantiques.

5.2.2 Optimisation de l'extraction classique

L'extracteur classique est muni d'un dispositif de remplacement (décrit à la section 4.4.4) qui permet de substituer un certain nombre de réalisateurs prédéfinis à ceux qui seraient normalement produits par le système au cours de l'extraction. Outre le traitement des axiomes classiques, ce mécanisme permet d'effectuer à la volée un certain nombre d'optimisations essentielles sur le code produit, à savoir :

- Un changement de représentation des structures de données attachées à certains types inductifs. Ce changement de représentation concerne pour l'instant le type `nat` (dont les habitants sont traduits par des entiers paresseux basés sur le type `Big_int` de Caml) et les types $n \leq m$ (que l'extracteur redéfinit comme une forme particulière de type égalité[7]).
- Le remplacement des réalisateurs issus de certaines preuves de la librairie standard par des réalisateurs beaucoup plus courts et efficaces (qui conservent néanmoins le même comportement calculatoire).

On notera que dans le cadre de l'extraction classique, cette dernière forme d'optimisation est cruciale pour l'efficacité du code produit. Contrairement à l'extraction constructive [65], les justifications qui accompagnent les témoins existentiels doivent être conservées dans le code extrait : ce sont en effet les réalisateurs issus de ces justifications qui permettent de propager les points de *backtrack* à travers le calcul. Les quelques tests que j'ai effectués montrent qu'en l'absence d'une telle optimisation, le calcul du point de backtrack coûte dans la plupart des cas bien plus cher que le calcul du témoin proprement dit.

Dans la version actuelle de l'extracteur, l'optimisation des réalisateurs de la librairie standard ne concerne qu'un faible nombre de théorèmes, dont les réalisateurs optimisés (souvent très simples) ont été construits à la main dans le fichier de remplacement. Pour systématiser et automatiser cette forme d'optimisation, il est d'abord nécessaire de dégager dans le langage de Coq une ou plusieurs classes de propositions jouant un rôle similaire aux formules de Harrop en réalisabilité intuitionniste. On notera cependant que la théorie des formules de Harrop ne peut pas être reprise telle quelle en réalisabilité classique, les preuves de ces formules ayant un comportement calculatoire bien plus riche qu'en réalisabilité intuitionniste[8].

5.2.3 Un compilateur en mode natif

L'évaluateur `jivaro` qui est fourni avec le module d'extraction classique en Coq est pour l'instant une simple machine à pile implémentée en Caml de la manière la plus traditionnelle qui soit : les termes clos sont représentés par des fermetures et les piles par des listes.

À plus long terme, il est raisonnable d'envisager l'implémentation d'un véritable compilateur produisant du code machine (ou du moins : du code écrit

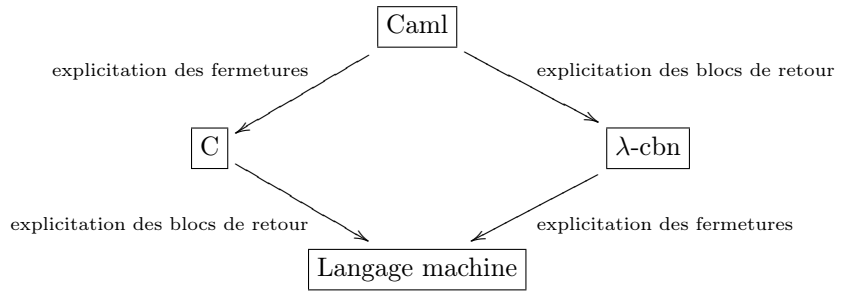
[7] Sans cette optimisation, la taille du terme extrait à partir de la preuve canonique de $0 \leq n$ est en $O(n^2)$. Après le changement de représentation, cette taille devient constante.

[8] On peut comprendre cette différence à travers le filtre de la réalisabilité intuitionniste en remarquant que la classe des formules de Harrop n'est pas stable par A-traduction.

dans un petit fragment du langage C) en combinant la pile système avec des listes chaînées de blocs de pile pour représenter la pile courante (suivant une technique habituelle dans les langages fonctionnels avec opérateurs de contrôle). À cet égard, il est utile de rappeler que le schéma de compilation de λ_c est beaucoup plus simple que celui de ML puisqu'en appel par nom, tous les appels de fonction sont terminaux. (On peut donc les traduire par des `goto`.)

La clé de la réussite d'un tel travail réside sans doute dans une réutilisation intelligente de la représentation des structures de données et des fermetures en Caml (pour assurer la compatibilité des données et donc l'interopérabilité entre les deux langages) ainsi que de son *garbage collector* générationnel.

Notons par ailleurs qu'il serait maladroit de chercher à compiler λ_c vers le langage Caml (étendu avec des opérateurs de contrôle) puisque ce dernier est un langage de plus haut niveau que λ_c. Dans Caml (comme dans tous les langages fonctionnels en appel par valeur), la gestion des fermetures et des blocs de retour est implicite, tandis que dans λ_c (comme dans tous les langages fonctionnels en appel par nom), seule la gestion des fermetures est implicite, les blocs de retours devant être obligatoirement mentionnés dans les fonctions sous la forme d'un argument supplémentaire. De fait, il est possible de compiler Caml en deux passes vers le langage machine en utilisant comme langage intermédiaire soit le langage C, soit le λ-calcul en appel par nom. Ce qui donne lieu à deux chaînes de compilation résumées dans le diagramme ci-dessous :



5.2.4 Extraction classique et effets de bord

Il est encore trop tôt pour pouvoir prédire l'impact que pourra avoir la réalisabilité avec états dans le cadre de l'extraction classique en Coq, même s'il me semble clair que les deux sont parfaitement compatibles dans la mesure où le modèle de réalisabilité classique du calcul des constructions avec univers s'étend sans difficulté en un modèle de réalisabilité avec états.

Cependant, la réalisabilité avec états offre des pistes intéressantes pour réaliser un certain nombre de principes logiques très utiles dans le cadre de la formalisation effective des mathématiques, et notamment diverses formes de l'axiome du choix et de l'axiome de description. Un principe fort utile en pratique est celui qui exprime que toute relation fonctionnelle sur $\mathbb{N}$ est une fonction (au sens de la théorie des types) :

$$\forall x \in \mathbb{N} \;\; \exists! y \in \mathbb{N} \;\; R(x,y) \quad \Rightarrow \quad \exists f \in \mathbb{N}^{\mathbb{N}} \;\; \forall x \in \mathbb{N} \;\; R(x, f(x))$$

Diverses méthodes [13, 14] ont été proposées pour réaliser la forme d'axiome du choix correspondante (en retirant l'hypothèse d'unicité sur y) à travers une

traduction négative de la logique classique, suivant une interprétation due à Spector. Mais les quelques tentatives que j'ai faites pour reformuler ces techniques dans le cadre défini par Krivine semblent indiquer de profondes incompatibilités entre les deux cadres. Là encore, on peut espérer que la réalisabilité classique avec états puisse remédier à cette situation, en donnant les outils nécessaires pour construire incrémentalement la fonction f, par exemple avec des techniques de mémoïsation.

Table des matières

Bibliographie

[1] P. Aczel. Non well-founded sets. *Center for the Study of Language and Information*, 1988.

[2] P. Aczel. On relating type theories and set theories. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *TYPES*, volume 1657 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1998.

[3] T. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, University of Edinburgh, November 1993.

[4] A. Arbiser, A. Miquel, and A. Ríos. A lambda-calculus with constructors. In Frank Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2006.

[5] A. Arbiser, A. Miquel, and A. Ríos. The lambda-calculus with constructors : syntax, confluence and separation. *Journal of Functional Programming*, 19(5) :581–631, 2009.

[6] F. Aschieri and S. Berardi. Interactive learning-based realizability interpretation for heyting arithmetic with em1. In Curien [27], pages 20–34.

[7] F. Baader and T. Nipkow. *Rewriting and All That.* Addison-Wesley, 1999.

[8] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Inf. Comput.*, 125(2) :103–117, 1996.

[9] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics.* North-Holland, 1984.

[10] H. Barendregt. Introduction to generalized type systems. Technical Report 90-8, University of Nijmegen, Department of Informatics, May 1990.

[11] H. Barendregt and Herman Geuvers. Proof-assistants using dependent type systems. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1149–1238. Elsevier and MIT Press, 2001.

[12] J. L. Bell. *Boolean-Valued Models and Independence Proofs in Set Theory.* Oxford, 1985.

[13] S. Berardi, M. Bezem, and T. Coquand. A realization of the negative interpretation of the axiom of choice. In Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin, editors, *TLCA*, volume 902 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 1995.

[14] U. Berger and P. Oliva. Modified bar recursion. *Mathematical Structures in Computer Science*, 16(2) :163–183, 2006.

[15] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art : The Calculus of Inductive Constructions*. Springer, 2004.

[16] C. Böhm, M. Dezani-Ciancaglini, P. Peretti, and S. Ronchi Della Rocha. A discrimination algorithm inside lambda-beta-calculus. *Theoretical Computer Science*, 8(3) :265–291, 1979.

[17] L. E. J. Brouwer. Intuïtionistische splitsing van mathematische grondbegrippen. *Nederl. Akad. Wetensch. Verslagen*, 32 :877–880, 1923.

[18] L. Chicli, L. Pottier, and C. Simpson. Mathematical quotients and quotient types in coq. In Geuvers and Wiedijk [40], pages 95–107.

[19] A. Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1 :40–41, 1936.

[20] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58 :345–363, 1936.

[21] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5, 1940.

[22] A. Church. *The calculi of lambda-conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton, 1941.

[23] P. J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6) :1143–1148, December 1963.

[24] P. J. Cohen. The independence of the continuum hypothesis II. *Proceedings of the National Academy of Sciences of the United States of America*, 51(1) :105–110, January 1964.

[25] T. Coquand and H. Herbelin. A-translation and looping combinators in pure type systems. *Journal of Functional Programming*, 4(1) :77–88, 1994.

[26] M. Crabbé. Non-normalisation de ZF. Manuscrit, 1974.

[27] P.-L. Curien, editor. *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, volume 5608 of *Lecture Notes in Computer Science*. Springer, 2009.

[28] P.-L. Curien and H. Herbelin. The duality of computation. In *ICFP*, pages 233–243, 2000.

[29] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland, 1958.

[30] G. Dowek. *Les Métamorphoses du calcul : Une étonnante histoire des mathématiques*. Éditions le Pommier, 2007.

[31] G. Dowek and A. Miquel. Cut elimination for Zermelo's set theory. Manuscrit.

[32] G. Dowek and B. Werner. Proof normalization modulo. *Journal of Symbolic Logic*, 68(4) :1289–1316, 2003.

[33] O. Esser and R. Hinnion. Antifoundation and transitive closure in the system of Zermelo. *Notre Dame Journal of Formal Logic*, 40(2) :197–205, 1999.

[34] X. Leroy et al. The objective caml system release 3.11 : Documentation and user's manual. Technical report, INRIA, November 2008.

[35] M. Forti and F. Honsell. Set theory with free construction principles. *Annali della Scuola Normale Superiore di Pisa - Classe di Scienze, Ser. 4, 10 no. 3*, pages 493–522, 1983.

[36] H. Friedman. The consistency of classical set theory relative to a set theory with intuitionistic logic. *J. Symb. Log.*, 38(2) :315–319, 1973.

[37] H. Friedman. Some applications of Kleene's methods for intuitionistic systems. In *Cambridge Summer School in Mathematical Logic*, volume 337 of *Springer Lecture Notes in Mathematics*, pages 113–170. Springer-Verlag, 1973.

[38] H. Friedman and A. Ščedrov. The lack of definable witnesses and provably recursive functions in intuitionistic set theories. *Advances in Mathematics*, 57(1) :1–13, 1985.

[39] G. Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39(1) :176–210, 1935.

[40] H. Geuvers and F. Wiedijk, editors. *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, volume 2646 of *Lecture Notes in Computer Science*. Springer, 2003.

[41] J.-Y. Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Doctorat d'État, Université Paris VII, Juin 1972.

[42] J.-Y. Girard. *Locus Solum. Mathematical Structures in Computer Science (MSCS)*, 2001.

[43] J.-Y. Girard. *Le point aveugle – Cours de logique – Volume I – vers la perfection*. Hermann, 2006.

[44] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[45] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12 :280–287, 1958.

[46] T. Griffin. A formulae-as-types notion of control. In *Principles Of Programming Languages (POPL'90)*, pages 47–58, 1990.

[47] M. Guillermo. *Jeux de réalisabilité en arithmétique classique*. PhD thesis, Université Paris 7, 2008.

[48] A. Heyting. *Mathematische grundlagenforschung. Intuitionismus. Beweisteorie*. Springer, 1934.

[49] D. Hilbert and P. Bernays. *Grundlagen der Mathematik I*. Springer Verlag, 1934. (2e éd., 1968).

[50] D. Hilbert and P. Bernays. *Grundlagen der Mathematik II*. Springer Verlag, 1939. (2e éd., 1970).

[51] D. Hilbert and P. Bernays. *Fondements des mathématiques*. Éditions L'Harmattan, 2001. (Trad. : F. Gaillard, E. Gaillard et M. Guillaume).

[52] W. A. Howard. The formulae-as-types notion of construction. Privately circulated notes, 1969.

[53] J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*. North Holland, 1982.

[54] S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10 :109–124, 1945.

[55] G. Kreisel. On the interpretation of non-finitist proofs, part I. *Journal of Symbolic Logic*, 16 :241–267, 1951.

[56] G. Kreisel. On the interpretation of non-finitist proofs, part II : Interpretation of number theory. *Journal of Symbolic Logic*, 17 :43–58, 1952.

[57] J. L. Krivine. *Lambda-calcul, types et modèles*. Masson, 1991.

[58] J.-L. Krivine. A general storage theorem for integers in call-by-name lambda-calculus. *Th. Comp. Sc.*, 129 :79–94, 1994.

[59] J.-L. Krivine. *Théorie des ensembles*. Cassini, 1998.

[60] J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3) :189–205, 2001.

[61] J.-L. Krivine. Dependent choice, 'quote' and the clock. *Theor. Comput. Sci.*, 308(1-3) :259–276, 2003.

[62] J.-L. Krivine. Realizability in classical logic. *Panoramas et synthèses, Société Mathématique de France*, 2005.

[63] J.-L. Krivine. Structures de réalisabilité, RAM et ultrafiltre sur N. Manuscript, available on the author's web page, 2008.

[64] S. Lengrand and A. Miquel. Classical F_ω, orthogonality and symmetric candidates. *Ann. Pure Appl. Logic*, 153(1-3) :3–20, 2008.

[65] P. Letouzey. A new extraction for Coq. In Geuvers and Wiedijk [40], pages 200–219.

[66] G. Longo and E. Moggi. A category-theoretic characterization of functional completeness. *Theor. Comput. Sci.*, 70(2) :193–211, 1990.

[67] P. Martin-Löf. A theory of types. Manuscrit non publié, 1971.

[68] P. Martin-Löf. An intuitionistic type theory : Predicative part. *Logic Colloquium*, pages 73–118, 1975.

[69] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[70] D. McCarty. *Realizability and Recursive Mathematics*. PhD thesis, Oxford University, 1984.

[71] T. F. Melham. The HOL logic extended with quantification over type variables. *Formal Methods in System Design*, 3(1/2) :7–24, 1993.

[72] P.-A. Melliès and B. Werner. A generic normalisation proof for pure type systems. In Eduardo Giménez and Christine Paulin-Mohring, editors, *TYPES*, volume 1512 of *Lecture Notes in Computer Science*, pages 254–276. Springer, 1996.

[73] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, 1997.

[74] A. Miquel. *Le calcul des constructions implicite : syntaxe et sémantique*. PhD thesis, Université Paris 7, 2001.

[75] A. Miquel. A strongly normalising Curry-Howard correspondence for IZF set theory. In Matthias Baaz and Johann A. Makowsky, editors, *CSL'03*, volume 2803 of *Lecture Notes in Computer Science*, pages 441–454. Springer, 2003.

[76] A. Miquel. Lambda-Z : Zermelo's set theory as a PTS with 4 sorts. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *TYPES*, volume 3839 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2004.

[77] A. Miquel. Classical program extraction in the calculus of constructions. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2007.

[78] A. Miquel. Classical realizability with forcing and the axiom of countable choice, 2009. Unpublished manuscript.

[79] A. Miquel. Relating classical realizability and negative translation for existential witness extraction. In Curien [27], pages 188–202.

[80] A. Miquel and B. Werner. The not so simple proof-irrelevant model of cc. In Geuvers and Wiedijk [40], pages 240–258.

[81] W. Moczydłowski. Normalization of IZF with replacement. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 516–530. Springer, 2006.

[82] J. Myhill. Some properties of intuitionistic Zermelo-Fraenkel set theory. *Lecture Notes in Mathematics*, 337 :206–231, 1973.

[83] B. Nordström, K. Peterson, and J. Smith. *Programming in Martin-Löf's Type Theory. An Introduction.* Oxford University Press, 1990.

[84] P. Oliva and T. Streicher. On Krivine's realizability interpretation of classical second-order arithmetic. *Fundam. Inform.*, 84(2) :207–220, 2008.

[85] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. Symb. Log.*, 62(4) :1461–1479, 1997.

[86] C. Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur.* Habilitation à diriger les recherches, Université Claude Bernard Lyon I, 1996.

[87] B. Petit. A polymorphic type system for the lambda-calculus with constructors. In Curien [27], pages 234–248.

[88] K. R. Popper. *La logique de la découverte scientifique.* Payot, 1973.

[89] D. Prawitz. Natural deduction. a proof-theoretical study. *Studies in Philosophy*, 3, 1965.

[90] G. E. Révész. An extension of lambda-calculus for functional programming. *Journal of Logic Programming*, 1(3) :241–251, 1984.

[91] G. E. Révész. A list-oriented extension of the lambda-calculus satisfying the Church-Rosser property. *Theoretical Computer Science*, 93 :75–89, 1992.

[92] C. Riba. On the stability by union of reducibility candidates. In Helmut Seidl, editor, *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2007.

[93] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites.* PhD thesis, Université Paris 7, 1993.

[94] J. Rouxel. Et voilà le Shadok. Service de la Recherche de l'ORTF, 1968. `http://www.ina.fr/`.

[95] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30 :222–262, 1908.

[96] H. Schwichtenberg. Minimal logic for computable functionals. Technical report, Mathematisches Institut der Universität München, 2005.

[97] T. Streicher. *Semantics of Type Theory*. Birkhäuser, 1991.

[98] The Coq Development Team. The coq proof assistant reference manual – version v8.2. Technical report, INRIA, 2009.

[99] Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science, 2003.

[100] A. Trybulec and H. A. Blair. Computer assisted reasoning with Mizar. In Rohit Parikh, editor, *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 406–412. Springer, 1985.

[101] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2) :230–265, 1936.

[102] B. Werner. *Une théorie des Constructions Inductives*. PhD thesis, Université Paris VII, 1994.

[103] B. Werner. Sets in types, types in sets. In Martín Abadi and Takayasu Ito, editors, *TACS*, volume 1281 of *Lecture Notes in Computer Science*, pages 530–346. Springer, 1997.

[104] A. N. Whitehead and B. Russell. *Principia Mathematica*, volume 1. Cambridge University Press, 1910.

[105] E. Wigner. The unreasonable effectiveness of mathematics in the natural sciences. *Communications in Pure and Applied Mathematics*, 13(1), February 1960.

Résumé :

Ce mémoire présente plusieurs résultats de théorie de la démonstration dans le cadre de la correspondance preuves-programmes (dite de Curry-Howard). On y aborde des problèmes de formalisation mathématique (en théorie des ensembles et en théorie des types), de réécriture (filtrage) et d'extraction de programmes à partir de preuves en logique classique.

La première partie est consacrée à l'étude comparée des forces théoriques de la théorie des types et de la théorie des ensembles. Pour cela, nous commençons par rappeler la technique utilisée pour traduire la théorie des ensembles en théorie des types, qui consiste à représenter les ensembles par des graphes pointés. Nous présentons quatre systèmes satisfaisant tous la propriété de normalisation forte : un premier système de types permettant de représenter (au moins) la théorie des ensembles de Zermelo-Fraenkel intuitionniste d'ordre supérieur, un système de types purs (PTS) à quatre sortes capturant exactement la force de la théorie des ensembles de Zermelo, une présentation de la théorie des ensembles de Zermelo intuitionniste en déduction modulo, et finalement une extension de la logique d'ordre supérieur capturant exactement la force de la théorie des ensembles de Zermelo-Fraenkel.

Dans la deuxième partie de ce mémoire, nous présentons le λ-calcul avec constructeurs, une extension du λ-calcul avec un mécanisme de filtrage basé sur la commutation de l'application avec l'analyse par cas. Après avoir donné quelques exemples d'utilisation du formalisme, nous montrons deux théorèmes : le théorème de Church-Rosser modulaire (qui établit la confluence du calcul et caractérise plus généralement tous ses sous-systèmes confluents) et le théorème de séparation faible (dans l'esprit du théorème de Böhm).

La troisième partie est consacrée à l'extraction de programmes à partir de preuves formalisées en logique classique, dans le cadre de la théorie de la réalisabilité classique introduite par Krivine. Nous présentons dans ce cadre plusieurs méthodes (directes) d'extraction de témoin à partir de preuves classiques de formules existentielles, et relions l'une d'entre-elles avec la méthode d'extraction (indirecte) basée sur la A-traduction de Friedman. Nous montrons également comment la réalisabilité classique permet de résoudre le problème du *modus tollens expérimental* soulevé par Popper. Ensuite, nous présentons une extension du modèle de réalisabilité classique de Krivine au calcul des constructions avec univers (le noyau du système Coq), qui permet d'étendre le schéma d'extraction classique à tout le système. Nous terminons cette partie par une description du module d'extraction classique en Coq.

Nous concluons le mémoire sur les perspectives soulevées par les résultats présentés dans la dernière partie, en évoquant des travaux récents qui permettent d'envisager l'intégration des traits de programmation impératifs (effets de bord) à la correspondance preuves-programmes en lien avec la théorie du *forcing*.

Ce mémoire s'appuie notamment sur des travaux en collaboration avec Ariel Arbiser, Gilles Dowek, Stéphane Lengrand, Alejandro Ríos et Benjamin Werner.