UNIVERSITÉ PARIS 7 - DENIS DIDEROT
U.F.R. D'INFORMATIQUE

N° attribué par la bibliothèque

Année 2001

T H È S E

pour l'obtention du diplôme de

DOCTEUR DE L'UNIVERSITÉ PARIS 7
Spécialité : Informatique fondamentale

présentée et soutenue publiquement

par

Alexandre Miquel

le 14 décembre 2001

TITRE :

LE CALCUL DES CONSTRUCTIONS IMPLICITE : SYNTAXE ET SÉMANTIQUE

Directeur de thèse :

M. Hugo Herbelin

JURY

MM.	Pierre-Louis	Curien,	Président
	Thierry	Coquand	
	Furio	Honsell,	Rapporteurs
	Jean-Yves	Girard	
	Jean	Goubault-Larrecq	
	Thomas	Streicher,	Examinateurs

Remerciements

Débuter une thèse après avoir effectué une parenthèse de trois ans dans l'enseignement secondaire loin du monde de la recherche n'était pas une chose qui allait de soi. C'est pourquoi je tiens ici à remercier chaleureusement toutes les personnes qui m'ont assisté dans cette démarche et sans qui les fruits de ce travail n'auraient pas pu voir le jour.

Mes remerciements vont tout d'abord à Hugo Herbelin, qui a encadré mon stage de DEA au printemps 1998 et qui a ensuite pris en charge la direction de ma thèse. Pendant plus de trois ans, Hugo m'a aiguillé et soutenu dans mon travail, en portant toujours un très grand intérêt aux directions que prenait ma recherche, quelles qu'elles étaient. Mais surtout, il m'a apporté son précieux soutien, dans les moments faciles comme dans ceux qui l'étaient moins, avec la grande patience et l'extrême gentillesse qui sont les siennes, et je voudrais ici qu'il sache à quel point je lui en suis reconnaissant.

Une thèse est un travail de longue haleine qu'il n'est pas possible d'entreprendre sans un environnement de travail adéquat. C'est pourquoi je tiens à remercier toute l'équipe du projet LogiCal (ex-Coq), et plus particulièrement Christine Paulin, Gilles Dowek, Benjamin Werner et Bruno Barras, qui m'ont offert un environnement de travail d'une grande qualité scientifique, et dans lequel j'ai disposé d'une liberté exceptionnelle. Cette liberté, j'en ai profité en grande partie pour assouvir ma soif de programmation, et les progrès que j'ai faits dans ce domaine n'auraient sans doute pas été possibles sans les conseils avisés et l'attention permanente des membres du projet Cristal, à qui va également toute ma gratitude. À Paris 7, Chantal Berline a su débusquer une erreur subtile (que je n'aurais sans doute jamais vue sans elle) dans une première présentation de mes modèles, et je l'en remercie.

Je tiens à remercier chaleureusement les rapporteurs de cette thèse, Thierry Coquand et Furio Honsell, pour la relecture attentive du manuscrit comme pour l'intérêt qu'ils ont porté à mon travail, et cela à maintes reprises. Pierre-Louis Curien, qui a très tôt porté son regard sur mon travail, m'a fait bénéficier de ses conseils en relisant une version préliminaire de ma thèse, et je suis très heureux qu'il ait accepté de présider le jury.

Je remercie également Jean-Yves Girard pour les discussions ludiques et enrichissantes sur le sous-typage et la logique ainsi que Jean Goubault-Larrecq et Thomas Streicher pour l'honneur qu'il me font d'examiner ce travail, et qui ont eu la gentillesse de bien vouloir faire partie du jury.

Enfin, je remercie mes proches, et plus particulièrement Thomas, Olivier et Patrice, pour le soutien moral qu'ils m'ont apporté tout au long de cette thèse.

Table des matières

Introduction

Près de trois cents ans après son introduction en mathématiques, la notion de fonction reste un mystère. Pour Euler, à qui l'on doit la notation $f(x)$, une fonction est essentiellement une expression algébrique dépendant d'une variable x. Dans ce cadre, il est possible de manipuler les fonctions et d'effectuer des calculs d'une manière purement formelle, comme par exemple montrer l'égalité

$$\frac{1}{1-x} \quad = \quad 1 + x + x^2 + x^3 + x^4 + \cdots$$

sans jamais se soucier du domaine sur lequel les expressions correspondantes sont définies. Pour Riemann au contraire, une fonction est un procédé — une sorte de "boîte noire" — qui à tout nombre x associe au plus un nombre noté $f(x)$ lorsque ce nombre existe.

$$x \to \boxed{\quad f \quad} \to f(x)$$

Là où la définition d'Euler insiste sur la construction des expressions qui définissent les fonctions et sur leurs propriétés calculatoires, Riemann oppose une conception abstraite selon laquelle ce n'est pas la formule permettant de calculer f qui importe, mais plutôt l'ensemble des valeurs de x pour lesquelles ce calcul est défini, ainsi que les valeurs retournées par le calcul proprement dit.[1]

La définition de Riemann est clairement très adaptée au cadre de la méthode axiomatique qui prédomine depuis plus d'un siècle en mathématiques, et ce n'est donc pas un hasard si cette définition est précisément celle qui est enseignée dans les cours de premier cycle. En théorie des ensembles, ce point de vue abstrait est même poussé à son paroxysme puisque dans ce cadre, chaque fonction est identifiée purement et simplement avec son graphe, c'est-à-dire avec l'ensemble des couples $(x, f(x))$ lorsque x décrit le domaine de définition de f.

Dans les années 1930 cependant, le point de vue d'Euler revient sur le devant de la scène en logique lorsque le mathématicien Alonzo Church invente le λ-calcul [15], un formalisme basé sur la seule notion de fonction. L'approche suivie par Church a ceci de remarquable qu'elle n'est pas fondée sur une axiomatisation mais sur une description purement calculatoire du concept de fonction. Dans ce nouveau cadre dirigé par le calcul, les fonctions acquièrent une telle autonomie que la notion même de domaine de définition perd tout son sens : en λ-calcul, une fonction peut être appliquée à n'importe quel autre objet du formalisme — y compris à elle-même — sans que cela n'affecte en rien le déroulement du processus calculatoire.

Depuis l'apparition des premiers ordinateurs, le λ-calcul est naturellement devenu l'un des outils essentiels de l'informatique théorique, ne serait-ce que parce qu'il capture exactement la

[1] Bien entendu, les deux mathématiciens sont loin d'être contemporains. Riemann propose sa définition un siècle après Euler, à un moment où les notions abstraites de limite et de convergence ont été établies.

notion de fonction calculable. Dans le domaine de la programmation, l'élégance et la remarquable simplicité de ce formalisme ont en outre suscité l'émergence de toute une famille de langages basés sur le même principe — les langages de programmation fonctionnels — parmi lesquels on trouve des langages non-typés, tels que les langages de la famille LISP, ainsi que des langages typés, tels que les langages de la famille ML par exemple.

Un autre champ d'application du λ-calcul est la vérification de la correction des démonstrations mathématiques sur ordinateur, dont un des principaux débouchés est la vérification de programmes. Depuis les années 1970 en effet, on sait exprimer par des termes du λ-calcul les preuves construites dans de nombreux systèmes logiques en utilisant une correspondance très naturelle — la correspondance de Curry-Howard — à travers laquelle une proposition est un type de données, et une preuve de cette proposition un programme habitant ce type de données. Au-delà de ses nombreuses répercussions dans le domaine de la logique, la correspondance de Curry-Howard permet en pratique de réduire très simplement la question « π est-elle une preuve de la proposition A ? » à la question « π est-il un programme de type A ? ».

De nombreux assistants à la démonstration mathématique sont à l'heure actuelle basés sur ce principe, tels que les systèmes Coq [11], LEGO [43], NuPrl [36], ALF [44] et Agda pour n'en citer que quelques-uns. Ces logiciels sont basés sur un formalisme — la théorie des types, ou l'une de ses nombreuses variantes — qui se distingue de la théorie des ensembles non seulement en considérant les programmes comme des objets primitifs, mais surtout en procédant à une identification complète des notions de preuve et de proposition avec les notions de programme et de type respectivement.

Pourtant, contrairement au λ-calcul pur ou aux langages de la famille ML, la théorie des types est généralement basée sur une approche dans laquelle chaque objet est "décoré" avec un certain nombre d'annotations de type qui permettent de déterminer le type de cet objet sans la moindre ambiguïté. En particulier, la théorie des types ne permet de construire que des fonctions dont le domaine de définition est donné à l'avance, réintroduisant ainsi un peu de la conception riemannienne là où les succès du λ-calcul pur semblaient indiquer au contraire que l'on pouvait enfin se passer de la notion de domaine de définition.

Ceci m'amène alors à formuler la question suivante. Quand on désire faire des mathématiques — mais aussi programmer — au sein d'un système aussi riche que la théorie des types, est-il nécessaire de ne considérer que des fonctions dont le domaine est clairement défini ?

C'est à cette question qu'essaie de répondre le Calcul des Constructions implicite, dont l'étude est le principal objet de cette thèse. Bien que le Calcul des Constructions implicite (que j'appellerai plus simplement « calcul implicite » dans ce qui suit) soit formellement une variante du Calcul des Constructions avec univers,[2] ce formalisme diffère des systèmes de types purs standards par le fait que les fonctions qu'il permet d'exprimer n'ont pas réellement de domaine de définition. Techniquement, la disparition du domaine de définition n'est pas seulement obtenue en supprimant l'annotation de type sous la λ-abstraction,[3] mais en introduisant dans le système de types une nouvelle construction appelée *produit dépendant implicite* et notée $\forall x : T \,.\, U$ qui permet de former l'intersection d'une famille de types, et d'exprimer dans le formalisme lui-même qu'un même objet ou une même fonction appartient à une famille (éventuellement infinie) de types différents.

[2] Qui est lui même le noyau du Calcul des Constructions inductives — le formalisme à la base du système Coq — privé de son mécanisme primitif de définitions inductives.

[3] L'étude des systèmes de types purs dépourvus d'annotation de type sous la λ-abstraction constitue l'objet de la théorie des *domain-free pure type systems*, dont on trouvera une présentation dans [12].

Contrairement à l'approche des *Type Assignment Systems* [40, 27, 60] qui est décrite au chapitre 1, le produit dépendant implicite n'est pas cantoné au niveau imprédicatif, mais peut être utilisé à n'importe quel endroit de la hiérarchie d'univers, tout comme le produit dépendant usuel $\Pi x : T \,.\, U$ (qualifié par opposition de *produit dépendant explicite*) avec qui il partage les mêmes règles de formation. Bien entendu, le produit dépendant implicite peut être également utilisé pour exprimer le polymorphisme dans un esprit qui est très proche des langages de programmation de la famille ML, sans surcharger les fonctions avec des arguments de type supplémentaires.

Là où mon approche s'écarte significativement de l'approche de ML, c'est que la généralisation de ce constructeur d'intersection induit naturellement une riche relation de sous-typage (décrite au chapitre 2), dont l'ordre d'instanciation de ML ou du système F deviennent des cas particuliers. Cette relation de sous-typage qui découle naturellement des règles de typage donne une très grande expressivité au formalisme, et c'est un point sur lequel j'aurai l'occasion de revenir quand je décrirai les codages imprédicatifs qu'il est possible d'effectuer dans ce cadre. Par ailleurs, les spécificités du calcul implicite ont certaines conséquences logiques pour le moins surprenantes, puisque le bon fonctionnement de la relation de sous-typage semble nécessiter l'introduction d'une règle de renforcement qui autorise l'utilisation de paralogismes temporaires dans les dérivations, sans pour autant mettre en danger la cohérence logique du formalisme.

La contribution essentielle de cette thèse réside dans les modèles du calcul implicite, dont l'étude détaillée est l'objet de toute la deuxième partie. Contrairement aux systèmes de types usuels (tels que le Calcul des Constructions avec univers par exemple), le calcul implicite semble être totalement rétif aux modèles ensemblistes traditionnels, qui ne permettent d'interpréter ni la très grande ambiguïté typique du formalisme ni la relation de sous-typage qui en découle, ainsi que je le montre dans le chapitre 3.

En revanche, la théorie des espaces cohérents — qui est présentée au chapitre 4 — constitue un bien meilleur cadre pour interpréter les spécificités du calcul implicite. Cette structure permet en particulier d'interpréter le sous-typage de manière très naturelle sans recourir à un quelconque mécanisme de coercitions. Cette interprétation du sous-typage repose sur une interprétation non-standard des types à l'aide d'un paradigme assez simple qui est introduit au chapitre 5, et qui est formellement très proche de celui qui est utilisé par J.-Y. Girard dans la Ludique [30] bien qu'il ne s'y ramène pas complètement.[4] Dans ce même chapitre, j'effectue la construction d'un premier modèle d'une restriction du calcul implicite, et dans lequel tous les termes de preuves sont indiscernables sans être toutefois identifiés comme c'est généralement le cas dans les modèles ensemblistes traditionnels de la *proof-irrelevance*.

Cette étude des modèles du calcul implicite fait apparaître un autre aspect assez inattendu, qui est que la notion de réalisabilité trouve dans les espaces cohérents un nouveau cadre qui lui est très adapté, ainsi que je le montre au chapitre 6. Un des défauts du modèle de réalisabilité standard du Calcul des Constructions avec univers est de recourir systématiquement au raisonnement "à isomorphisme près", ce qui est dû au fait que la partie imprédicative du système de types est interprétée par deux catégories différentes qui, si elles sont équivalentes, sont cependant de tailles très différentes. Dans les espaces cohérents en revanche, la notion

[4] Le paradigme que j'utilise pour interpréter les types n'est pas basé sur l'orthogonalité, contrairement à la notion de comportement introduite par J.-Y. Girard. En particulier, les comportements (*i.e.* les ensembles de cliques égaux à leur bi-orthogonal) sont tous des types sémantiques au sens qui sera défini au chapitre 5 de cette thèse, mais la réciproque est fausse.

de réalisabilité concrète que je présente fait disparaître tout raisonnement à isomorphisme près, ce qui est possible dans la mesure où la représentation des fonctions stables par leurs traces effectue automatiquement toutes les identifications nécessaires. Grâce à cette notion de réalisabilité concrète, je construis un second modèle du même fragment du calcul implicite qui invalide le principe du tiers-exclus, contrairement au premier modèle.

L'étude des modèles du calcul implicite se conclut au chapitre 7 par la construction d'un modèle de normalisation du formalisme complet, dont se déduit le théorème de normalisation forte du calcul implicite qui exprime que tous les programmes que le formalisme permet d'exprimer terminent, et qui constitue le principal résultat technique de cette thèse. Ce théorème qui est établi dans une théorie des ensembles très forte entraîne en particulier la cohérence logique du Calcul des Constructions implicite.

Lorsque j'ai entrepris l'étude du Calcul des Constructions implicite, je me suis naturellement intéressé à son expressivité mathématique, autrement dit je me suis posé la question : jusqu'où est-il possible de formaliser les mathématiques dans le calcul implicite ? Cette interrogation m'a permis de me rendre compte que l'expressivité logique de la plupart des systèmes de types imprédicatifs avec univers (tels que le Calcul des Constructions avec univers par exemple) est en général très mal connue.

Cette question fait l'objet de la troisième partie de cette thèse, qui est totalement indépendante des deux premières parties dans la mesure où elle ne fait pas référence au calcul implicite[5] ni à ses modèles. Dans cette partie, je montre que le paradigme qui consiste à représenter les ensembles par des graphes pointés (et qui est à la base de la théorie des hyper-ensembles et de l'axiome d'anti-fondation) se traduit très naturellement en théorie des types, mais surtout qu'il permet de représenter les ensembles même dans un cadre où l'on ne dispose pas d'un mécanisme de définitions inductives primitives.

Dans le chapitre 8, j'utilise ce paradigme pour traduire le fragment intuitionniste de la théorie des ensembles (incohérente) de Cantor-Frege dans le système U^-, ce qui me permet de montrer que tous les paradoxes de la théorie des ensembles sont susceptibles d'être traduits de manière uniforme dans ce formalisme. Utilisant ce résultat, je donne alors une nouvelle preuve d'incohérence des systèmes U et U^- par une traduction immédiate du paradoxe de Russell.

Le chapitre 9 reprend la même problématique, mais dans le cadre de théories supposées cohérentes. En utilisant cette représentation des ensembles par des graphes pointés, je montre qu'il existe une traduction de la théorie des ensembles intuitionniste de Zermelo d'ordre supérieur dans le système $F\omega$ auquel on a ajouté deux univers prédicatifs, et que ce résultat se généralise naturellement à la théorie classique au moyen d'une non-non traduction définie par T. Coquand et H. Herbelin dans le cadre d'une certaine classe de systèmes de types purs. Ceci me permet en particulier de montrer que le Calcul des Constructions avec univers (sans entiers primitifs ni types inductifs) a une puissance théorique très supérieure à celle de la théorie classique de Zermelo, répondant ainsi à une conjecture posée par T. Coquand en 1986.

Le contenu des deux premières parties de cette thèse (hormis la preuve de normalisation forte décrite dans le chapitre 7) a par ailleurs été publié dans [50, 51].

[5] En revanche, le résultat établi au chapitre 9 s'instancie immédiatement au calcul implicite dans la mesure où celui-ci est une extension du Calcul des Constructions avec univers, et permet en particulier d'affirmer que la puissance théorique du calcul implicite dépasse strictement — et de loin — celle de la théorie des ensembles (classique) de Zermelo, bien que le calcul implicite ne comporte aucun type inductif primitif.

Première partie

Constructions implicites

Chapitre 1

Arguments implicites et systèmes de types à la Curry

1.1 Une introduction à la théorie des types

1.1.1 Types, termes et jugements

Dans la plupart des langages de programmation fonctionnels,[1] la syntaxe du langage peut être définie essentiellement à partir de deux catégories syntaxiques, qui sont la catégorie syntaxique des *types* et la catégorie syntaxique des *termes* (ou *programmes*).

Chaque type définit une classe d'objets structurés de manière similaire, et qu'il est possible de manipuler à travers un certain nombre d'opérations communes. Dans le langage, les types forment une catégorie syntaxique autonome, en ce sens que chaque type est soit un type élémentaire (comme par exemple le type des entiers ou le type des booléens), soit un type formé à partir d'autres types (comme par exemple le type des listes d'entiers ou le type des tableaux de booléens), mais dans tous les cas, les types sont construits uniquement à partir d'autres types, et le fait même d'être un type est une qualité immédiate qui ne relève que d'un simple problème d'analyse syntaxique.

Là où les types indiquent la *nature* des objets calculés, les termes définissent le calcul proprement dit, c'est à dire la façon dont les diverses opérations s'enchaînent (*i.e.* l'*algorithme*) de manière à produire le résultat recherché (par exemple, une liste triée à partir d'une liste dont les éléments sont rangés dans le désordre). Bien que la catégorie syntaxique des termes soit définie de manière très similaire à la catégorie syntaxique des types — en faisant parfois référence à cette dernière[2] — le critère de correction syntaxique ne suffit pas à établir la bonne formation d'un terme dans le langage. Chacun des termes donnés par la syntaxe doit encore passer l'épreuve de la *vérification de type*, qui permet de dire si toutes les opérations décrites par le programme sont effectuées en accord avec leur type et, à l'issue de ce processus, quel est le type de ce programme (ou plus exactement, quelle est le type de la valeur calculée par ce programme).[3]

[1] Du moins en ce qui concerne les langages typés. Nous ne considérerons pas ici le cas des langages de la famille LISP.

[2] Comme par exemple l'annotation de type sous l'abstraction à la Church ou, de manière plus profonde, l'application de type du système F.

[3] La plupart des langages fonctionnels effectuent une distinction très nette entre *valeurs* et *programmes*,

Formellement, le système de types d'un langage de programmation fonctionnel est organisé autour d'un *jugement de typage* noté

$$\vdash M : A$$

et se lisant « M est un terme de type A », qui est défini par un jeu de règles d'inférences spécifique au langage considéré. D'un point de vue algorithmique, la relation entre termes et types induite par le jugement de typage pose naturellement deux problèmes, qui sont les suivants :

1. Existe-t-il un algorithme capable de déterminer si pour un terme M et un type A arbitraires la relation "$\vdash M : A$" est satisfaite ? (problème de la *vérification de type*)
2. Existe-t-il un algorithme capable de déterminer si un terme M donné est bien typé et, lorsque c'est le cas, de construire un type A tel que la relation "$\vdash M : A$" soit satisfaite ? (problème de l'*inférence de type*)

En pratique, les langages fonctionnels sont généralement conçus autour d'un système de types pour lequel les problèmes de vérification de type et d'inférence de type sont tous les deux décidables. Nous verrons cependant dans la suite de ce chapitre qu'il existe des systèmes de types pour lesquels l'inférence et la vérification de type sont indécidables. Bien que cette caractéristique soit un sérieux obstacle à l'implémentation de ces langages, elle n'en diminue pas pour autant leur intérêt théorique, et constitue généralement le prix à payer pour obtenir un système de types fin. Ce sera en particulier le cas du Calcul des Constructions implicite, auquel sont consacrées les deux premières parties de cette thèse.

1.1.2 Le λ-calcul simplement typé

La plupart des langages fonctionnels typés[4] sont construits autour d'un noyau commun constitué par le *λ-calcul simplement typé*. Ce formalisme, introduit à l'origine par Church [14], est basé sur le système des *types simples*, qui sont formés à partir d'un certain nombre de *types de base* (tels que le type des booléens ou des entiers par exemple) à l'aide d'une unique construction qui à chaque couple de types A et B déjà formés associe un nouveau type noté

$$A \to B,$$

et qui désigne le type des fonctions définies sur A et à valeurs dans B. Formellement, la syntaxe des types simples est donnée par :

Types simples	A, B	$::=$	bool	$\mid$ nat	$\cdots$	(types de base)
		$\mid$	$A \to B$			(types flèche)

Notons que le choix des types de base bool et nat est ici purement arbitraire[5], l'intérêt du système se situant davantage dans la construction flèche. Dans ce qui suit, on se réservera la

qui est liée au fait que ces entités sont représentées dans la machine de manière différente, et généralement dans des segments de mémoire séparés. À mesure que l'on s'achemine vers des langages de programmation de plus en plus évolués — typiquement les systèmes de traitement des démonstrations mathématiques — la différence entre valeurs et programmes tend à s'estomper, le calcul mathématique étant par nature une activité symbolique.

[4] Parmi les exceptions à ce schéma, il est intéressant de mentionner le langage Cayenne [7] qui, contrairement aux langages de la famille ML, est basé sur un système de types dépendants.

[5] Dans le formalisme tel qu'il est présenté à l'origine, les types simples ne comportent qu'un seul type de base, qui est noté 0, et dont la signification n'a pas besoin d'être précisée.

possibilité de modifier l'ensemble des types de base du λ-calcul simplement typé en fonction de nos besoins.

Les termes du λ-calcul simplement typé sont les mêmes que ceux du λ-calcul pur, à cette différence près que la λ-abstraction est annotée par le type de son argument. On y retrouve les trois constructions fondamentales du λ-calcul, à savoir :

- les *variables*, notées x, y, z, etc. ;
- l'*abstraction typée* (où *abstraction à la Church*), notée $\lambda x : A \,.\, M$, qui désigne la fonction associant à tout objet x de type A le terme M (dans lequel figure éventuellement la variable libre x) ;
- l'*application*, notée MN, qui désigne l'application de la fonction M à son argument N.

Formellement, la syntaxe des termes du λ-calcul simplement typé est donc donnée par[6] :

$$\begin{array}{lrcll} \textbf{Termes} & M, N & ::= & x & \text{(variable)} \\ & & | & \lambda x : A \,.\, M & \text{(abstraction)} \\ & & | & M\ N & \text{(application)} \end{array}$$

Comme dans le λ-calcul, on distingue les notions de *variable libre* et de *variable liée*, sachant que toute occurrence libre de la variable x dans le terme M devient une occurrence liée dans le terme $\lambda x : A \,.\, M$. L'ensemble des variables libres d'un terme M est noté $FV(M)$.

Réduction La notion de calcul afférente au λ-calcul simplement typé est définie comme dans le λ-calcul pur par la *β-réduction*, par laquelle l'application d'une fonction $\lambda x : A \,.\, M$ à un argument N se réduit sur le corps de la fonction dans lequel on a substitué l'argument réel N au paramètre formel x, le résultat de cette substitution étant noté $M\{x := N\}$:

$$(\beta) \qquad (\lambda x : A \,.\, M)\ N \quad \triangleright_\beta \quad M\{x := N\}.$$

Dans la relation exprimée ci-dessus, on dit que le terme de gauche est un *β-radical* et que le terme de droite est son *β-contracté*.

Cette relation définie initialement à la racine des termes se propage naturellement à tous leurs sous termes. Ainsi, on note $M \to_\beta M'$ la relation de *β-réduction en une étape*, qui exprime que le terme M' est obtenu en contractant un β-radical quelconque dans un sous-terme du terme M. Le préordre engendré par la relation $M \to_\beta M'$ (*i.e.* sa clôture réflexive et transitive) est noté $M \twoheadrightarrow_\beta M'$ (*β-réduction en un nombre arbitraire d'étapes*, ou plus simplement *β-réduction*), et la relation d'équivalence engendrée par la relation $M \to_\beta M'$ (*i.e.* sa clôture réflexive, symétrique et transitive) est notée $M =_\beta M'$ (*β-conversion*).

Bien que la notion de β-réduction ne précise pas l'ordre dans lequel doivent être effectuées les différentes étapes de β-contraction, le calcul est fondamentalement déterministe, puisqu'il satisfait — à l'instar du λ-calcul pur — la propriété de Church-Rosser et est confluent :

- (*Propriété de Church-Rosser*) Si $M \twoheadrightarrow_\beta M_1$ et $M \twoheadrightarrow_\beta M_2$, alors il existe M' tel que $M_1 \twoheadrightarrow_\beta M'$ et $M_2 \twoheadrightarrow_\beta M'$.
- (*Confluence*) Si $M_1 =_\beta M_2$, alors il existe M' tel que $M_1 \twoheadrightarrow_\beta M'$ et $M_2 \twoheadrightarrow_\beta M'$.

[6] En toute rigueur, il faudrait également ajouter à cette grammaire les différentes constantes habitant les types de base (entiers, booléens, etc.) ainsi que les opérations primitives permettant de manipuler ces valeurs (addition, multiplication, etc.), ce que nous ne ferons pas ici afin de nous concentrer sur le problème essentiel, à savoir la β-réduction.

Lorsqu'un terme M ne contient aucun β-radical (dans ses sous-termes), on dit qu'il est en *forme normale* (ou, plus précisément, en *forme β-normale*). Pour tout terme M, il existe au plus un terme M' en forme normale tel que $M =_\beta M'$ (auquel cas on a $M \twoheadrightarrow_\beta M'$). Lorsqu'un tel terme M' existe, on dit que M' est *la forme normale* du terme M. En termes de programmation, la forme normale d'un terme M, lorsqu'elle existe, correspond (plus ou moins) à la *valeur* calculée par le programme M.[7]

Les règles de typage Les termes du λ-calcul simplement typé tels qu'il sont donnés par la syntaxe ne font pas tous du sens. Par exemple, la fonction identité sur le type A, qui est représentée par le terme $\lambda x : A \,.\, x$, ne peut être appliquée qu'à un terme de type A. Par conséquent, le terme $(\lambda x : A \,.\, x)N$ n'aura un sens que dans la mesure où le terme N est lui-même un terme de type A.

En toute généralité, la relation de typage (*i.e.* le jugement $\vdash M : A$) ne peut pas être définie uniquement sur les termes clos (*i.e.* sur les termes ne comportant aucune variable libre), mais doit être définie également sur les termes ouverts. Pour que cette relation ait un sens, il est nécessaire de paramétrer le jugement de typage par un *contexte de typage* (ou plus simplement *contexte*) dans lequel seront déclarés les types des différentes variables libres du terme M. Formellement, un contexte est une liste finie (non-ordonnée) dont les éléments sont des couples $(x : A)$ déclarant la variable x avec le type A, et dans lequel chaque variable est déclarée au plus une fois :

$$\textbf{Contextes} \qquad \Gamma \quad ::= \quad [x_1 : A_1; \ldots; x_n : A_n] \qquad (n \geq 0, \text{ et } x_i \neq x_j \text{ si } i \neq j)$$

L'ensemble des variables déclarées dans Γ est noté $DV(\Gamma)$, et si $DV(\Gamma_1) \cap DV(\Gamma_2) = \varnothing$, on note $\Gamma_1; \Gamma_2$ le contexte formé par la réunion des contextes Γ_1 et Γ_2.

Dans le cas général, le jugement de typage est une relation ternaire reliant un contexte, un terme et un type, que l'on note $\Gamma \vdash M : A$ et qui se lit : « sous le contexte Γ, le terme M a pour type A ». Cette relation est définie par les trois règles d'inférence suivantes :

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

Dans le cas où le contexte Γ est vide, la relation $[] \vdash M : A$ est notée plus simplement $\vdash M : A$. On remarquera que pour qu'un jugement $\Gamma \vdash M : N$ soit valide, il est nécessaire (mais non suffisant) que chacune des variables libres figurant dans le terme M soit déclarée dans Γ, *i.e.* $FV(M) \subset DV(\Gamma)$. En particulier, un jugement $[] \vdash M : A$ ne peut être dérivable dans le contexte vide que dans la mesure où le terme M est clos.

La relation de typage ainsi définie satisfait la propriété d'*unicité du type*, de même que la propriété de *préservation du type par la β-réduction* (également appelée *β-autoréduction*) :

- (*Unicité du type*) Si $\Gamma \vdash M : A$ et $\Gamma \vdash M : A'$, alors $A = A'$;
- (*β-autoréduction*) Si $\Gamma \vdash M : A$ et $M \to_\beta M'$, alors $\Gamma \vdash M' : A$.

[7] En fait, dans le cadre des langages de programmation fonctionnels, on considère généralement que ce sont les termes en forme normale de tête faible [8] qui représentent les valeurs et non les termes en forme normale, puisque l'évaluation sur laquelle sont basés ces langages est une évaluation faible, qui ne traverse pas les abstractions.

Notons que cette seconde propriété n'exprime rien d'autre que la correction du calcul vis-à-vis des types de données. Enfin, le problème de l'inférence de type est décidable dans le λ-calcul simplement typé

- (*Inférence*) Il existe un algorithme qui, un contexte Γ et un terme M étant donnés, est capable de déterminer s'il existe un type A tel que $\Gamma \vdash M : A$, et de construire ce type (qui est unique) lorsque c'est le cas.

d'où il ressort immédiatement que la vérification de type est décidable également (d'après la propriété d'unicité).

Normalisation forte Dans le λ-calcul pur, il existe des termes qui n'ont pas de forme normale, tels que le terme $\Omega = (\lambda x\,.\,xx)(\lambda x\,.\,xx)$ qui se β-réduit indéfiniment sur lui-même. Le résultat central du λ-calcul simplement typé est le théorème de *normalisation forte*, qui exprime que tout terme M bien typé (dans un contexte quelconque) est *fortement normalisable*, c'est-à-dire que :

1. le terme M a une forme normale ;
2. toute stratégie de réduction appliquée au terme M conduit invariablement à la forme normale de M en un nombre fini d'étapes.

Théorème 1.1.1 — *Si $\Gamma \vdash M : A$, alors M est fortement normalisable.*

1.1.3 L'isomorphisme de Curry-Howard

En plus de son intérêt pratique indéniable dans le domaine de la programmation, le λ-calcul simplement typé est également un outil fondamental en logique, puisqu'il permet de représenter par des termes toutes les preuves du fragment implicationnel du calcul propositionnel intuitionniste. Quitte à ajouter des constructions supplémentaires au langage, cette correspondance entre termes et démonstrations — appelée *correspondance de Curry-Howard* — peut être étendue à tout le calcul propositionnel intuitionniste (dans ce cas précis, il s'agit même d'un isomorphisme) et même au calcul propositionnel classique. Au-delà du calcul propositionnel, la correspondance de Curry-Howard s'étend à une très grande classe de formalismes — calcul des prédicats intuitionniste, logique propositionnelle du second ordre (système F), calcul des prédicats d'ordre supérieur (système $F\omega$, Calcul des Constructions) — et constitue la base de la théorie des types, qui est une famille de formalismes (tous basés sur le λ-calcul typé) caractérisée par une identification complète de la notion de programme avec la notion de démonstration.

L'interprétation de Heyting-Kolmogorov Pour comprendre la portée de la correspondance de Curry-Howard, il faut d'abord se replacer dans le cadre de l'interprétation de Heyting-Kolmogorov [13, 33, 38], qui dresse de manière informelle le cahier des charges de la logique intuitionniste. Insistons sur le fait que cette interprétation est purement informelle, puisqu'elle ne précise ni le langage dans lequel sont construites les "preuves canoniques" ni même la notion de fonction qui est utilisée dans la définition des preuves canoniques de l'implication. D'un point de vue méthodologique, l'interprétation de Heyting-Kolmogorov sert surtout à préciser le sens constructif que les mathématiciens intuitionnistes entendent donner à la notion de démonstration mathématique.

Selon l'interprétation de Heyting-Kolmogorov, la signification d'une proposition A en logique intuitionniste n'est pas donnée par une *valeur de vérité* (ce qui est le point de vue de la logique classique), mais par l'ensemble de ses *preuves*, ou plus exactement par l'ensemble de ses *preuves canoniques*[8] — ensemble que nous noterons ici $\Phi(A)$ — c'est-à-dire par un ensemble d'objets dont la structure même emporte la conviction que la proposition est vraie. Autrement dit, une proposition A est définie précisément par l'ensemble de ses preuves canoniques $\Phi(A)$, et cette proposition est *vraie* (au sens de la logique intuitionniste) lorsqu'il est possible d'exhiber une preuve canonique de A, c'est-à-dire un élément de $\Phi(A)$.

En termes d'ensembles de preuves canoniques, l'interprétation de Heyting-Kolmogorov propose de définir la signification intuitionniste des connecteurs logiques de la manière suivante :

- (*Absurdité*) La formule $\bot$ est la formule qui n'a pas de preuve canonique. L'ensemble des preuves canoniques de la formule $\bot$ est vide, c'est-à-dire :
$$\Phi(\bot) \quad = \quad \varnothing.$$
- (*Conjonction*) Une preuve canonique de la proposition $A \wedge B$ est un couple (a, b) formé par une preuve canonique $a \in \Phi(A)$ et une preuve canonique $b \in \Phi(B)$. L'ensemble des preuves canoniques de $A \wedge B$ est donc le produit cartésien de $\Phi(A)$ par $\Phi(B)$, c'est-à-dire :
$$\Phi(A \wedge B) \quad = \quad \Phi(A) \times \Phi(B).$$
- (*Disjonction*) Une preuve canonique de la proposition $A \vee B$ est soit de la forme $i_1(a)$ avec $a \in \Phi(A)$, soit de la forme $i_2(b)$ avec $b \in \Phi(B)$, où i_1 et i_2 sont les injections qui envoient respectivement les ensembles $\Phi(A)$ et $\Phi(B)$ dans leur union disjointe, qui constitue donc l'ensemble des preuves canoniques de $A \vee B$:
$$\Phi(A \vee B) \quad = \quad \Phi(A) + \Phi(B).$$
- (*Implication*) Une preuve canonique de la proposition $A \Rightarrow B$ est une fonction f associant à toute preuve canonique de A une preuve canonique de B. L'ensemble des preuves canoniques de $A \Rightarrow B$ est donc l'espace des fonctions de $\Phi(A)$ dans $\Phi(B)$, c'est-à-dire :
$$\Phi(A \Rightarrow B) \quad = \quad \Phi(A) \to \Phi(B).$$

Plus généralement, l'interprétation de Heyting-Kolmogorov précise également la signification intuitionniste des quantificateurs universel et existentiel (en notant $\mathcal{D}$ le domaine de la quantification), dont elle définit l'ensemble des preuves canoniques de la manière suivante :

- (*Quantification universelle*) Une preuve canonique de la proposition $\forall x.A(x)$ est une fonction qui à tout objet $x \in \mathcal{D}$ associe une preuve canonique de la proposition $A(x)$. L'ensemble des preuves canoniques de $\forall x.A(x)$ est donc le produit cartésien de la famille d'ensembles $\Phi(A(x))$ lorsque x parcourt $\mathcal{D}$, c'est-à-dire :
$$\Phi\big(\forall x.A(x)\big) \quad = \quad \prod_{x \in \mathcal{D}} \Phi(A(x)).$$

[8] Ici, l'expression « preuve canonique » est employée dans un sens intuitif qui n'a pour le moment rien à voir avec la notion formelle de dérivation. Ce n'est qu'à travers la correspondance de Curry-Howard combinée avec le théorème de normalisation forte (qui est l'équivalent calculatoire de la propriété logique d'élimination des coupures) que l'on pourra s'assurer que dans le contexte vide, les termes (qui représentent les dérivations) ont bien une structure de preuve canonique au sens de l'interprétation de Heyting-Kolmogorov (et que par conséquent, le cahier des charges a bien été rempli). En ce sens, la correspondance de Curry-Howard, et plus généralement la théorie des types, marque l'achèvement du programme intuitionniste.

- (*Quantification existentielle*) Une preuve canonique de la proposition $\exists x.A(x)$ est un couple (x, a) formé par un objet $x \in \mathcal{D}$ et une preuve canonique a de la proposition $A(x)$. L'ensemble des preuves de $\exists x.A(x)$ est donc la somme disjointe de la famille d'ensembles $\Phi(A(x))$ lorsque x parcourt $\mathcal{D}$, c'est-à-dire :

$$\Phi\big(\exists x.A(x)\big) \quad = \quad \sum_{x \in \mathcal{D}} \Phi(A(x)).$$

Techniquement, la logique intuitionniste se distingue de la logique classique par le fait qu'elle refuse le principe du tiers-exclu. En effet, si A est une proposition arbitraire (typiquement si A est une formule atomique), il se peut que l'on ne sache rien dire ni de A, ni de $\neg A$. Comme au sens de l'interprétation de Heyting-Kolmogorov ces deux propositions n'ont pas de preuve canonique, la proposition $A \vee \neg A$ n'a pas de preuve canonique également.[9]

Un fragment du calcul propositionnel intuitionniste Ainsi que nous venons de le voir, l'interprétation de Heyting-Kolmogorov identifie (au moins de manière implicite) chaque formule du calcul propositionnel avec l'ensemble de ses preuves canoniques, qui n'est rien d'autre que le type de données que l'on obtient en remplaçant dans la formule en question chacun des connecteurs $\wedge$, $\vee$ et $\Rightarrow$ par les constructeurs de types $\times$ (produit cartésien), $+$ (union disjointe) et $\to$ (flèche) respectivement.

Comme le λ-calcul simplement typé ne dispose que de la construction flèche $A \to B$ (qui va nous permettre de représenter le type des preuves de la proposition $A \Rightarrow B$), nous allons nous restreindre au fragment implicationnel du calcul propositionnel intuitionniste, c'est-à-dire au fragement du calcul propositionnel intuitionniste dont les formules sont construites uniquement à partir des variables propositionnelles et de l'implication :

Formules	A, B	$::=$	$\alpha \mid \beta \mid \gamma \quad \ldots$	(variables propositionnelles)
		$\mid$	$A \Rightarrow B$	(implication)

La théorie de la démonstration de ce fragment est basée sur les séquents asymétriques de la logique intuitionniste, qui sont des couples hypothèses/conclusion de la forme

Séquents $\qquad \Gamma \vdash A$

où Γ est une liste éventuellement vide de formules[10] (le *subséquent*) et A une formule (le *conséquent*). Ses règles de déduction sont simplement les règles de la déduction naturelle qui ne concernent pas d'autre connecteur que l'implication, c'est-à-dire la règle d'axiome, la règle d'introduction de l'implication et sa règle d'élimination (ou *modus ponens*) :

$$\text{(Ax.)}\ \frac{}{\Gamma \vdash A}\ {\scriptstyle (A \in \Gamma)} \qquad (\Rightarrow\text{-intro})\ \frac{\Gamma; A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad (\Rightarrow\text{-élim})\ \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

[9] On remarquera que si l'on formalise naïvement l'interprétation de Heyting-Kolmogorov en théorie des ensembles — en interprétant les preuves de $A \Rightarrow B$ par les fonctions de $\Phi(A)$ dans $\Phi(B)$ au sens de la théorie des ensembles (en logique classique) — le tiers-exclu est validé par la fonction d'interprétation de manière immédiate. L'affirmation selon laquelle l'interprétation de Heyting-Komogorov ne validerait pas le tiers-exclu ne doit donc pas être prise au pied de la lettre. On notera toutefois que, prise en tant que *modèle* de la logique intuitionniste, l'interprétation de Heyting-Kolmogorov est très sensible au langage dans lequel elle est formalisée : en théorie des ensembles classique, l'interprétation de Heyting-Kolmogorov valide le tiers-exclu alors que dans le λ-calcul simplement typé (étendu par les types somme et produit), elle l'invalide.

[10] Pour des raisons de commodité, on supposera que les listes considérées sont ordonnées, et qu'elles sont susceptibles de contenir un même élément en plusieurs exemplaires.

Afin d'illustrer l'utilisation de ces règles, nous avons représenté dans la figure 1.1 une dérivation complète du séquent $\vdash (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow A \Rightarrow C$, qui exprime la transitivité de l'implication (*i.e.* le syllogisme Barbara).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow B;\ B \Rightarrow C;\ A \vdash B \Rightarrow C} \quad \dfrac{\dfrac{}{A \Rightarrow B;\ B \Rightarrow C;\ A \vdash A \Rightarrow B} \quad \dfrac{}{A \Rightarrow B;\ B \Rightarrow C;\ A \vdash A}}{A \Rightarrow B;\ B \Rightarrow C;\ A \vdash B}}{A \Rightarrow B;\ B \Rightarrow C;\ A \vdash C}}{A \Rightarrow B;\ B \Rightarrow C \vdash A \Rightarrow C}}{A \Rightarrow B \vdash (B \Rightarrow C) \Rightarrow A \Rightarrow C}}{\vdash (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow A \Rightarrow C}$$

FIG. 1.1 – Une dérivation du syllogisme Barbara

Représentation des dérivations par des termes Quitte à se placer dans un λ-calcul simplement typé dans lequel les types de base sont exactement les variables propositionnelles, il n'y a plus de différence entre la notion de type simple et la notion de formule telle que nous l'avons définie ci-dessus. On identifiera donc chaque formule avec le type correspondant, en considérant que les constructions $A \Rightarrow B$ et $A \to B$ sont synonymes.

Un examen rapide des trois règles de déduction de notre fragment de la logique intuitionniste montre qu'il s'agit exactement des règles de typage du λ-calcul simplement typé dans lesquelles on a effacé tout ce qui mentionne les termes, y compris les variables déclarées dans les contextes (qui deviennent ainsi des listes de formules).[11] De ceci, il découle naturellement que toute dérivation de typage dans le λ-calcul simplement typé, lorsqu'on y efface les termes et les variables déclarées dans les contextes, devient une dérivation du séquent correspondant dans le fragment du calcul propositionnel intuitionniste réduit à l'implication. Autrement dit :

> *Si le jugement de typage* $[x_1 : A_1; \ldots; x_n : A_n] \vdash M : B$ *est dérivable dans le* λ*-calcul simplement typé, alors le séquent* $A_1; \ldots; A_n \vdash B$ *est dérivable dans le calcul propositionnel intuitionniste réduit à l'implication.*

Bien entendu, la réciproque est beaucoup plus intéressante, car les dérivations de typage du λ-calcul simplement typé contiennent plus d'informations que les dérivations des séquents de notre fragment de la logique intuitionniste. Cette information supplémentaire est donnée sous la forme d'un terme M (dépendant des variables déclarées dans le contexte), qui est précisément l'expression — au sens de l'interprétation de Heyting-Kolmogorov — de la preuve que l'on est en train de construire. Autrement dit, l'interprétation de Heyting-Kolmogorov propose naturellement de relire chaque jugement de typage $[x_1 : A_1; \ldots; x_n : A_n] \vdash M : B$ de la manière suivante :

> *Sous les hypothèses* $x_1 : A_1, \ldots, x_n : A_n$ (chaque hypothèse étant introduite par une preuve abstraite x_i de la proposition A_i)*, le terme* M *est une preuve de la proposition* B (construite à partir des preuves abstraites $x_1, \ldots, x_n$)*.*

Cette nouvelle lecture permet d'interpréter la correspondance entre les règles de typage du λ-calcul simplement typé et les règles de déduction de notre fragment du calcul propositionnel intuitionniste de la manière suivante :

[11] On remarquera en particulier que la condition de bord $(x : A) \in \Gamma$ dans la règle de typage de la variable x reste conservée dans la règle d'axiome sous la forme de la condition de bord $A \in \Gamma$.

- (*Variable/Axiome*) Sous les hypothèses $x_1 : A_1, \ldots, x_n : A_n$, le terme x_i est une preuve de la proposition A_i.
- (*Abstraction/Introduction de l'implication*) Si sous les hypothèses $x_1 : A_1, \ldots, x_n : A_n$ et $x : A$ le terme M est une preuve de la proposition B, alors sous les hypothèses $x_1 : A_1, \ldots, x_n : A_n$, le terme $\lambda x : A \,.\, M$ est une preuve de la proposition $A \Rightarrow B$.
- (*Application/Élimination de l'implication*) Si sous les hypothèses $x_1 : A_1, \ldots, x_n : A_n$ le terme M est une preuve de la proposition $A \Rightarrow B$, et si sous les mêmes hypothèses le terme N est une preuve de la proposition A, alors sous ces mêmes hypothèses le terme $M\ N$ est une preuve de la proposition B.

Il est maintenant facile de se convaincre que chaque dérivation du calcul propositionnel intuitionniste réduit à l'implication peut être représentée par un terme M du λ-calcul simplement typé en suivant pas-à-pas la correspondance décrite ci-dessus. Par conséquent :

> *Si $A_1; \ldots; A_n \vdash B$ est un séquent dérivable dans le calcul propositionnel intuitionniste réduit à l'implication, et si π est une dérivation particulière de ce séquent, alors il existe un terme M tel que le jugement de typage*
>
> $$[x_1 : A_1; \ldots; x_n : A_n] \ \vdash \ M \ : \ B$$
>
> *est dérivable (en attachant une variable x_i à chaque hypothèse A_i), et dont la dérivation a exactement la même structure que la dérivation π.*

Insistons sur le fait que le terme de preuve associé à une dérivation du calcul propositionnel intuitionniste réduit à l'implication peut être construit par un procédé purement mécanique. À titre d'exemple, on pourra vérifier que le terme de preuve correspondant à la dérivation donnée par la figure 1.1 est précisément le terme

$$\lambda f : (A \to B) \,.\, \lambda g : (B \to C) \,.\, \lambda x : A \,.\, g\ (f\ x) \quad : \quad (A \to B) \to (B \to C) \to A \to C$$

qui représente dans le λ-calcul simplement typé l'opérateur de composition de fonctions.

Extension au calcul propositionnel intuitionniste Ainsi que nous l'avons déjà mentionné, le λ-calcul simplement typé peut être enrichi de manière à étendre l'isomorphisme de Curry-Howard à tout le calcul propositionnel intuitionniste (**LJ**), en ajoutant un type vide $\mathsf{0}$ ainsi que des constructions $A \times B$ (produit cartésien) et $A + B$ (union disjointe). La syntaxe du formalisme étendu et de ses règles de réduction et de typage sont données dans la figure 1.2.

Nous n'irons pas plus loin dans la description et l'étude de ce formalisme. Nous nous contenterons simplement de préciser qu'il satisfait les mêmes propriétés que le λ-calcul simplement typé (préservation du typage par la réduction étendue, normalisation forte des termes bien typés pour toutes les règles de calcul, décidabilité de la vérification et de l'inférence de type), et que ses règles de typage correspondent exactement aux règles de déduction du calcul propositionnel intuitionniste qui s'en déduisent en effaçant les termes ainsi que les variables déclarées dans les contextes (et en identifiant $\bot$ avec $\mathsf{0}$, $A \wedge B$ avec $A \times B$, $A \vee B$ avec $A + B$ et $A \Rightarrow B$ avec $A \to B$).

Syntaxe des types, termes et contextes

Types $A, B ::= \alpha \mid \beta \mid \gamma \mid \cdots \mid 0 \mid A \times B \mid A + B \mid A \to B$

Termes $M, N ::=$
x
$\mid \mathsf{R}_A(M)$
$\mid \langle M; N\rangle \mid \pi_1(M) \mid \pi_2(M)$
$\mid \mathsf{inl}_{A,B}(M) \mid \mathsf{inr}_{A,B}(M) \mid \mathsf{case}\ N\ \mathsf{of}\ \mathsf{inl}(x).M_1 \mid \mathsf{inr}(x).M_2$
$\mid \lambda x : A\,.\,M \mid M\ N$

Contextes $\Gamma ::= [x_1 : A_1; \ldots; x_n : A_n]$

Règles de réduction

$$
\begin{array}{ll}
\pi_1(\langle M_1; M_2\rangle) & \triangleright\ M_1 \\
\pi_2(\langle M_1; M_2\rangle) & \triangleright\ M_2 \\
\mathsf{case}\ \mathsf{inl}_{A,B}(N)\ \mathsf{of}\ \mathsf{inl}(x).M_1 \mid \mathsf{inr}(x).M_2 & \triangleright\ M_1\{x := N\} \\
\mathsf{case}\ \mathsf{inr}_{A,B}(N)\ \mathsf{of}\ \mathsf{inl}(x).M_1 \mid \mathsf{inr}(x).M_2 & \triangleright\ M_2\{x := N\} \\
(\lambda x : A\,.\,M)\ N & \triangleright\ M\{x := N\}
\end{array}
$$

Règles de typage

(Variable) $$\frac{}{\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma}$$

(0-élim) $$\frac{\Gamma \vdash M : 0}{\Gamma \vdash \mathsf{R}_A(M) : A}$$

(×-intro) $$\frac{\Gamma \vdash M_1 : A \qquad \Gamma \vdash M_2 : A}{\Gamma \vdash \langle M_1; M_2\rangle : A \times B}$$

(×-élim$_1$, ×-élim$_2$) $$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1(M) : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2(M) : B}$$

(+-intro$_1$, +-intro$_2$) $$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{inl}_{A,B}(M) : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \mathsf{inr}_{A,B}(M) : A + B}$$

(+-élim) $$\frac{\Gamma; [x : A] \vdash M_1 : C \qquad \Gamma; [x : B] \vdash M_2 : C \qquad \Gamma \vdash N : A + B}{\Gamma \vdash \mathsf{case}\ N\ \mathsf{of}\ \mathsf{inl}(x).M_1 \mid \mathsf{inr}(x).M_2 : C}$$

(→-intro) $$\frac{\Gamma; [x : A] \vdash M : B}{\Gamma \vdash \lambda x : A\,.\,M : A \to B}$$

(→-élim) $$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

FIG. 1.2 – Une extension du λ-calcul simplement typé correspondant à **LJ**

1.1.4 Les types dépendants

En pratique, les systèmes de types des langages de programmation fonctionnels permettent de détecter un grand nombre d'erreurs au moment de la compilation, et constituent pour cette raison une aide précieuse pour le programmeur. Dans les langages basés sur le système des types simples (tels que le langage ML par exemple), certaines erreurs ne peuvent pas être détectées par le typage, telles que l'accès à un élément d'une liste ou d'un tableau en dehors de ses bornes. Par exemple, si M est une expression de type $\mathsf{list}(A)$ — le type des listes d'objets de type A — il n'est pas possible de déterminer par le typage si un accès à son troisième élément est une opération correcte ou non : pour cela, il faudrait connaître à l'avance la longueur de la liste, ce que le type du terme M n'indique pas.

Pour résoudre ce problème, il est naturel d'incorporer l'information manquante au système de types, en l'enrichissant par exemple avec une nouvelle construction notée

$$\mathsf{vect}(A, N)$$

qui désigne le type des *vecteurs* de longueur N à valeurs dans A. Contrairement aux types que nous avons introduits jusqu'ici — lesquels étaient formés uniquement à partir d'autres types — le type $\mathsf{vect}(A, N)$ dépend d'un terme N de type nat, d'où la terminologie de *type dépendant*. À ce stade, il est important de remarquer que le type $\mathsf{vect}(A, N)$ est très similaire à l'expression de type du langage Pascal

$$\texttt{array [1..}n\texttt{] of } A$$

à cette différence près qu'en Pascal, l'entier naturel n est donné sous la forme d'une *constante* connue au moment de la compilation, alors que dans le cas du type dépendant $\mathsf{vect}(A, N)$, l'entier N peut être donné par une expression arbitraire (c'est-à-dire un terme) de type nat, dont la valeur pourra varier suivant le contexte d'exécution.[12]

En pratique, l'introduction des types dépendants bouleverse considérablement la structure du système de types, et rend caduc le schéma général que nous avons exposé au tout début de ce chapitre. Dans un système de types dépendants, la notion de type ne peut plus être définie uniquement à l'aide d'une description purement syntaxique (comme c'était le cas jusqu'ici), puisqu'il faut également s'assurer que chacune des dépendances a le bon type. En reprenant l'exemple ci-dessus, il est clair que l'expression de type $\mathsf{vect}(A, N)$ ne pourra avoir un sens que dans la mesure où on a vérifié que la dépendance donnée par le terme N a le bon type, à savoir le type des entiers naturels.

Pour cette raison, il est nécessaire d'introduire aux côtés du jugement de typage $\Gamma \vdash M : A$ un nouveau jugement, appelé *jugement de bonne formation de type* (et noté $\Gamma \vdash A \ \ \mathsf{type}$), dont la fonction est de vérifier que les contraintes de typage induites par la présence des dépendances dans les types sont satisfaites. Comme en outre les contextes de typage sont eux-mêmes construits à partir des types, aux deux jugements précédents s'ajoute un troisième jugement, appelé *jugement de bonne formation de contexte* (et noté $\Gamma \vdash$), qui remplit la même fonction pour les contextes. Nous ne décrirons pas davantage ces trois jugements dont la définition précise sera donnée au moment où nous introduirons la notion de système de types purs (*cf* section 1.2).

[12] Ce qui est typiquement le cas si le type $\mathsf{vect}(A, n)$ est utilisé dans le corps d'une fonction dont l'entier n est un des arguments. Dans tout ce qui suit, nous ne considérons que le cas des langages fonctionnels purs, ce qui nous évitera de devoir traiter la question épineuse des restrictions qu'il est nécessaire d'imposer au système de types dépendants dans un langage avec effets de bord [65].

Le produit dépendant Dans un système de types dépendants, la construction flèche du λ-calcul simplement typé ne suffit plus à exprimer correctement la fonctionnalité des termes, et il faut la remplacer par une construction plus générale, appelée *produit dépendant*. Pour comprendre ce point, considérons par exemple une fonction $\mathsf{vect\_create}$ qui à tout entier naturel $n : \mathsf{nat}$ associe un vecteur d'entiers relatifs contenant n zéros :

$$\mathsf{vect\_create}(n) \quad = \quad [\underbrace{0; \ldots; 0}_{n \text{ fois}}]$$

Dans cet exemple, il n'est pas possible d'utiliser la construction flèche pour exprimer le type de la fonction $\mathsf{vect\_create}$, car le type de son résulat (*i.e.* le type $\mathsf{vect}(\mathsf{int}, n)$) dépend de l'argument de la fonction (*i.e.* l'entier $n : \mathsf{nat}$). Pour exprimer le type de cette fonction, on utilise un produit dépendant

$$\mathsf{vect\_create} \quad : \quad \Pi n : \mathsf{nat} \,.\, \mathsf{vect}(\mathsf{int}, n)$$

qui exprime ici que la fonction $\mathsf{vect\_create}$ associe à tout entier $n : \mathsf{nat}$ un vecteur d'entiers relatifs de longueur n (*i.e.* un objet de type $\mathsf{vect}(\mathsf{int}, n)$).

Plus généralement, on introduit dans le système de types une nouvelle construction appelée *produit dépendant* et notée

$$\Pi x : A \,.\, B,$$

qui désigne le type des fonctions associant à tout objet x de type A un objet de type B, sachant que le type B peut dépendre de la variable x.[13] Remarquons que dans le cas où le type B ne dépend pas de la variable x (c'est-à-dire lorsque $x \notin FV(B)$), le produit dépendant $\Pi x : A \,.\, B$ est exactement le type des fonctions de A dans B. Pour cette raison, on considérera par la suite que la notation flèche n'est qu'une abréviation du produit dépendant

$$A \to B \quad \equiv \quad \Pi x : A \,.\, B \qquad\qquad (\text{si } x \notin FV(B))$$

dans le cas où x ne figure pas parmi les variables libres du type B.

À l'aide du produit dépendant, il est possible d'exprimer le type de nombreuses fonctions, telles que par exemple la fonction de concaténation de vecteurs :

$$\mathsf{append}_A \quad : \quad \Pi n_1 : \mathsf{nat} \,.\, \mathsf{vect}(A, n_1) \to \Pi n_2 : \mathsf{nat} \,.\, \mathsf{vect}(A, n_2) \to \mathsf{vect}(A, n_1 + n_2).$$

On remarquera que la fonction ci-dessus prend quatre arguments, à savoir un entier n_1, un vecteur de longueur n_1, un entier n_2 et un vecteur de longueur n_2, pour retourner un vecteur de longueur $n_1 + n_2$. Dans un langage de programmation, il serait sans doute plus commode de ne donner à cette fonction que ses deux arguments principaux — les deux vecteurs — et de faire en sorte que le système se débrouille pour retrouver à partir du contexte les entiers n_1 et n_2. Nous aurons l'occasion de revenir plus dans le détail à la fin de ce chapitre sur cette imperfection de la syntaxe, qui est ici liée à la présence de types dépendants.

[13] Il serait sans doute plus parlant d'écrire $\Pi x : A \,.\, B(x)$. L'utilisation du symbole Π provient de la notation $\prod_{x \in A} B_x$ qui, en théorie des ensembles, désigne le produit cartésien d'une famille d'ensembles $(B_x)_{x \in A}$ indicée par un ensemble A. Au contenu calculatoire près, le produit dépendant de la théorie des types et le produit cartésien généralisé de la théorie des ensembles ont exactement la même signification.

Typage de l'abstraction et de l'application, règle de conversion Vis-à-vis du jugement de typage, l'introduction des types dépendants nécessite de modifier les règles de typage de l'abstraction et de l'application. La modification est mineure pour l'abstraction

$$\frac{\Gamma; [x : A] \vdash M : B}{\Gamma \vdash \lambda x : A \,.\, M : \Pi x : A \,.\, B}$$

puisqu'il s'agit simplement de remplacer la construction flèche par le produit dépendant, qui tient compte du fait que la variable x est susceptible d'apparaître dans le type B en introduisant la liaison nécessaire. Pour l'application en revanche

$$\frac{\Gamma \vdash M : \Pi x : A \,.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B\{x := N\}}$$

il est nécessaire de substituer l'argument N à la variable x dans le type du résultat, de manière à exprimer la dépendance du type du résultat par rapport à l'argument lui-même. (Bien entendu, cette substitution est inopérante dans le cas non-dépendant, dans lequel on retrouve le comportement habituel de la règle de typage de l'application.)

La présence de termes dans les types nécessite d'introduire une règle de typage supplémentaire pour prendre en compte les calculs qui peuvent être effectués dans ces termes. Par exemple, si v_1 et v_2 sont des vecteurs de types $\mathsf{vect}(A, 3)$ et $\mathsf{vect}(A, 4)$ respectivement, la concaténation de ces vecteurs au moyen de la fonction append introduite ci-dessus

$$\mathsf{append}_A \ 3 \ v_1 \ 4 \ v_2 \quad : \quad \mathsf{vect}(A, 3 + 4)$$

retourne un résultat de type $\mathsf{vect}(A, 3+4)$. Pour obtenir un résultat de type $\mathsf{vect}(A, 7)$, il suffit d'identifier les types qui sont équivalents par le calcul (lequel intervient dans les dépendances) en introduisant une nouvelle règle de typage, appelée *règle de conversion* :

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' \ \mathsf{type}}{\Gamma \vdash M : A'} \ {\scriptstyle (A = A')}$$

(la condition de bord $A = A'$ exprimant ici que les types A et A' sont convertibles pour la notion de calcul considérée).

Dans la plupart des systèmes de types dépendants[14], la décidabilité de la vérification et de l'inférence de type est conditionnée par la décidabilité du test de conversion de types $A = A'$. En pratique, une condition suffisante pour obtenir cette propriété est que tous les calculs terminent (*i.e.* propriété de normalisation).

Les quantificateurs D'un point de vue logique, le produit dépendant $\Pi x : A \,.\, B$ correspond à l'interprétation de la quantification universelle $\forall x \in A.\, B$ au sens de Heyting et Kolmogorov (*cf* paragraphe 1.1.3). Pour cette raison, les types dépendants permettent d'étendre la correspondance de Curry-Howard au calcul des prédicats intuitionniste du premier ordre (du moins en ce qui concerne son fragment restreint aux connecteurs logiques et à la quantification universelle). Dans ce cadre, la proposition

$$(\forall x \,.\, P(x)) \quad \Rightarrow \quad (\forall x \,.\, Q(x)) \quad \Rightarrow \quad \forall x \,.\, (P(x) \wedge Q(x))$$

[14] Du moins dans les présentations à la Church de ces systèmes.

est traduite par le type de données

$$(\Pi x : D \,.\, P(x)) \quad \to \quad (\Pi x : D \,.\, Q(x)) \quad \to \quad \Pi x : D \,.\, (P(x) \times Q(x))$$

en notant D le type représentant le domaine sur lequel porte les quantifications, et en introduisant deux familles $P(x)$ et $Q(x)$ de types dépendant du terme $x : D$. Par la correspondance de Curry-Howard, une preuve de cette proposition est donnée par le terme suivant :

$$\lambda f : (\Pi x : D \,.\, P(x)) \,.\, \lambda g : (\Pi x : D \,.\, Q(x)) \,.\, \lambda x : D \,.\, \langle f(x); g(x) \rangle.$$

Afin de compléter la correspondance vis-à-vis du calcul des prédicats intuitionniste, il est possible également d'introduire un type *somme dépendante* notée

$$\Sigma x : A \,.\, B$$

(où x figure éventuellement parmi les variables libres de B) qui désigne le type des couples $\langle a; b \rangle$ formés par un objet a de type A et un objet b de type $B\{x := a\}$,[15] et qui permet d'interpréter la quantification existentielle intuitionniste $\exists x \in A \,.\, B$. On remarquera également que dans le cas non-dépendant, la somme dépendante $\Sigma x : A \,.\, B$ correspond au produit cartésien :

$$A \times B \quad \equiv \quad \Sigma x : A \,.\, B \qquad\qquad (\text{si } x \notin FV(B))$$

Dans la suite de cette thèse, nous nous placerons systématiquement dans des systèmes de types imprédicatifs dans lesquels il est possible de redéfinir la somme dépendante (de même que le produit cartésien et la somme disjointe de deux types) uniquement à l'aide du produit dépendant, qui est pour cette raison la seule construction de type primitive à laquelle nous nous intéresserons désormais.

1.1.5 Un type de tous les types

Un type dépendant n'est jamais défini de manière isolée, mais est toujours introduit à partir d'une construction définissant une *famille de types* indicée par les éléments d'un type donné.[16] Par exemple, le type $\mathsf{vect}(\mathsf{int}, 7)$ est un élément particulier de la famille de types $\mathsf{vect}(\mathsf{int}, n)$ indicée par la variable $n : \mathsf{nat}$.

Pour pouvoir manipuler plus facilement cette notion de famille de types, il est commode d'introduire dans le système un nouveau type noté Type désignant précisément le *type des types*. Ceci permet de considérer chaque famille de types $T(n)$ indicée par le type des entiers naturels (ou par tout autre type plus généralement) comme une fonction des entiers naturels dans les types, c'est-à-dire :

$$T \quad : \quad \mathsf{nat} \to \mathsf{Type}.$$

Si on accorde à la constante Type elle-même le statut de type (c'est-à-dire en supposant $\mathsf{Type} : \mathsf{Type}$), on peut même définir des familles de types indicées par le type de tous les types, ce qui nous permet de considérer chaque type de la forme $\mathsf{vect}(A, n)$ comme une instance particulière de la famille

$$\mathsf{vect} \quad : \quad \mathsf{Type} \to \mathsf{nat} \to \mathsf{Type}$$

dont les deux indices (ou arguments) parcourent respectivement le type des types (Type) et le type des entiers naturels (nat).

[15] La somme dépendante $\Sigma x : A \,.\, B$ correspond en théorie des ensembles à l'union disjointe d'une famille d'ensembles, notée généralement $\Sigma_{x \in A} B_x$.

[16] À travers la correspondance de Curry-Howard, la notion de *famille de types* (indicée par les éléments d'un autre type) correspond à la notion de prédicat en logique du premier ordre.

Les types comme citoyens de première classe D'un point de vue théorique, l'introduction de la constante Type permet d'abolir définitivement la distinction syntaxique entre les termes et les types, qu'il est désormais possible de présenter d'une manière uniforme. En pratique, le formalisme gagne en concision ainsi que le montre sa syntaxe qui est donnée par :

$$\begin{array}{lrcll} \textbf{Termes} & M, N, T, U & ::= & x & \text{(variable)} \\ & & \mid & \mathsf{Type} & \text{(type des types)} \\ & & \mid & \Pi x : T \,.\, U & \text{(produit dépendant)} \\ & & \mid & \lambda x : T \,.\, M & \text{(abstraction)} \\ & & \mid & M\ N & \text{(application)} \end{array}$$

Dans ce système, il n'est plus nécessaire de recourir au jugement de bonne formation de type $\Gamma \vdash T \ \ \mathsf{type}$ qui peut être remplacé par le jugement de typage $\Gamma \vdash T : \mathsf{Type}$. Comme le produit dépendant fait désormais partie de la syntaxe des termes, il faut ajouter une règle supplémentaire pour assurer sa bonne formation, laquelle est donnée par :

$$\frac{\Gamma; [x : T] \vdash U \,:\, \mathsf{Type}}{\Gamma \vdash \Pi x : T \,.\, U \,:\, \mathsf{Type}}$$

Un système très (trop ?) expressif Le système que nous venons de présenter a été introduit à l'origine par P. Martin-Löf [45] pour formaliser les mathématiques constructives, sur la base d'une identification complète de la notion de proposition avec la notion de type (ce qui revient à considérer le type Type comme le type des propositions). C'est un système très expressif dans lequel il est possible de redéfinir tous les connecteurs logiques ainsi que les quantifications existentielle et universelle (cette dernière étant donnée par le produit dépendant) de même que de nombreuses constructions mathématiques (telles que le type des entiers naturels ou des réels constructifs). Dans ce système par exemple, la proposition absurde $\bot$ est représentée par le type $\Pi A : \mathsf{Type} \,.\, A$, qui exprime exactement le « *ex falso sequitur quod libet* ».[17]

Malheureusement, ce système de types est trop expressif d'un point de vue logique, car le formalisme permet de construire un terme de preuve de la proposition $\bot \equiv \Pi A : \mathsf{Type} \,.\, A$ ainsi que l'a démontré J.-Y. Girard dans [28].[18] En particulier, les preuves de la proposition $\bot$ (construites dans le contexte vide) n'ont pas de forme normale (ni même de forme normale de tête), ce qui montre que le système $\mathsf{Type} : \mathsf{Type}$ ne satisfait pas la propriété de normalisation attendue. Bien que la logique exprimée par ce système soit incohérente, le langage de programmation sous-jacent n'en demeure pas moins d'une expressivité remarquable, ce qui fait de lui une base très intéressante pour la conception d'un langage de programmation fonctionnel avec types dépendants.

Sur le plan de la logique, une façon naturelle de corriger le "défaut" de $\mathsf{Type} : \mathsf{Type}$ consiste à introduire non pas une unique constante Type regroupant tous les types, mais plusieurs

[17] Si la proposition $\Pi A : \mathsf{Type} \,.\, A$ est prouvable par un terme M, alors toute proposition A est prouvable par le terme MA : « du faux déduis ce qu'il te plait ».

[18] Il est intéressant de préciser que la preuve d'incohérence n'est pas triviale contrairement à ce que pourrait laisser penser l'axiome $\mathsf{Type} : \mathsf{Type}$. L'analogie avec la théorie munie d'un "ensemble de tous les ensembles" ne fonctionne pas, car le système $\mathsf{Type} : \mathsf{Type}$ ne dispose pas d'un schéma de compréhension qui permettrait d'obtenir de manière directe le paradoxe de Russell. De fait, le paradoxe de Girard est établi dans un système de types différent — le système U — qui ne comporte pas d'axiome $\mathsf{Type} : \mathsf{Type}$, mais dont les dérivations se traduisent de manière immédiate dans $\mathsf{Type} : \mathsf{Type}$. Nous aurons l'occasion de revenir sur la question des paradoxes dans le système U dans le chapitre 8 qui est consacré à cette question.

constantes (par exemple Prop et Type) représentent chacune une certaine classe de types, ce qui nous amène naturellement à la notion de système de types purs.

1.2 Systèmes de types purs

Historiquement, la notion de système de type pur (PTS[19]) a été dégagée par Berardi et Barendregt [9] en remarquant un grand nombre de similitudes formelles partagées par des formalismes aussi divers que le λ-calcul simplement typé, le système F, le *Logical Framework* de Martin-Löf [52] ou le Calcul des Constructions [16, 20]. Un des résultats les plus remarquables de la théorie des PTS est sans doute la structuration d'un ensemble de huit formalismes parmi les plus courants sous la forme d'un cube — le cube de Barendregt — dans lequel chacune des trois directions spatiales figurent une forme particulière de dépendance entre les termes et les types. D'un point de vue technique, la notion de PTS permet également de factoriser la preuve des principales propriétés métathéoriques communes à un grand nombre de systèmes de types, telles que la propriété de préservation du typage par la β-réduction (encore appelée *β-autoréduction*) pour ne citer qu'un exemple.

1.2.1 Le formalisme

Termes, types et sortes Comme pour le système Type : Type (*cf* paragraphe 1.1.5), la théorie des PTS n'effectue aucune distinction syntaxique entre les termes et les types, qui sont regroupés dans une même catégorie syntaxique. Dans ce cadre, les types sont des termes comme les autres, et sont susceptibles d'être passés en argument à certaines fonctions tout comme il peuvent être retournés par d'autres fonctions. En particulier, le jugement de typage des PTS met en relation deux objets — un terme et son type — qui appartiennent tous les deux à la même catégorie syntaxique.

Dans un PTS, les types sont distingués des autres termes par la relation de typage, au moyen d'un ensemble de constantes primitives appelées *sortes* (ou *univers*). Les sortes désignent les *types des types*, et on dira par définition qu'un terme est un *type* (dans un contexte donné) lorsque son type est une sorte. Une propriété essentielle des PTS est que pour tout jugement dérivable $\vdash M : T$ (« M est un terme de type T »), ou bien T est une sorte (auquel cas M est un type), ou bien il existe une sorte s telle que le jugement de typage $\vdash T : s$ est dérivable (ce qui signifie que T est effectivement un type au sens de la définition précédente).

Axiomes et règles Pour définir la structure d'un PTS, il ne suffit pas de donner l'ensemble de ses sortes, mais il faut encore préciser comment ces sortes sont agencées entre elles. Pour cela, chaque PTS est muni d'un ensemble d'*axiomes* et d'un ensemble de *règles*, qui précisent respectivement :

- pour quels couples de sortes (s_1, s_2) on a la relation $\vdash s_1 : s_2$;
- pour quels triplets de sortes (s_1, s_2, s_3) le produit dépendant $\Pi x : T \,.\, U$, lorsque ses composantes T et U sont formées dans les sortes s_1 et s_2 respectivement, est lui-même un type formé dans la sorte s_3.

Ces quelques points de repère étant posés, nous pouvons maintenant définir formellement ce qu'est un PTS.

[19] *Pure Type System* en anglais

Définition 1.2.1 (PTS) — On appelle *système de types purs* (ou PTS en abrégé) tout triplet $L = (\mathcal{S}, \mathbf{Axiom}, \mathbf{Rule})$ où

- $\mathcal{S}$ est un ensemble (fini ou dénombrable) de constantes appelées *sortes* de L ;
- $\mathbf{Axiom} \subset \mathcal{S}^2$ est un ensemble de couples de sortes, appelés *axiomes* de L ;
- $\mathbf{Rule} \subset \mathcal{S}^3$ est un ensemble de triplets de sortes, appelés *règles* de L.

Par exemple, le système $\mathsf{Type} : \mathsf{Type}$ est un PTS caractérisé par les ensembles

$$\mathcal{S} \;=\; \{\mathsf{Type}\}; \qquad \mathbf{Axiom} \;=\; \{(\mathsf{Type} : \mathsf{Type})\} \qquad \text{et} \qquad \mathbf{Rule} = \{(\mathsf{Type}, \mathsf{Type}, \mathsf{Type})\}.$$

Syntaxe des PTS Les *termes* d'un PTS basé sur l'ensemble de sortes $\mathcal{S}$ sont construits à partir des variables (notées x, y, z, etc.) et des sortes (notées s, s', etc.) au moyen du *produit dépendant* $\Pi x : T \,.\, U$, de l'*abstraction* $\lambda x : T \,.\, M$ et de l'*application* $M\ N$:

$$\begin{array}{lrcll} \textbf{Termes} & M, N, T, U & ::= & x & \text{(variable)} \\ & & \mid & s & \text{(sorte)} \\ & & \mid & \Pi x : T \,.\, U & \text{(produit dépendant)} \\ & & \mid & \lambda x : T \,.\, M & \text{(abstraction)} \\ & & \mid & M\ N & \text{(application)} \end{array}$$

Les notions de variables libres et liées sont définies d'une manière très similaire au λ-calcul simplement typé. En particulier, les constructions $\Pi x : T \,.\, U$ et $\lambda x : T \,.\, M$ lient chacune des occurrences libres de la variable x dans le terme U (resp. dans le terme M), mais aucune des occurrences libres de la variable x dans le terme T. (Autrement dit, les occurrences libres de la variable x dans le terme T restent libres dans les termes $\Pi x : T \,.\, U$ et $\lambda x : T \,.\, M$.) De même, on note $M\{x := N\}$ le terme obtenu en substituant le terme N à chacune des occurrences libres de la variable x dans le terme M (en effectuant des renommages de variables liées si nécessaire).

Réduction Le calcul dans les PTS est organisé autour de la β-réduction, qui est définie de la même manière que dans le λ-calcul simplement typé, à savoir par

$$(\lambda x : T \,.\, M)N \quad \triangleright_\beta \quad M\{x := N\}.$$

De même, on utilise les notations $M \to_\beta M'$, $M \twoheadrightarrow_\beta M'$ et $M =_\beta M'$ pour désigner respectivement les relations de *β-réduction en une étape*, de *β-réduction* (en un nombre arbitraire d'étapes) et de *β-conversion*. À l'instar du λ-calcul simplement typé, la β-réduction dans les PTS satisfait la propriété de Church-Rosser et est confluente (*cf* paragraphe 1.1.2).

Règles de typage Les contextes (de typage) des PTS sont des listes finies et ordonnées de couples $(x : T)$ déclarant la variable x avec le type T :

$$\textbf{Contextes} \qquad \Gamma \quad ::= \quad [x_1 : T_1; \ldots; x_n : T_n] \qquad (n \geq 0)$$

Notons que contrairement au λ-calcul simplement typé, l'ordre dans lequel apparaissent les déclarations est important, car chaque type T figurant dans une déclaration $(x : T)$ est susceptible de contenir lui-même des variables libres, ce qui ne fait du sens que dans la mesure où ces variables libres ont été déclarées auparavant. Le système de typage d'un PTS $L = (\mathcal{S}, \mathbf{Axiom}, \mathbf{Rule})$ est organisé autour de deux jugements, qui sont

– le jugement $\Gamma \vdash$ (« le contexte Γ est bien formé ») ;
– le jugement $\Gamma \vdash M : T$ (« sous le contexte Γ, le terme M a pour type T »).

Les relations $\Gamma \vdash$ (bonne formation de contexte) et $\Gamma \vdash M : T$ (typage) sont définies de manière mutuellement récursive par les règles de typage indiquées par la figure 1.3. On notera que la règle (Sort) (typage des sortes) fait référence à l'ensemble **Axiom** tandis que la règle (Prd) (formation des produits dépendants) fait référence à l'ensemble (Rule). De plus, la règle (Lam) comporte une seconde prémisse, qui impose que le type $\Pi x : T \,.\, U$ de l'abstraction $\lambda x : T \,.\, M$ ait un sens dans le cadre du PTS considéré (à savoir qu'il est lui-même un type bien formé).

Règles de bonne formation de contexte

$$\frac{}{[] \vdash} \text{ (WF-E)} \qquad \frac{\Gamma \vdash T : s \qquad x \notin DV(\Gamma)}{\Gamma; [x : T] \vdash} \text{ (WF-S)}$$

Règles de typage

$$\frac{\Gamma \vdash \qquad (x : T) \in \Gamma}{\Gamma \vdash x : T} \text{ (Var)} \qquad \frac{\Gamma \vdash \qquad (s, s') \in \textbf{Axiom}}{\Gamma \vdash s : s'} \text{ (Sort)}$$

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma; [x : T] \vdash U : s_2 \qquad (s_1, s_2, s_3) \in \textbf{Rule}}{\Gamma \vdash \Pi x : T \,.\, U : s_3} \text{ (Prd)}$$

$$\frac{\Gamma; [x : T] \vdash M : U \qquad \Gamma \vdash \Pi x : T \,.\, U : s}{\Gamma \vdash \lambda x : T \,.\, M : \Pi x : T \,.\, U} \text{ (Lam)} \qquad \frac{\Gamma \vdash M : \Pi x : T \,.\, U \qquad \Gamma \vdash N : T}{\Gamma \vdash M\ N : U\{x := N\}} \text{ (App)}$$

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T' : s \qquad T =_\beta T'}{\Gamma \vdash M : T'} \text{ (Conv)}$$

Fig. 1.3 – Règles de typage dans les PTS

Propriétés du typage Le système de typage des PTS jouit de bonnes propriétés [26, 10], dont les principales sont les suivantes :

- (*Bonne formation de contexte*) Si le jugement de typage $\Gamma \vdash M : T$ est dérivable, alors le contexte Γ est bien formé.
- (*Déclaration des variables libres*) Si $\Gamma \vdash M : T$, alors toutes les variables libres de M et de T sont déclarées dans Γ. De plus, si $\Gamma = [x_1 : T_1; \ldots; x_n : T_n]$ est un contexte bien formé, toutes les variables libres du terme T_i (pour $i = 1, \ldots, n$) figurent parmi les variables $x_1, \ldots, x_{i-1}$ (c'est-à-dire parmi les variables déclarées dans le même contexte auparavant).
- (*Affaiblissement*) Si le jugement $\Gamma \vdash M : T$ est dérivable, alors pour tout contexte Γ' bien formé contenant au moins toutes les déclarations effectuées dans Γ (ce que l'on note $\Gamma \subset \Gamma'$), le jugement $\Gamma' \vdash M : T$ est dérivable également.

- (*Substitutivité*) Soient Γ_1 un contexte, N_0 et T_0 des termes tels que $\Gamma_1 \vdash N_0 : T_0$.
  1. Si $\Gamma_1; [x_0 : T_0]; \Gamma_2$ est un contexte bien formé, alors $\Gamma_1; (\Gamma_2\{x_0 := N_0\})$ est un contexte bien formé ;
  2. Si le jugement de typage $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash M : T$ est dérivable, alors le jugement de typage $\Gamma_1; (\Gamma_2\{x_0 := N_0\}) \vdash M\{x_0 := N_0\} : T\{x_0 := N_0\}$ est dérivable.
- (*Types des types*) Si le jugement $\Gamma \vdash M : T$ est dérivable, alors ou bien T est une sorte, ou bien il existe une sorte s telle que $\Gamma \vdash T : s$.
- (β-*autoréduction*) Si le jugement $\Gamma \vdash M : T$ est dérivable, et si $M \to_\beta M'$, alors le jugement $\Gamma \vdash M' : T$ est dérivable également.

Normalisation et décidabilité du typage Contrairement au λ-calcul simplement typé, de nombreux PTS ne satisfont pas la propriété de normalisation forte ni même la propriété de normalisation faible[20] : c'est le cas en particulier du système $\mathsf{Type} : \mathsf{Type}$ (*cf* paragraphe 1.1.5) et des systèmes U et U^-, qui sont tous les trois des systèmes logiquement incohérents.

Dans les PTS satisfaisant la propriété de normalisation forte en revanche (et même dans ceux qui ne satisfont que la propriété de normalisation faible), les problèmes d'inférence et de vérification de type sont tous les deux décidables. Plus précisément, les PTS fortement normalisants satisfont les propriétés suivantes :

- (*Vérification de contexte*) Il existe un algorithme qui permet de déterminer si un contexte Γ est bien formé ou non.
- (*Vérification de type*) Il existe un algorithme qui, pour un contexte Γ et deux termes M, T donnés, permet de déterminer si le jugement $\Gamma \vdash M : T$ est dérivable.
- (*Inférence de type*) Il existe un algorithme qui, pour un contexte Γ et un terme M donnés, permet de déterminer s'il existe un terme T tel que le jugement $\Gamma \vdash M : T$ est dérivable, et de construire ce terme T le cas échéant.

Par ailleurs, les *PTS logiques* (cette notion sera définie formellement au chapitre 9) qui satisfont la propriété de normalisation (faible ou forte) sont des systèmes logiquement cohérents. Contentons-nous pour le moment de préciser que les systèmes du cube que nous allons introduire dans le paragraphe suivant sont tous des systèmes logiques, qui satisfont les propriétés de normalisation forte et de cohérence.

1.2.2 Les systèmes du cube

Les systèmes du cube sont une famille de huit PTS construits sur un ensemble de deux sortes, notées Prop et Type,[21] et un unique axiome ($\mathsf{Prop} : \mathsf{Type}$) :

$$\mathcal{S} \quad = \quad \{\mathsf{Prop};\ \mathsf{Type}\} \qquad \text{et} \qquad \mathbf{Axiom} \quad = \quad \{(\mathsf{Prop} : \mathsf{Type})\}.$$

Les règles de formation de ces huit PTS sont toutes de la forme (s_1, s_2, s_2) — c'est-à-dire des triplets de sortes dont les deuxième et troisième composantes sont identiques — et sont

[20] Précisons qu'à l'heure actuelle, on ne sait toujours pas s'il existe des PTS satisfaisant la propriété de normalisation faible (*i.e.* tous les termes bien typés ont une forme normale) sans satisfaire la propriété de normalisation forte.

[21] Dans la littérature sur les PTS, les sortes Prop et Type sont fréquemment notées $*$ ("*star*") et $\Box$ ("*box*") respectivement, et la sorte Prop est parfois notée Set. Dans un souci de cohérence avec les notations utilisées dans la suite de cette thèse, nous utiliserons ici les notations Prop et Type pour désigner ces deux sortes.

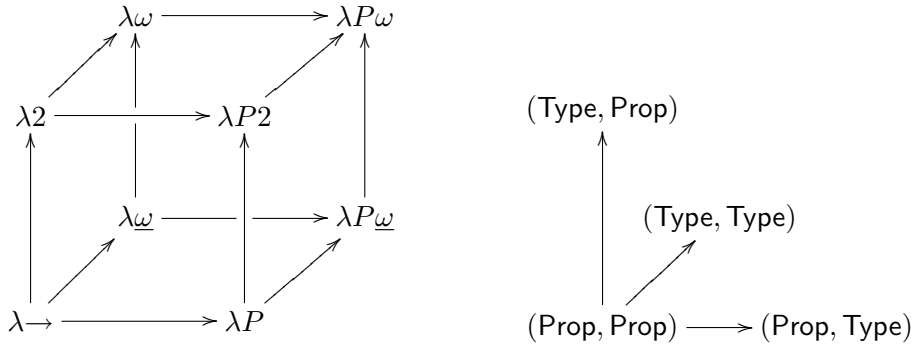
généralement représentées par des couples (s_1, s_2) pour cette raison (*i.e.* en convenant que (s_1, s_2) est une abréviation pour désigner le triplet (s_1, s_2, s_2)). À partir de l'ensemble de sortes $\mathcal{S} = \{\mathsf{Prop}; \mathsf{Type}\}$, il est possible de former les quatre règles suivantes :

$(\mathsf{Prop}, \mathsf{Prop})$	(types simples)
$(\mathsf{Type}, \mathsf{Prop})$	(polymorphisme)
$(\mathsf{Prop}, \mathsf{Type})$	(types dépendants)
$(\mathsf{Type}, \mathsf{Type})$	(constructeurs de types)

(en ne considérant bien entendu que des règles de la forme $(s_1, s_2) \equiv (s_1, s_2, s_2)$).

En pratique, les seuls systèmes intéressants sont ceux qui contiennent la règle $(\mathsf{Prop}, \mathsf{Prop})$ (la règle permettant de construire les types simples du λ-calcul simplement typé). Chacun des systèmes du cube est construit à partir de cette règle en effectuant l'une des huit combinaisons possibles des trois autres règles que sont la règle $(\mathsf{Type}, \mathsf{Prop})$ (polymorphisme), la règle $(\mathsf{Prop}, \mathsf{Type})$ (types dépendants) et la règle $(\mathsf{Type}, \mathsf{Type})$ (constructeurs de types).

La géométrie des systèmes du cube Graphiquement, il est d'usage de représenter ces huit systèmes sur les sommets d'un cube, en associant une direction spatiale à chacune des trois règles $(\mathsf{Type}, \mathsf{Prop})$, $(\mathsf{Prop}, \mathsf{Type})$ et $(\mathsf{Type}, \mathsf{Type})$:



Dans le dessin ci-dessus à gauche, chacune des flèches symbolise l'ajout d'une règle, qui permet de passer d'un système à un système adjacent, la nature de la règle étant donnée par la direction de la flèche par la correspondance indiquée dans le dessin de droite. (Les flèches vont ainsi des systèmes les moins expressifs vers les systèmes les plus expressifs.) La face haute du cube regroupe les systèmes *polymorphes* $(\mathsf{Type}, \mathsf{Prop})$, la face droite les systèmes avec *types dépendants* $(\mathsf{Prop}, \mathsf{Type})$ et la face arrière les systèmes avec *constructeurs de types* $(\mathsf{Type}, \mathsf{Type})$.

Le plus simple de ces systèmes, noté $\lambda{\to}$, est celui qui ne contient que la règle $(\mathsf{Prop}, \mathsf{Prop})$. Ce système est le λ-calcul simplement typé, qui est défini ici sous une forme comportant quelques différences avec la présentation du paragraphe 1.1.2, puisque les types de base ne sont pas donnés *a priori* mais sous la forme de variables de type $\alpha : \mathsf{Prop}$ déclarées dans le contexte. Mise à part cette différence de présentation, on vérifie aisément que les types $T : \mathsf{Prop}$ correspondent aux types simples, et que les termes $M : T$ avec $T : \mathsf{Prop}$ correspondent aux termes du λ-calcul simplement typé.

À l'opposé du λ-calcul simplement typé se trouve le Calcul des Constructions, noté $\lambda P\omega$ (ou plus couramment CC), qui est formé à partir des quatre règles $(\mathsf{Prop}, \mathsf{Prop})$, $(\mathsf{Type}, \mathsf{Prop})$,

(Prop, Type) et (Type, Type). Comme ce système de types contient chacun des sept autres, son étude permet en pratique de dériver la plupart des propriétés communes aux systèmes du cube qui s'en déduisent par restriction. Parmi ces propriétés communes, on retiendra plus particulièrement les deux suivantes :

- (*Unicité du type*) Si $\Gamma \vdash M : T$ et $\Gamma \vdash M : T'$, alors $T =_\beta T'$.
- (*Normalisation forte*) Si $\Gamma \vdash M : T$, alors M est un terme fortement normalisable.

Ainsi que nous l'avons déjà souligné, le cadre des systèmes du cube permet de "retrouver" naturellement un certain nombre de systèmes de types standards, tels que le λ-calcul simplement typé ($\lambda\!\!\rightarrow$), le système F ($\lambda 2$), le système $F\omega$ ($\lambda\omega$), la théorie des types de Martin-Löf[22] (λP) et le Calcul des Constructions ($\lambda P\omega$).

Stratification des termes Dans chacun des huit systèmes du cube de Barendregt, la propriété d'unicité du type permet de classer les termes bien typés suivant la sorte de leur type (en distinguant le cas particulier des termes de type Type, puisque la sorte Type n'a pas de type). Formellement, si le jugement $\Gamma \vdash M : T$ est dérivable (dans un système du cube donné), alors une et une seule des trois assertions suivantes est satisfaite :

- $T \equiv \mathsf{Type}$, auquel cas on dit que M est un *genre* dans le contexte Γ ;
- $\Gamma \vdash T : \mathsf{Type}$, auquel cas on dit que M est un *constructeur* dans le contexte Γ ;
- $\Gamma \vdash T : \mathsf{Prop}$, auquel cas on dit que M est un *terme de preuve* dans le contexte Γ.

Cette classification des jugements de typage permet de présenter le formalisme sous forme stratifiée, en introduisant trois catégories syntaxiques distinctes pour définir les genres (notés K, L, etc.), les constructeurs (notés ϕ, ψ, etc.) et les termes de preuves (notés t, u, etc.) La présentation stratifiée de la syntaxe des systèmes du cube est récapitulée par la figure 1.4, dans laquelle nous avons écrit à droite de chaque produit dépendant, de chaque abstraction et de chaque application la règle de formation de produit dépendant qui autorise cette construction.

Genres	K, L	$::=$	Prop		
		$\mid$	$\Pi x : \phi \,.\, L$		(Prop, Type)
		$\mid$	$\Pi \alpha : K \,.\, L$		(Type, Type)
Constructeurs	ϕ, ψ	$::=$	α		
		$\mid$	$\Pi x : \phi \,.\, \psi$		(Prop, Prop)
		$\mid$	$\Pi \alpha : K \,.\, \psi$	$\mid$	(Type, Prop)
		$\mid$	$\lambda x : \phi \,.\, \psi$	$\mid \phi\ t$	(Prop, Type)
		$\mid$	$\lambda \alpha : K \,.\, \psi$	$\mid \phi\ \psi$	(Type, Type)
Termes de preuves	t, u	$::=$	x		
		$\mid$	$\lambda x : \phi \,.\, t$	$\mid t\ u$	(Prop, Prop)
		$\mid$	$\lambda \alpha : K \,.\, t$	$\mid t\ \phi$	(Type, Prop)

FIG. 1.4 – Stratification des termes dans les systèmes du cube

[22] Ou plus exactement la théorie des types de Martin-Löf [46] sans types sommes dépendantes et sans constructions inductives telles que le type de l'égalité ou le type des entiers naturels.

Bien entendu, la syntaxe de chacun des huit systèmes du cube de Barendregt se déduit de la figure 1.4 en ne considèrent que les constructions autorisées par ses règles de formation de produit dépendant.[23] La présentation stratifiée du formalisme nécessite également de distinguer trois jugements de typage, notés $\Gamma \vdash K : \mathsf{Type}$ (typage des genres), $\Gamma \vdash \phi : K$ (typage des constructeurs) et $\Gamma \vdash t : \phi$ (typage des termes de preuves) en plus du jugement de bonne formation de contexte. Les règles de typage sous forme stratifiée se déduisent naturellement des règles de typage des PTS en distinguant quatre règles de typage pour le produit dépendant, quatre règles pour l'abstraction et quatre règles pour l'application, ainsi que deux règles de conversion de type (une pour les constructeurs et une autre pour les termes de preuves). Par ailleurs, les règles de bonne formation de contexte comprennent deux règles d'extension, qui correspondent respectivement à la déclaration d'une variable de constructeur (notée α, β, etc.) et à la déclaration d'une variable de terme de preuve (notée x, y, z, etc.)

Bien que la présentation stratifiée des systèmes du cube soit sensiblement plus lourde à manipuler que la présentation usuelle dans le style des PTS, son utilisation permet en pratique de simplifier l'étude des propriétés fines du formalisme (telles que la propriété de normalisation forte) ainsi que la définition de nombreuses fonctions de traduction (telles que la fonction d'effacement des dépendances [26], qui envoie les systèmes de la face droite du cube sur les systèmes de la face gauche). Nous aurons l'occasion de revenir sur cette présentation stratifiée à la section 1.4 lorsque nous présenterons le cube des *Type Assignement Systems* ainsi que la fonction d'effacement qui lui est associée.

1.3 Arguments implicites

1.3.1 Les limites des PTS

En pratique, l'expressivité des PTS se paye par une certaine lourdeur de la syntaxe, qui nécessite d'introduire dans les termes de nombreuses abstractions et applications purement administratives. Contrairement aux langages de la famille ML, le polymorphisme s'introduit et s'élimine explicitement, et l'identité polymorphe par exemple

$$\mathsf{id} \quad : \quad \Pi A : \mathsf{Prop} \,.\, A \to A \quad := \quad \lambda A : \mathsf{Prop} \,.\, \lambda x : A \,.\, x$$

s'écrit dans les systèmes du cube disposant de la règle $(\mathsf{Type}, \mathsf{Prop}, \mathsf{Prop})$ à l'aide de deux abstractions à la Church, là où une seule abstraction (dépourvue d'annotation de type) suffirait en ML. De même, chaque utilisation de la constante id ainsi définie requiert deux applications : une pour le type A, et l'autre pour l'argument réel x (de type A).[24] Notons que ce problème n'est pas spécifiquement lié à l'imprédicativité, puisqu'il se pose de la même façon dans le

[23] Les constructions qui ne sont marquées par aucune règle sont donc communes aux huit systèmes du cube de Barendregt, de même que les constructions marquées par la règle $(\mathsf{Prop}, \mathsf{Prop})$ (*cf* Fig. 1.4). On remarquera que dans le cas des systèmes dépourvus de types dépendants (*i.e.* dans lesquels la règle $(\mathsf{Prop}, \mathsf{Type})$ est exclue), les genres forment une catégorie syntaxique autonome (les genres ne dépendent que d'eux-mêmes) et les constructeurs ne dépendent que des constructeurs et des genres. Dans ce cadre particulier, il est facile de vérifier que les produits $\Pi\alpha : K \,.\, L$ (genre) et $\Pi x : \phi \,.\, \psi$ (constructeur) sont des produits non-dépendants, que l'on note généralement $K \to L$ et $\phi \to \psi$ pour cette raison.

[24] On remarquera également que la réduction d'un terme de la forme $\mathsf{id}\ A\ x$ (où A et x sont des variables de type Prop et A respectivement) nécessite deux étapes de β-réduction, alors qu'en ML, le calcul équivalent s'effectue en une seule étape. En pratique, la présence des abstractions et applications de type entraîne l'apparition de β-radicaux administratifs qui ont naturellement tendance à dégrader l'efficacité de la β-réduction.

cadre du polymorphisme prédicatif du *Logical Framework* de Martin-Löf [47] ou du Calcul des Constructions avec univers (en considérant l'identité polymorphe $\lambda A : \mathsf{Type}_i \,.\, \lambda x : A \,.\, x$ par exemple).

Bien entendu, ces abstractions et applications supplémentaires ont de bonnes raison d'exister. Dans un formalisme où les types peuvent être manipulés de la même façon que les autres termes, il n'y a pas de raison fondamentale de différencier l'abstraction et l'application de type de toute autre abstraction ou application. De plus, dans les PTS comportant le polymorphisme du système F et/ou les types dépendants de la théorie de Martin-Löf, la relation de typage devient indécidable en l'absence de telles annotations [62, 21].

La verbosité de la syntaxe des PTS n'est pas seulement un problème pour l'utilisateur intéressé par la programmation ou plus généralement par la formalisation effective des mathématiques sur machine. D'un point de vue théorique, les abstractions et applications administratives (sans parler des annotations de type sous les abstractions) constituent dans les termes de preuves un bruit de fond qui a tendance a en obérer le contenu calculatoire. Pour illustrer ce point, considérons par exemple la relation d'égalité de Leibniz dans le Calcul des Constructions, qui est définie de manière imprédicative par :

$$\begin{array}{rcl} \mathsf{eq} & : & \Pi A : \mathsf{Prop} \,.\, A \to A \to \mathsf{Prop} \\ & := & \lambda A : \mathsf{Prop} \,.\, \lambda x, y : A \,.\, \Pi P : (A \to \mathsf{Prop}) \,.\, Px \to Py \end{array}$$

La réflexivité de cette relation est établie par le terme de preuve suivant

$$\begin{array}{rcl} \mathsf{refl} & : & \Pi A : \mathsf{Prop} \,.\, \Pi x : A \,.\, \mathsf{eq}\, A x x \\ & := & \lambda A : \mathsf{Prop} \,.\, \lambda x : A \,.\, \lambda P : (A \to \mathsf{Prop}) \,.\, \lambda p : Px \,.\, p \end{array}$$

dont un examen attentif montre que seule la quatrième abstraction introduit une variable qui a un véritable contenu calculatoire dans la preuve. Cet exemple montre clairement que le contenu calculatoire de la constante refl — qui n'est rien d'autre que la fonction identité $\lambda p \,.\, p$ — est complètement masqué par les trois abstractions précédentes dont la seule fonction est de propager les contraintes de typage à travers le terme de preuve lui-même.[25]

1.3.2 Les arguments implicites de LEGO et de Coq

De nombreuses approches ont été proposées pour apporter une réponse à ce problème, que ce soit de manière pratique ou sur un plan plus théorique. D'un point de vue pratique, la plupart des assistants à la démonstration dont le formalisme est basé sur une extension d'un PTS (tels que Coq [11] ou LEGO [43]) offrent un mécanisme d'*arguments implicites* qui évite à l'utilisateur d'écrire explicitement les applications de type dans la mesure où celles-ci peuvent être automatiquement synthétisées par la machine.

Le système d'arguments implicites tel qu'il est implémenté dans LEGO [43] (et, à quelques différences près, dans Coq également [56]) repose sur une distinction syntaxique entre deux formes de produits dépendants, d'abstractions et d'applications, les uns étant appelés *explicites* et les autres *implicites*. Formellement, ces systèmes sont basés sur une variante des PTS — que nous appellerons ici les *PTS bicolores* — dans laquelle chaque produit dépendant, chaque

[25] On notera également que la seconde abstraction $\lambda x : A \ldots$ n'est pas introduite par la règle de polymorphisme, mais par la règle $(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop})$, ce qui montre la relative indépendance de ce problème vis-à-vis de la stratification.

abstraction et chaque application est marqué avec la couleur *explicite* ou la couleur *implicite*. On trouvera dans la figure 1.5 un récapitulatif de la syntaxe des PTS bicolores, qui correspond à peu de choses près à celle qui est utilisée dans le système LEGO.

M, N, T, U	$::=$	x			(variable)
	$\mid$	s			(sorte)
	$\mid$	$\Pi x : T \,.\, U$	$\mid$	$\Pi\, x \mid T \,.\, U$	(produit explicite/implicite)
	$\mid$	$\lambda x : T \,.\, M$	$\mid$	$\lambda x \mid T \,.\, M$	(abstraction explicite/implicite)
	$\mid$	$M\ N$	$\mid$	$M \mid N$	(aplication explicite/implicite)

FIG. 1.5 – Syntaxe des PTS bicolores

Les règles de typage des PTS bicolores sont les mêmes que celles des PTS standards, à cette différence près que les règles de formation du produit dépendant, de l'abstraction et de l'application sont dupliquées de manière à respecter la coloration. Grosso modo, ces règles de typage font en sorte que le type d'une abstraction d'une certaine couleur soit un produit dépendant de la même couleur, et qu'une application colorée ne puisse être formée que dans la mesure ou son membre gauche est une fonction de la même couleur que l'application.[26]

Bien que d'un point de vue théorique il n'y ait aucune différence fondamentale entre les deux couleurs, en pratique, le système de vérification de type (que ce soit dans LEGO ou dans Coq) les distingue en permettant à l'utilisateur d'omettre systématiquement les arguments des applications implicites — appelés *arguments implicites* pour cette raison — qui sont ensuite reconstruits automatiquement par le système dans la mesure du possible. Autrement dit, la couleur implicite n'est rien d'autre qu'une marque indiquant au système que l'argument de la fonction implicite doit être deviné en fonction du contexte (typiquement à partir des types des arguments explicites suivants).

Dans LEGO, l'identité polymorphe (dans la sorte Type) est définie par une abstraction implicite suivie d'une abstraction explicite

$$\mathsf{id} \quad : \quad \Pi\, A \mid \mathsf{Type} \,.\, A \to A \quad := \quad \lambda A \mid \mathsf{Type} \,.\, \lambda x : A \,.\, x,$$

ce qui indique au système que le premier argument de cette fonction doit être systématiquement reconstruit à partir de l'information donnée par le second argument (qui est ici le seul argument fourni par l'utilisateur). Lorsque l'utilisateur écrit le terme $\mathsf{id}\ 0$ par exemple, le système reconstruit l'argument implicite manquant (ici le type nat) qui est ensuite conservé dans la représentation interne du terme.

Insistons sur le fait que ce mécanisme n'est juste qu'une facilité syntaxique destinée à alléger la saisie des termes ainsi que leur affichage. Dans la représentation interne cependant, les termes sont conservés sous leur forme complète (c'est-à-dire sous la forme $\mathsf{Id} \mid \mathsf{nat}\ 0$ dans l'exemple précédent), puisque les arguments implicites peuvent être utilisés explicitement par un calcul au cours d'un test de conversion effectué ultérieurement.

[26] En conséquence de quoi tous les β-radicaux bien typés sont formés d'une abstraction et d'une application de la même couleur, cet invariant étant bien sûr préservé au cours de la β-réduction.

Le principal avantage de cette méthode est de préserver toutes les propriétés syntaxiques et sémantiques du PTS initial (modulo la "colorisation" de la syntaxe)[27] car les arguments implicites ne sont implicites que pour l'utilisateur, mais pas pour le système. Néanmoins, ce mécanisme peut parfois porter à confusion pour la raison que le système conserve en mémoire une information implicite invisible pour l'utilisateur. En particulier, deux termes peuvent paraître égaux à l'affichage, alors qu'ils ne le sont pas de manière interne en raison de la présence d'informations cachées qui diffèrent. Un tel exemple est donné par les termes

$$P\,\mathsf{nat}\;(\mathsf{id}\,\mathsf{nat}) \qquad \text{et} \qquad P\,\mathsf{bool}\;(\mathsf{id}\,\mathsf{bool})$$

(où P est une variable de type $\Pi\, A \mid \mathsf{Type}\,.\,(A{\to}A) \to \mathsf{Prop}$) qui sont affichés tous les deux de la même manière ("P id").

1.3.3 Les arguments implicites de M. Hagiya et Y. Toda

Dans [32], M. Hagiya et Y. Toda reprennent la syntaxe des PTS bicolores en étudiant la possibilité de faire disparaître définitivement les arguments implicites, y compris dans la représentation interne des termes.

L'idée sur laquelle est basé leur travail est la suivante : si par un critère bien choisi on s'assure que les arguments implicites sont reconstruits de manière unique (à β-conversion près), il n'est plus nécessaire de les conserver de manière interne puisque le test de conversion effectué sur les termes du calcul implicite (*i.e.* dans lesquels les arguments implicites ont été effacés) donnera le même résultat que s'il avait été effectué sur les termes correspondants dans le calcul bicolore (obtenus en reconstruisant les arguments implicites absents).

En suivant cette idée, les auteurs définissent un calcul implicite — qui est essentiellement le calcul bicolore privé de l'application implicite — et introduisent un certain nombre de restrictions assez fortes dans les règles de typage du calcul (issues des règles correspondantes dans le calcul bicolore) de manière à s'assurer de l'existence et de l'unicité (à β-conversion près) de la reconstruction des arguments implicites dans le calcul bicolore. De cette manière, ils montrent que deux termes bien typés dans le calcul implicite sont convertibles si et seulement si les termes obtenus par reconstruction des arguments implicites dans le calcul bicolore le sont également, ce qui leur permet ainsi de s'assurer de la préservation de toutes les bonnes propriétés du calcul (préservation du typage par la β-réduction, cohérence et normalisation forte) dans un cadre où l'application implicite a complètement disparu.

Malheureusement, l'algorithme de typage qu'ils proposent est très complexe, et la propriété d'unicité de la reconstruction des arguments implicites n'est obtenue qu'au prix d'un ensemble de restrictions drastiques. En particulier, leur approche ne permet pas de traiter le cas de l'abstraction implicite, qui disparaît donc complètement du formalisme.[28]

[27] Les termes d'un PTS bicolore se plongent dans le PTS standard correspondant par un processus de décoloration évident. En particulier, le passage d'un PTS (standard) au PTS bicolore correspondant préserve les propriétés de normalisation forte et de cohérence.

[28] Dans l'approche proposée dans [32], on ne peut pas définir une identité polymorphe implicite (comme celle qui est définie dans LEGO), mais il est possible de l'*axiomatiser* en introduisant dans le contexte une variable id de type $\Pi\, A \mid \mathsf{Type}\,.\,A \to A$ munie de l'axiome correspondant. Bien entendu, cette façon de procéder interdit définitivement toute possibilité de calculer avec la constante id, ce qui limite sérieusement ses possibilités d'utilisation.

Le problème de l'abstraction implicite Il est intéressant de s'attarder quelques instants sur le problème posé par l'abstraction implicite dans le calcul implicite proposé dans [32]. Dans ce cadre, la fonction d'effacement des arguments implicites $M \mapsto |M|$ ne fait disparaître que les applications implicites, mais conserve les abstractions implicites :

$$\begin{array}{lcllcl} |s| & = & s & |x| & = & x \\ |\Pi x : T \,.\, U| & = & \Pi x : |T| \,.\, |U| & |\Pi\, x \mid T \,.\, U| & = & \Pi\, x \mid |T| \,.\, |U| \\ |\lambda x : T \,.\, M| & = & \lambda x : |T| \,.\, |M| & |\lambda x \mid T \,.\, M| & = & \lambda x \mid |T| \,.\, |M| \\ |M\ N| & = & |M|\ |N| & |M \mid N| & = & |M| \end{array}$$

Ce choix — dicté sans doute par une volonté de conserver la décidabilité de l'inférence des arguments implicites — entraîne une dissymétrie calculatoire dans le formalisme implicite (*i.e.* obtenu après effacement). En effet, si dans le calcul bicolore, la réduction du terme $\mathsf{id} \mid \mathsf{nat}\ 0$ ne pose aucun problème

$$(\lambda A \mid \mathsf{Prop} \,.\, \lambda x : A \,.\, x) \mid \mathsf{nat}\ 0 \quad \to_\beta \quad (\lambda x : \mathsf{nat} \,.\, x)\ 0 \quad \to_\beta \quad 0,$$

la disparition de l'application implicite rompt la symétrie entre l'abstraction et l'application, ce qui ne permet plus de donner un sens calculatoire à l'expression

$$(\lambda A \mid \mathsf{Prop} \,.\, \lambda x : A \,.\, x)\ 0 \quad \to_\beta \quad ?$$

dont ont voit mal comment on pourrait la β-réduire sans faire intervenir un processus complexe de reconstruction des arguments implicites à la volée au cours de la β-réduction.[29]

Dans [32], le problème est supprimé en faisant disparaître l'abstraction implicite du formalisme, mais il semble clair que si on doit la réintroduire, il est préférable de l'effacer en même temps que l'application implicite pour des raisons de symétrie évidentes :

$$|\lambda x : T \,.\, M| \quad \triangleq \quad |M| \qquad\qquad |M \mid N| \quad \triangleq \quad |M|.$$

Cependant, cette nouvelle définition de la fonction d'effacement fait rapidement surgir de nouveaux problèmes. Si l'on s'en tient à la définition ci-dessus, l'effacement du terme représentant l'identité polymorphe implicite

$$|\lambda A \mid \mathsf{Prop} \,.\, \lambda x : A \,.\, x| \quad = \quad \lambda x : A \,.\, x$$

conduit à un terme dans lequel la variable A n'est plus liée par l'abstraction implicite qui a disparu, ce qui entraîne une incohérence manifeste vis-à-vis du scope des variables. Pour résoudre ce problème, il faut donc également effacer systématiquement les annotations de type sous les abstractions explicites

$$|\lambda x \mid T \,.\, M| \quad \triangleq \quad \lambda x \,.\, |M|$$

de manière à obtenir le résultat correct, soit : $|\lambda A \mid \mathsf{Prop} \,.\, \lambda x : A \,.\, x| \quad = \quad \lambda x \,.\, x.$

Cette modification (mineure) n'est toutefois pas suffisante. Comme dans le calcul bicolore, les constructions explicites et implicites sont symétriques, il est toujours possible de les échanger et de considérer par exemple une fonction identité monomorphe "inversée" représentée par

[29] Un tel mécanisme, un moment envisagé dans [32], semble difficile à définir de manière satisfaisante.

le terme $\lambda x \mid \mathsf{nat} \,.\, x$, c'est-à-dire construite avec une abstraction implicite plutôt qu'avec une abstraction explicite. Sur ce terme, la fonction d'effacement

$$|\lambda x \mid \mathsf{nat} \,.\, x| \quad = \quad x$$

donne une fois de plus un résultat incorrect, puisque le lieur introduisant la variable x est effacé, mais pas la variable elle-même. L'échec était prévisible, car on voit mal quelle signification accorder au terme $\lambda x \mid \mathsf{nat} \,.\, x$ qui utilise de manière explicite (*i.e.* en position de calcul) un de ses arguments implicites.

L'exemple ci-dessus fait apparaître le principal défaut du calcul bicolore, qui est de traiter de manière symétrique des constructions dont la signification ne l'est pas, et qui de toutes façons ne sont pas traitées de la même manière par la fonction d'effacement. Intuitivement, une fonction implicite (définie par abstraction implicite) ne peut faire du sens que dans la mesure où son argument est toujours utilisé en position implicite (*i.e.* dans un sous-terme destiné à être effacé) sans jamais intervenir dans le calcul réel.

Pour résoudre ce problème, il suffit de rompre la symétrie du calcul bicolore en introduisant une fonction d'effacement partielle qui rejettera systématiquement les termes qui ne respectent pas le critère édicté ci-dessus. La définition correcte de la fonction d'effacement devient alors :

$$|\lambda x : T \,.\, M| \triangleq \lambda x \,.\, |M| \qquad |\lambda x \mid T \,.\, M| \triangleq \begin{cases} |M| & \text{si } x \notin FV(|M|) \\ \text{échec} & \text{sinon} \end{cases}$$

$$|M\ N| \triangleq |M|\ |N| \qquad |M \mid N| \triangleq |M|$$

(les autres cas demeurant inchangés). Nous aurons l'occasion de revenir plus en détails sur la dissymétrie profonde entre les constructions explicites et les constructions implicites à la fin de ce chapitre.

1.4 Systèmes de types à la Curry

Ainsi que nous l'avons souligné au début de la section précédente, les redondances syntaxiques des PTS ne posent pas seulement un problème pratique dans le cadre de la formalisation des mathématiques sur machine, mais constituent également un problème théorique dès qu'il s'agit d'analyser le contenu calculatoire des termes de preuves.

Ces redondances s'expliquent en grande partie par la présentation du formalisme : depuis l'introduction de ce chapitre en effet, nous avons systématiquement adopté une présentation à la Church, dans laquelle les termes sont quasiment donnés avec leur type (qui se déduit aisément des annotations de type figurant dans les termes). Cette façon de présenter les systèmes de types reflète le point de vue selon lequel un terme ne fait du sens qu'au sein d'un type donné. Dans les systèmes de types à la Church, le type d'un terme est nécessairement unique[30], et les termes bruts[31] tels qu'ils sont donnés par la syntaxe ne constituent qu'une

[30] Cette affirmation doit être nuancée par le fait que la propriété d'unicité du type — qui est généralement caractéristique des systèmes de types à la Church — n'est pas vraie dans tous les PTS, mais uniquement dans les PTS fonctionnels [26].

[31] *Raw terms* en anglais. Afin d'insister sur le fait que les termes bruts ne sont pas des "vrais termes", certains auteurs parlent de *pré-termes* pour désigner ces termes qui n'ont pas encore passé l'épreuve du jugement de typage. Dans la suite de cette thèse, on suivra davantage l'esprit des systèmes de types à la Curry en parlant de *termes* pour désigner les termes bruts, et en appelant *termes bien typés* les termes qui admettent au moins un type (dans un contexte donné).

première épure des termes susceptible d'être rejetée par le jugement de typage.

Afin d'éviter les redondances syntaxiques inhérentes aux systèmes de types à la Church, la programmation fonctionnelle suit généralement une toute autre approche, qui est celle des systèmes de types à la Curry. Dans les systèmes de types à la Curry, les termes sont davantage perçus comme des programmes ayant une existence autonome en dehors des types, qui ne sont plus que des spécifications exprimant certaines propriétés calculatoires des termes. De même qu'un programme peut satisfaire plusieurs spécifications, un même terme admet souvent une infinité de types différents dans un système de types à la Curry. Dans le λ-calcul simplement typé à la Curry par exemple, la fonction identité est représentée par un terme unique noté $\lambda x\,.\,x$ (*i.e.* la fonction identité du λ-calcul pur) à qui il est possible d'attribuer une infinité de types différents, soit ici tous les types de la forme $A \to A$ (où A est un type simple arbitraire).

En pratique, la syntaxe des systèmes de types à la Curry est beaucoup plus concise que celle des systèmes de types à la Church, car les termes de preuves sont donnés par les termes du λ-calcul pur, qui ne comportent ni abstraction ni application de type, ni même l'annotation de type sous le symbole λ. Bien entendu, cette concision a un prix, puisque les règles de typage ne sont plus dirigées par la syntaxe : dans les systèmes de types à la Curry suffisamment expressifs — typiquement en présence du polymorphisme du système F ou des types dépendants — la vérification et l'inférence de type sont tous les deux indécidables [62, 21].

Dans cette section, nous nous proposons de passer brièvement en revue quelques systèmes de types à la Curry. Nous commencerons par présenter le système F à la Curry, puis nous étudierons une généralisation de cette approche à tous les systèmes du cube de Barendregt en introduisant le cube des *Type Assignment Systems*.

1.4.1 Les systèmes F à la Church et à la Curry

Dans la présentation usuelle du système F (*i.e.* la présentation à la Church), la quantification de type $\Pi\alpha\,.\,A$ (où A est une expression de type dépendant de la variable de type α) désigne le type des familles de termes de type A indicées par la variable de type α. Dans ce cadre, les familles de termes sont introduites au moyen d'une abstraction de type notée $\Lambda\alpha\,.\,M$, et on en extrait un élément particulier avec une application de type notée $M\ A$.[32] Formellement, la syntaxe du système F à la Church est donnée par :

Types	A, B	$::=$	$\alpha \mid A \to B$	(variable de type, type flèche)
		$\mid$	$\Pi\alpha\,.\,A$	(quantification de type)
Termes	M, N	$::=$	x	(variable)
		$\mid$	$\lambda x : A\,.\,M \mid M\ N$	(abstraction/application de terme)
		$\mid$	$\Lambda\alpha\,.\,M \mid M\ A$	(abstraction/application de type)

Dans le système F à la Curry, la quantification de type $\Pi\alpha\,.\,A$ a une signification très différente puisqu'elle ne désigne pas un type de *familles de termes* (c'est-à-dire un espace de fonctions) mais plutôt une *intersection de types*, et plus précisément l'intersection de tous les

[32] Rappelons que dans le cadre des PTS (dans lequel le système F correspond au système du cube de Barendregt noté $\lambda 2$) les notations $\Pi\alpha\,.\,A$ (quantification de type) et $\Lambda\alpha\,.\,M$ (abstraction de type) ne sont que des abréviations pour désigner les termes $\Pi\alpha : \mathsf{Prop}\,.\,A$ et $\lambda\alpha : \mathsf{Prop}\,.\,M$ respectivement. Bien que les PTS ne fassent pas de distinction entre les types et les termes, on adoptera ici l'approche plus conventionnelle qui consiste à les présenter de manière séparée.

types A (dépendant de α) lorsque la variable de type α parcourt l'ensemble de tous les types. Afin d'éviter toute confusion, nous utiliserons dorénavant la notation $\forall\alpha\,.\,A$ pour désigner la quantification de type du système F à la Curry.

Parce que dans le système F à la Curry, la quantification de type a la signification d'une intersection, il n'est plus nécessaire pour introduire et éliminer le polymorphisme de recourir ni à l'abstraction de type ni à l'application de type. Par conséquent, les termes du système F à la Curry ne sont rien d'autre que les termes du λ-calcul pur, et la syntaxe du formalisme est donnée par :

$$\begin{array}{lllllll} \textbf{Types} & A, B & ::= & \alpha & \mid & A \to B & \mid \quad \forall\alpha\,.\,A \\ \textbf{Termes} & M, N & ::= & x & \mid & \lambda x\,.\,M & \mid \quad M\ N \end{array}$$

Malgré ces différences, ces deux systèmes de types ont une structure très similaire, ce qui apparaît très nettement quand on observe leurs règles de typage, que nous avons rappelées dans la figure 1.6.[33] La seule différence notable apparaît dans les règles d'introduction et d'élimination de la quantification de type : dans le cas du système F à la Church, ces deux règles s'accompagnent d'une transformation du terme (par l'apparition d'une abstraction ou d'une application de type) alors que dans le système F à la Curry, les termes restent inchangés lorsqu'on passe de la prémisse principale à la conclusion.

La fonction d'effacement La correspondance entre ces deux systèmes peut être formalisée au moyen d'une *fonction d'effacement* qui envoie chaque terme du système F à la Church sur un terme du système F à la Curry, simplement en faisant disparaître toutes les abstractions et applications de type, ainsi que les annotations de type dans les abstractions. Formellement, cette fonction d'effacement notée $M \mapsto |M|$ est définie par :

$$\begin{array}{lcl} |x| & \equiv & x \\ |\lambda x : A\,.\,M| & \equiv & \lambda x\,.\,|M| \\ |M\ N| & \equiv & |M|\ |N| \\ |\Lambda\alpha\,.\,M| & \equiv & |M| \\ |M\ A| & \equiv & |M| \end{array}$$

En toute rigueur, il faudrait également définir une fonction de traduction sur les types qui transformerait $\Pi\alpha\,.\,A$ en $\forall\alpha\,.\,A$, mais le cadre du système F reste suffisamment simple pour que nous puissions nous permettre de ne pas le faire en considérant simplement que le passage d'un système à l'autre s'accompagne tacitement d'un tel changement de notation.

La fonction d'effacement s'étend naturellement aux dérivations de typage : si π désigne une dérivation dans le système F à la Church, on note $|\pi|$ l'arbre obtenu en effaçant toutes les

[33] Les contextes de typage de ces deux systèmes sont des listes finies de la forme $[x_1 : A_1; \ldots; x_n : A_n]$ dans lesquelles les types A_i sont susceptibles de contenir des variables de types libres, lesquelles ne sont jamais déclarées dans les contextes (contrairement à la présentation sous forme de PTS). Ceci explique l'utilité de la condition de bord $\alpha \notin \Gamma$ dans la règle d'introduction de la quantification de type, qui ne fait pas de sens lorsque cette condition n'est pas satisfaite. Dans la présentation sous forme de PTS le problème ne se pose pas (pour le système F à la Church), car les variables de types sont explicitement déclarées dans les contextes, et la prémisse de la règle d'introduction de la quantification de type suppose que la variable α est la dernière variable déclarée dans le contexte, ce qui entraîne qu'elle ne peut pas apparaître dans un type déclaré ailleurs. Bien entendu, ces deux points de vue sont complètement équivalents.

Système F à la Church

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

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \Lambda\alpha \,.\, M : \Pi\alpha \,.\, A}\ {\scriptstyle \alpha\notin\Gamma} \qquad \frac{\Gamma \vdash M : \Pi\alpha \,.\, A}{\Gamma \vdash M\ B : A\{\alpha := B\}}$$

Système F à la Curry

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

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall\alpha \,.\, A}\ {\scriptstyle \alpha\notin\Gamma} \qquad \frac{\Gamma \vdash M : \forall\alpha \,.\, A}{\Gamma \vdash M : A\{\alpha := B\}}$$

FIG. 1.6 – Règles de typage des systèmes F à la Church et à la Curry

abstractions, applications et annotations de type dans les termes qui figurent dans π (et en remplaçant dans les types chaque occurrence du symbole Π par le symbole $\forall$). Cette fonction d'effacement sur les dérivations étant définie, il est facile de vérifier qu'elle satisfait les deux propriétés suivantes :

1. Si π est une dérivation du jugement $\Gamma \vdash M : A$ dans le système F à la Church, alors $|\pi|$ est une dérivation du jugement $\Gamma \vdash |M| : A$ dans le système F à la Curry.
2. Si π est une dérivation du système F à la Curry, il existe une unique dérivation π_0 dans le système F à la Church telle que $|\pi_0| = \pi$.

Les deux propriétés ci-dessus expriment que les systèmes F à la Church et à la Curry sont isomorphes, en ce sens que la fonction d'effacement établit une correspondance bijective entre les dérivations de typage de ces deux systèmes. On remarquera en particulier que le second point évoqué ci-dessus (la partie réciproque) entraîne que tout jugement dérivable dans le système F à la Curry provient d'un jugement dérivable dans le système F à la Church, en ce sens que pour tout jugement dérivable $\Gamma \vdash M : A$ dans le système F à la Curry, il existe un unique terme M_0 du système F à la Church tel que $|M_0| = M$, et tel que le jugement $\Gamma \vdash M_0 : A$ est dérivable dans le système F à la Church. Une conséquence immédiate de cette remarque est que dans le système F à la Curry, tous les termes bien typés sont fortement normalisables.[34]

[34]Pour établir ce résultat, il faut bien entendu s'assurer que 1. la fonction d'effacement est correcte vis-à-vis de la β-réduction, mais surtout que 2. toute étape de β-réduction effectuée dans le système F à la Curry (à partir d'un terme bien typé) peut être relevée en une ou plusieurs étapes de β-réductions dans le système F à

Il est par ailleurs important de remarquer que l'isomorphisme entre le système F à la Church et le système F à la Curry est un isomorphisme au niveau des *dérivations*, mais pas au niveau des *jugements dérivables*. Sur l'ensemble des jugements dérivables du système F à la Church, la fonction d'effacement n'est pas injective ainsi que le montrent les deux jugements suivants

$$[f : \Pi\alpha\,.\,\alpha \to \alpha] \;\vdash\; f\ (\Pi\alpha\,.\,\alpha \to \alpha)\ f \quad : \quad \Pi\alpha\,.\,\alpha \to \alpha$$

$$[f : \Pi\alpha\,.\,\alpha \to \alpha] \;\vdash\; \Lambda\alpha\,.\,f\ (\alpha \to \alpha)\ (f\ \alpha) \quad : \quad \Pi\alpha\,.\,\alpha \to \alpha$$

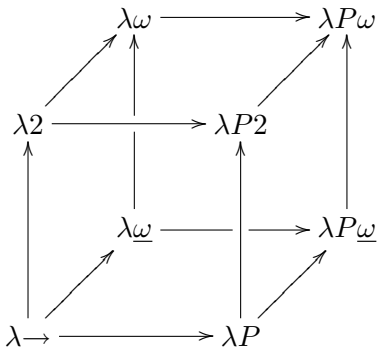
qui se traduisent en un seul et même jugement du système F à la Curry

$$[f : \forall\alpha\,.\,\alpha \to \alpha] \;\vdash\; f\ f \quad : \quad \forall\alpha\,.\,\alpha \to \alpha.$$

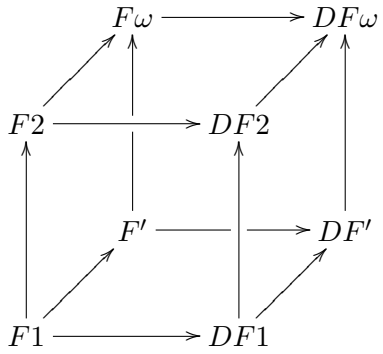
Nous aurons l'occasion de voir à la fin du paragraphe suivant que la non-injectivité de la fonction d'effacement (sur les jugements dérivables) a des répercussions importantes sur la structure des *Type Assignment Systems* en présence de types dépendants et de polymorphisme.

1.4.2 Le cube des *Type Assignment Systems*

Dans [60], S. van Bakel *et al.* proposent d'étendre cette correspondance à tous les systèmes du cube de Barendregt. Pour cela, ils introduisent un cube formé par huit systèmes de types à la Curry — les *Type Assignment Systems* (TAS) — qui sont reliés aux huit systèmes du cube de Barendregt par une fonction d'effacement qui généralise la fonction d'effacement du système F :



Le cube de Barendregt



Le cube des TAS

De même que le système F à la Church correspond au PTS noté $\lambda 2$ dans le cube de Barendregt (ci-dessus à gauche), le système F à la Curry correspond au TAS noté $F2$ dans le cube des *Type Assignment Systems* (ci-dessus à droite), et la fonction d'effacement généralisée restreinte à ces deux systèmes est identique — au changement de présentation près — à la fonction d'effacement que nous avons introduite dans le paragraphe précédent.

Les *Type Assignement Systems* La syntaxe et les règles de typage des TAS sont basées sur la présentation stratifiée des systèmes du cube de Barendregt que nous avons introduite

la Church. Le fait qu'une étape de β-réduction dans le système F à la Curry se relève en *au moins une* étape de β-réduction dans le système F à la Church est crucial pour établir le résultat de normalisation.

au paragraphe 1.2.2.[35] La syntaxe des genres, des constructeurs et des termes de preuves dans le cube des TAS est essentiellement la même que dans le cube de Barendregt (Fig. 1.4 p. 35), à ces deux différences près que :

1. le produit dépendant imprédicatif $\Pi\alpha : K \,.\, \psi$ (formé par la règle $(\mathsf{Type}, \mathsf{Prop})$ dans les constructeurs) est remplacé par un constructeur d'intersection noté $\forall\alpha : K \,.\, \psi$;
2. la syntaxe des termes de preuves telle qu'elle est donnée par la figure 1.4 est remplacée par la syntaxe du λ-calcul pur (*i.e.* variable, abstraction non-typée et application).

On trouvera dans la figure 1.7 un récapitulatif de la syntaxe des TAS, sachant que chacun des huit TAS n'utilise parmi les constructions syntaxiques proposées que celles qui sont autorisées par ses règles de formation de produits dépendants.

Genres	K, L	$::=$	Prop		
		$\mid$	$\Pi x : \phi \,.\, L$		$(\mathsf{Prop}, \mathsf{Type})$
		$\mid$	$\Pi\alpha : K \,.\, L$		$(\mathsf{Type}, \mathsf{Type})$
Constructeurs	ϕ, ψ	$::=$	α		
		$\mid$	$\Pi x : \phi \,.\, \psi$		$(\mathsf{Prop}, \mathsf{Prop})$
		$\mid$	$\forall\alpha : K \,.\, \psi$	$\mid$	$(\mathsf{Type}, \mathsf{Prop})$
		$\mid$	$\lambda x : \phi \,.\, \psi$	$\mid \phi\ t$	$(\mathsf{Prop}, \mathsf{Type})$
		$\mid$	$\lambda\alpha : K \,.\, \psi$	$\mid \phi\ \psi$	$(\mathsf{Type}, \mathsf{Type})$
Termes de preuves	t, u	$::=$	$x \mid \lambda x \,.\, t \mid t\ u$		

FIG. 1.7 – Syntaxe des *Type Assignment Systems*

Les règles de typage des TAS que nous avons rappelées dans la figure 1.8 se déduisent aisément des règles de typage des systèmes du cube de Barendregt (dans leur présentation stratifiée) en appliquant à tous les termes qui y figurent la fonction d'effacement que nous allons introduire dans les lignes qui suivent. On remarquera qu'en raison de la disparition de l'abstraction et de l'application de constructeur (Fig 1.4) ainsi que de l'annotation de type sous le symbole λ dans les termes de preuves, les règles de typage ne sont plus dirigées par la syntaxe. Néanmoins, ces règles de typage conservent la plupart des bonnes propriétés des règles de typage des systèmes du cube (substitutivité, affaiblissement, β-autoréduction) sauf bien entendu la propriété d'unicité du type, qui n'est plus satisfaite ici.

La fonction d'effacement La fonction d'effacement des TAS — qui envoie respectivement les genres, constructeurs et termes de preuves donnés par la figure 1.4 sur les genres, constructeurs et termes de preuves donnés par la figure 1.7 — est définie par analogie avec la fonction d'effacement du système F, de manière à remplacer le produit dépendant imprédicatif

[35] Contrairement aux systèmes du cube de Barendregt, les TAS n'admettent pas de présentation non stratifiée dans le style des PTS. En particulier, les abstractions $\lambda x : \phi \,.\, \psi$ et $\lambda\alpha : K \,.\, \psi$ qui sont définies au niveau des constructeurs comportent toutes les deux une annotation de type, contrairement à l'abstraction $\lambda x \,.\, t$ qui est définie au niveau des termes de preuves. Lorsque dans le chapitre suivant nous introduirons le Calcul des Constructions implicite, nous verrons que le choix d'une présentation non stratifiée similaire à celle des PTS conduit naturellement à abandonner toutes les annotations de types sous les abstractions.

Règles de bonne formation de contexte

$$\frac{}{[] \vdash} \qquad \frac{\Gamma \vdash \phi : \mathsf{Prop}}{\Gamma; [x : \phi] \vdash}\ {\scriptstyle x \notin DV(\Gamma)} \qquad \frac{\Gamma \vdash K : \mathsf{Type}}{\Gamma; [\alpha : K] \vdash}\ {\scriptstyle \alpha \notin DV(\Gamma)}$$

Règles de typage des genres

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Prop} : \mathsf{Type}} \qquad \frac{\Gamma; [x : \phi] \vdash L : \mathsf{Type}}{\Gamma \vdash \Pi x : \phi \,.\, L : \mathsf{Type}} \qquad \frac{\Gamma; [\alpha : K] \vdash L : \mathsf{Type}}{\Gamma \vdash \Pi \alpha : K \,.\, L : \mathsf{Type}}$$

Règles de typage des constructeurs

$$\frac{\Gamma \vdash}{\Gamma \vdash \alpha : K}\ {\scriptstyle (\alpha : K) \in \Gamma} \qquad \frac{\Gamma; [x : \phi] \vdash \psi : \mathsf{Prop}}{\Gamma \vdash \Pi x : \phi \,.\, \psi : \mathsf{Prop}} \qquad \frac{\Gamma; [\alpha : K] \vdash \psi : \mathsf{Prop}}{\Gamma \vdash \forall \alpha : K \,.\, \psi : \mathsf{Prop}}$$

$$\frac{\Gamma; [x : \phi] \vdash \psi : L}{\Gamma \vdash \lambda x : \phi \,.\, \psi : \Pi x : \phi \,.\, L} \qquad \frac{\Gamma \vdash \psi : \Pi x : \phi \,.\, L \qquad \Gamma \vdash t : \phi}{\Gamma \vdash \psi\ t : L\{x := t\}}$$

$$\frac{\Gamma; [\alpha : K] \vdash \psi : L}{\Gamma \vdash \lambda \alpha : K \,.\, \psi : \Pi \alpha : K \,.\, L} \qquad \frac{\Gamma \vdash \psi : \Pi \alpha : K \,.\, L \qquad \Gamma \vdash \phi : K}{\Gamma \vdash \psi\ \phi : L\{\alpha := \phi\}}$$

$$\frac{\Gamma \vdash \phi : K \qquad \Gamma \vdash K' : \mathsf{Type}}{\Gamma \vdash \phi : K'}\ {\scriptstyle K =_\beta K'}$$

Règles de typage des termes de preuves

$$\frac{\Gamma \vdash}{\Gamma \vdash x : \phi}\ {\scriptstyle (x : \phi) \in \Gamma}$$

$$\frac{\Gamma; [x : \phi] \vdash t : \psi}{\Gamma \vdash \lambda x \,.\, t : \Pi x : \phi \,.\, \psi} \qquad \frac{\Gamma \vdash t : \Pi x : \phi \,.\, \psi \qquad \Gamma \vdash u : \phi}{\Gamma \vdash t\ u : \psi\{x := u\}}$$

$$\frac{\Gamma; [\alpha : K] \vdash t : \psi}{\Gamma \vdash t : \forall \alpha : K \,.\, \psi} \qquad \frac{\Gamma \vdash t : \forall \alpha : K \,.\, \psi \qquad \Gamma \vdash \phi : K}{\Gamma \vdash t : \psi\{\alpha := \phi\}}$$

$$\frac{\Gamma \vdash t : \phi \qquad \Gamma \vdash \phi' : \mathsf{Prop}}{\Gamma \vdash t : \phi'}\ {\scriptstyle \phi =_\beta \phi'}$$

FIG. 1.8 – Règles de typage des *Type Assignment Systems*

$\Pi\alpha : K \,.\, \psi$ par la construction $\forall\alpha : K \,.\, \psi$ et à effacer dans les termes de preuves chacune des abstractions et applications de constructeur ainsi que toutes les annotations de type sous le symbole λ. Formellement, cette fonction est définie par :

Constructeurs	$\lvert\Pi\alpha : K \,.\, \psi\rvert$	$\equiv$	$\forall\alpha : \lvert K\rvert \,.\, \lvert\psi\rvert$			
Termes de preuves	$\lvert\lambda x : \phi \,.\, t\rvert$	$\equiv$	$\lambda x \,.\, \lvert t\rvert$			
	$\lvert\lambda\alpha : K \,.\, t\rvert$	$\equiv$	$\lvert t\rvert$	$\lvert t\ \phi\rvert$	$\equiv$	$\lvert t\rvert$

(Dans tous les autres cas la définition est triviale puisqu'elle consiste simplement à propager le processus d'effacement aux sous-termes directs.)

Comme dans le cas du système F, la fonction d'effacement généralisée s'étend aux dérivations, en associant à chaque dérivation π effectuée dans l'un des huit systèmes du cube de Barendregt une dérivation $\lvert\pi\rvert$ effectuée dans le TAS correspondant (*i.e.* construit à partir des mêmes règles de formation), d'où il ressort que :

- (*Correction de la fonction d'effacement*) Si dans l'un des huit systèmes du cube de Barendregt π est une dérivation du jugement $\Gamma \vdash M : T$, alors $\lvert\pi\rvert$ est une dérivation du jugement $\lvert\Gamma\rvert \vdash \lvert M\rvert : \lvert T\rvert$ dans le TAS correspondant.

Correspondance entre les deux cubes Contrairement à la fonction d'effacement du système F, la fonction d'effacement généralisée des TAS ne définit pas un isomorphisme pour chacun des systèmes du cube, mais seulement pour les systèmes du cube sans types dépendants (*i.e.* les systèmes situés sur la face gauche des deux cubes).

Pour comprendre ce problème, nous devons d'abord distinguer la notion de similarité de la notion d'isomorphisme. Soient $\mathcal{S}_0$ l'un des huit systèmes du cube de Barendregt et $\mathcal{S}$ le TAS correspondant. On dit que $\mathcal{S}_0$ et $\mathcal{S}$ sont

- *similaires* si pour tout jugement $\Gamma \vdash M : T$ dérivable dans $\mathcal{S}$, il existe dans $\mathcal{S}_0$ un jugement dérivable $\Gamma_0 \vdash M_0 : T_0$ tel que $\lvert\Gamma_0\rvert = \Gamma$, $\lvert M_0\rvert = M$ et $\lvert T_0\rvert = T$;
- *isomorphes* si pour toute dérivation de typage π effectuée dans $\mathcal{S}$, il existe dans $\mathcal{S}_0$ une dérivation de typage π_0 telle que $\lvert\pi_0\rvert = \pi$.

Intuitivement, la notion de similarité exprime que n'importe quel jugement dérivable dans $\mathcal{S}$ peut être relevé en un jugement dérivable dans $\mathcal{S}_0$, indépendamment de la structure des dérivations. La notion d'isomorphisme est beaucoup plus forte, puisqu'elle exprime la possibilité de relever dans $\mathcal{S}_0$ non pas les jugements dérivables de $\mathcal{S}$, mais les dérivations de $\mathcal{S}$ elles-mêmes (ce qui entraîne bien entendu que les systèmes $\mathcal{S}_0$ et $\mathcal{S}$ sont similaires).

Les systèmes du cube des TAS sont reliés aux systèmes du cube de Barendregt par le théorème suivant, dont on trouvera la démonstration dans [60] :

Théorème 1.4.1 — *Soient $\mathcal{S}_0$ un des huit systèmes du cube de Barendregt et $\mathcal{S}$ le TAS construit à partir des mêmes règles. On a les trois situations suivantes :*

1. *si $\mathcal{S}_0$ et $\mathcal{S}$ sont des systèmes sans types dépendants, alors $\mathcal{S}_0$ et $\mathcal{S}$ sont isomorphes ;*
2. *si $\mathcal{S}_0$ et $\mathcal{S}$ sont des systèmes avec types dépendants mais sans polymorphisme, alors $\mathcal{S}_0$ et $\mathcal{S}$ sont similaires mais non isomorphes ;*
3. *si $\mathcal{S}_0$ et $\mathcal{S}$ sont des systèmes avec types dépendants et polymorphisme, alors $\mathcal{S}_0$ et $\mathcal{S}$ ne sont ni isomorphes ni même similaires.*

Dans le cas où les systèmes $\mathcal{S}_0$ et $\mathcal{S}$ ne comportent pas de types dépendants — ce qui est le cas en particulier des systèmes F à la Church et à la Curry — l'isomorphisme entre les deux systèmes est évident. En l'absence de types dépendants, les genres et les constructeurs ne dépendent plus des termes de preuves, et ont par conséquent la même syntaxe, les mêmes règles de typage et finalement les mêmes dérivations de typage dans les systèmes $\mathcal{S}_0$ et $\mathcal{S}$ (mis à part le changement de notation $\Pi/\forall$ dans la syntaxe des constructeurs). La seule différence entre les systèmes $\mathcal{S}_0$ et $\mathcal{S}$ apparaît donc au niveau des dérivations des jugements de la forme $\Gamma \vdash t : \phi$, et pour montrer que les dérivations de typage des termes de preuves effectuées dans $\mathcal{S}$ peuvent être relevées dans $\mathcal{S}_0$, on procède de la même manière que pour le système F à la Curry en utilisant le fait que les règles de typage correspondantes n'imposent des contraintes que sur les constructeurs, qui ont la même structure dans les deux systèmes.

En revanche, le raisonnement que nous venons d'effectuer ne fonctionne plus en présence de types dépendants, car les règles de typage imposent — notamment par le biais de la règle de conversion — des contraintes sur les termes de preuves, lesquels n'ont pas la même structure dans les deux systèmes.

Permissivité de la règle de conversion Dans le cas où $\mathcal{S}$ et $\mathcal{S}_0$ sont formés à partir des règles $(\mathsf{Type}, \mathsf{Prop})$ et $(\mathsf{Prop}, \mathsf{Type})$, il est même possible de construire un jugement dérivable dans $\mathcal{S}$ qui ne provient d'aucun jugement dérivable dans $\mathcal{S}_0$ par la fonction d'effacement. L'idée sur laquelle est bâti un tel contre-exemple est la suivante :

1. En utilisant la règle de polymorphisme, on construit dans $\mathcal{S}_0$ deux termes de preuves t_1 et t_2 de type ϕ qui ne sont pas β-convertibles, mais qui le deviennent après effacement, c'est-à-dire tels que $t_1 \neq_\beta t_2$ et $|t_1| =_\beta |t_2|$. (Nous en avons donné un exemple à la fin du paragraphe précédent, dans le cas des systèmes F à la Church et à la Curry.)
2. Quitte à introduire des K-radicaux dans t_1 et t_2, il est possible de faire en sorte que les jugements $\vdash |t_1| : |\phi|$ et $\vdash |t_2| : |\phi|$ (qui sont dérivables dans $\mathcal{S}$) ne puissent provenir que des jugements $\vdash t_1 : \phi$ et $\vdash t_2 : \phi$ (dérivables dans $\mathcal{S}_0$) respectivement. Nous ne donnerons pas ici la technique qui permet de forcer l'unicité du relèvement des termes $|t_1|$ et $|t_2|$, dont on trouvera une explication détaillée dans [60].
3. En utilisant la règle des types dépendants, on introduit alors dans $\mathcal{S}_0$ une variable de constructeur α de type $\phi \to \mathsf{Prop}$, qui permet de former deux types dépendants αt_1 et αt_2 qui ne sont pas convertibles dans $\mathcal{S}_0$, mais tels que $|\alpha t_1| =_\beta |\alpha t_2|$ (dans $\mathcal{S}$). À ce stade, nous avons formé par le jeu des dépendances deux types convertibles dans $\mathcal{S}$ qui proviennent de deux types non-convertibles dans $\mathcal{S}_0$. Les contraintes posées dans le 2ème point font même en sorte qu'il n'y a pas d'autre relèvement possible.
4. Il est alors facile de conclure en utilisant la règle de conversion. Pour cela, on introduit dans $\mathcal{S}$ une variable x de type $|\alpha t_1|$ et une variable f de type $|\alpha t_2| \to \beta$, où β est un type arbitraire, ce qui permet de dériver dans $\mathcal{S}$ le jugement

$$[\alpha : |\phi| \to \mathsf{Prop};\ \beta : \mathsf{Prop};\ x : |\alpha t_1|;\ f : |\alpha t_2| \to \beta] \quad \vdash \quad f\ x \quad : \quad \beta$$

   en utilisant le fait que les types $|\alpha t_1|$ et $|\alpha t_2|$ sont β-convertibles. Bien entendu, les contraintes que nous avons imposées à ces deux types font que le jugement ci-dessus ne peut pas être relevé dans $\mathcal{S}_0$, précisément parce que l'instance de la règle de conversion qui a permis de le dériver ne fonctionne plus dans $\mathcal{S}_0$ puisque les types αt_1 et αt_2 (qui sont les seuls relèvements possibles des types $|\alpha t_1|$ et $|\alpha t_2|$ d'après le 2ème point) ne sont pas β-convertibles.

Intuitivement, le schéma de démonstration qui précède montre que dans les TAS polymorphes avec types dépendants, la règle de conversion devient plus permissive que dans le cube de Barendregt, car elle identifie des types qui ne sont pas convertibles dans le système à la Church correspondant. Pour cette raison, les TAS permettent généralement de typer plus de termes que dans les systèmes de types à la Church correspondants.

Propriétés des TAS Les TAS vérifient essentiellement les mêmes propriétés syntaxiques que les systèmes du cube de Barendregt (affaiblissement, substitutivité, β-autoréduction, etc.) mis à part bien entendu la propriété d'unicité du type, qui n'est plus vérifiée dans ce cadre. En particulier, la propriété de normalisation forte est toujours vraie :

Théorème 1.4.2 (Normalisation forte) — *Dans chacun des huit systèmes du cube des TAS, tous les termes bien typés sont fortement normalisables.*

Dans le cas non-dépendant (*i.e.* la face gauche du cube des TAS), ce résultat découle directement de la propriété d'isomorphisme, qui permet également de relever les séquences de β-réductions dans le système à la Church correspondant.[36] Dans le cas dépendant, on se ramène au cas non-dépendant en introduisant une fonction d'effacement des dépendances [27, 60] qui envoie les termes de $DF\omega$ dans $F\omega$ et dont la définition est très similaire à celle qui, dans le cube de Barendregt, envoie les termes du Calcul des Constructions dans $\lambda\omega$ [26].

1.4.3 *Type Assignment Systems* et arguments implicites

À ce stade il est important de comparer l'approche des *Type Assignment Systems* avec les systèmes d'arguments implicites que nous avons décrits dans la section 1.3, dans la mesure où ces deux approches reposent toutes les deux sur la distinction de deux formes de produit dépendant, à savoir :

- une forme *explicite*, dans laquelle les objets de ce type sont construits par abstraction et déstructurés par application (*i.e.* le produit dépendant explicite $\Pi x : T \,.\, U$ des systèmes d'arguments implicites, qui s'utilise de la même manière que les produits dépendants $\Pi x : \phi \,.\, \psi$ $(\mathsf{Prop}, \mathsf{Prop})$, $\Pi x : \phi \,.\, L$ $(\mathsf{Prop}, \mathsf{Type})$ et $\Pi \alpha : K \,.\, L$ $(\mathsf{Type}, \mathsf{Type})$ dans les TAS) ;
- une forme *implicite*, dans laquelle la généralisation et l'instanciation interviennent de manière automatique sans que cela se traduise par une quelconque modification du terme de preuve (*i.e.* le produit dépendant implicite $\Pi\, x \mid T \,.\, U$ des systèmes d'arguments implicites, qui s'utilise de la même manière que le produit dépendant imprédicatif $\forall \alpha : K \,.\, \psi$ des TAS, formé par la règle $(\mathsf{Type}, \mathsf{Prop})$).

Bien que ces deux approches traitent la question des arguments implicites d'une manière assez différente — l'une est basée sur une distinction syntaxique effectuée *a priori* et l'autre sur la stratification des termes dans le cube de Barendregt — elles soulèvent toutes les deux un même problème, qui est que la fonction d'effacement, en faisant disparaître les informations implicites dans les termes, identifie des types qui à l'origine n'étaient pas convertibles, au risque bien entendu de changer en profondeur la sémantique du langage.

[36] Cette méthode de preuve de normalisation par relèvement (*cf* note 34 page 45) n'utilise que la propriété de similarité, et reste donc valable pour les systèmes avec types dépendants mais sans polymorphisme (*i.e.* les systèmes $DF1$ et DF').

Dans le cadre du système d'arguments implicites proposé par M. Hagiya et Y. Toda (*cf* paragraphe 1.3.3), le problème est évité au moyen d'un certain nombre de restrictions syntaxiques très fortes assurant l'existence et l'unicité du relèvement dans le calcul bicolore (ce qui entraîne naturellement la conservation de la sémantique).[37] Dans le cas des *Type Assignment Systems* en revanche, le problème se pose *a priori* avec moins d'accuité puisque le produit implicite est confiné au polymorphisme — les trois autres formes de produits dépendants sont explicites — mais l'isomorphisme entre les présentations à la Church et à la Curry n'en est pas moins rompu en présence de types dépendants, ce qui entraîne une modification importante de la sémantique sous-jacente dans la mesure où davantage de termes peuvent être typés dans ce cadre.

1.4.4 Vers des systèmes de types purs implicites ?

Il est intéressant de noter que dans les deux cas, les auteurs considèrent que l'absence d'isomorphisme entre le système explicite (ou à la Church) et le système implicite (ou à la Curry) constitue un défaut du formalisme qu'il convient de corriger. Ainsi, dans [60], S. van Bakel *et al.* proposent de réintroduire les termes de preuves à la Church à chaque fois que ceux-ci interviennent en position de dépendance (*i.e.* en tant que sous-termes d'un genre ou d'un constructeur), ce qui permet de conserver le même test de conversion que dans le système à la Church correspondant, et de rétablir ainsi l'isomorphisme entre les deux présentations.[38]

Dans les deux premières parties de cette thèse, je vais adopter un point de vue très différent qui est le suivant. Dans les systèmes d'arguments implicites comme dans les *Type Assignment Systems*, le produit dépendant implicite — que nous noterons désormais $\forall x : T \,.\, U$ — se comporte bien plus comme un constructeur d'intersection que comme un constructeur d'espace de fonctions :

$$\Pi x : T \,.\, U(x) \quad \equiv \quad \prod_{x \in T} U(x) \qquad\qquad \forall x : T \,.\, U(x) \quad \equiv \quad \bigcap_{x \in T} U(x).$$

Si on généralise la construction $\forall x : T \,.\, U$ à tous les niveaux (sans la confiner au cadre du polymorphisme imprédicatif), cette construction n'a plus aucune raison d'être isomorphe au produit dépendant usuel $\Pi x : T \,.\, U$, de même que le produit cartésien binaire $A \times B$ n'est en général pas isomorphe à l'intersection binaire $A \cap B$. Autrement dit, le produit dépendant implicite n'est pas un espace de fonctions cachées — ce qui est le point de vue sous-jacent des systèmes d'arguments implicites et des TAS, en raison de l'existence même de la fonction d'effacement — mais une construction primitive différente du produit dépendant, et qui ne peut pas s'y ramener.[39]

[37] Un tel critère d'existence et d'unicité de la reconstruction n'est pas satisfait par le système d'arguments implicites du système Coq, ce qui ne pose dans ce cadre aucun problème puisque l'information implicite est conservée et prise en compte lors du test de conversion. Il est cependant facile de vérifier que si l'on décidait d'ignorer cette information lors du test de conversion, on aboutirait à un système non normalisable et logiquement incohérent.

[38] Bien entendu, cette variante des TAS est en fait une restriction, puisqu'elle permet *in fine* de typer moins de termes que dans la présentation originale.

[39] Dans une certaine mesure, le fait qu'il y ait isomorphisme entre les faces gauches des cubes de Barendregt et des TAS peut être vu comme une simple coïncidence liée au fait que dans le cas imprédicatif, le produit dépendant construit un espace de fonctions qui ne dépendent pas réellement de leur argument de type (*i.e.* la paramétricité), et peut être assimilé de ce fait à une intersection (en ignorant l'argument de type, qui n'a de fait aucune incidence sur le calcul proprement dit). Cette intuition est corroborée par le fait que la définition

Afin d'étudier les propriétés du produit dépendant implicite $\forall x : T \,.\, U$, nous allons introduire dans le chapitre suivant une extension du Calcul des Constructions avec univers[40] — appelée *Calcul des Constructions implicite* — munie de cette nouvelle construction primitive. Contrairement aux *Type Assignment Systems*, les deux formes de produits dépendants — explicite et implicite — pourront être utilisées à n'importe quel endroit de la hiérarchie d'univers. Syntaxiquement, le Calcul des Constructions implicite est très proche du calcul implicite de M. Hagiya et Y. Toda bien que dans son esprit, il soit sans doute davantage inspiré de l'approche des *Type Assignment Systems*.

Pourtant, contrairement à ces deux dernières approches, le Calcul des Constructions implicite est un formalisme à la Curry sans équivalent à la Church. L'absence d'une contrepartie *explicite* — munie d'une fonction d'effacement qui nous permettrait de passer d'un formalisme à l'autre — est la conséquence la plus notable des observations que nous avons faites plus haut concernant la dissymétrie fondamentale entre les deux formes de produits dépendants. Dans ce qui suit, nous ne chercherons donc pas à nous ramener à un hypothétique formalisme à la Church — tant il semble clair que la généralisation du produit implicite à toute la hiérarchie d'univers interdit définitivement tout isomorphisme syntaxique avec un fragment du calcul bicolore[41] — mais nous essaierons au contraire d'exploiter les identifications de types induites par la présence du produit dépendant implicite, en étudiant plus particulièrement la riche relation de sous-typage à laquelle cette nouvelle construction donne lieu.

Bien entendu, le passage d'un point de vue à la Church — qui domine dans la tradition de la théorie des types — à un point de vue à la Curry aussi radical ne peut pas s'effectuer sans un changement profond de la sémantique sous-jacente. Dans la seconde partie de cette thèse, nous aurons l'occasion d'étudier les modèles du Calcul des Constructions implicite, dont nous verrons que la construction ne peut pas s'effectuer à partir des modèles standards du Calcul des Constructions avec univers, mais nécessite d'introduire de nouveaux outils basés sur la théorie des espaces cohérents (pour interpréter en particulier la relation de sous-typage).

Il est intéressant de revenir quelques instants sur la question des arguments implicites, puisque c'est elle qui a à l'origine motivé l'introduction de ce formalisme. Dans sa version actuelle, le Calcul des Constructions implicite ne peut sans doute pas servir de base à une implémentation réaliste d'un système d'arguments implicites dans la mesure où nous conjec-

---

de la fonction d'effacement ne pose aucun problème dans les TAS — contrairement aux systèmes d'arguments implicites où la fonction d'effacement est définie partiellement (*cf* paragraphe 1.3.3) — puisque toutes les variables de constructeurs introduites par les abstractions effaçables (*i.e.* polymorphes) sont systématiquement utilisées dans les termes de preuve en position effaçable (dans les applications de constructeur et dans les annotations de type sous les abstractions).

[40] Il serait plus correct de dire que le Calcul des Constructions implicite est une *variante* du Calcul des Constructions avec univers, puisque le premier dispose d'une abstraction à la Curry (qui le rend de ce fait plus proche des *domain-free pure type systems* [12]) alors que le second dispose d'une abstraction à la Church. Cependant, on vérifiera aisément que cette différence mise à part, le calcul implicite que nous allons présenter est une extension du Calcul des Constructions avec univers dans la mesure où il contient tous les jugements dérivables de ce formalisme quitte à y effacer toutes les annotations de types qui figurent dans les abstractions.

[41] On pourra vérifier en particulier que tous les contre-exemples illustrant la disparition de l'isomorphisme entre les présentations à la Church et à la Curry dans le cadre des TAS s'adaptent sans difficulté au cadre du Calcul des Constructions implicite, qui contient de manière immédiate cette famille de formalismes. Dans le calcul implicite, le relèvement des jugements de typage dans le calcul bicolore est bien plus difficile que dans les TAS puisque le produit dépendant implicite n'est pas présent seulement au niveau imprédicatif mais dans tous les niveaux de la hiérarchie d'univers, et également en raison du rôle central joué par la règle η dans ce formalisme.

turons que la vérification de type est (sans doute très fortement) indécidable.[42] Cependant, l'étude que nous allons entreprendre montre clairement que la disparition complète des informations implicites dans les termes soulève des problèmes plus profonds que les problèmes de reconstruction et d'inférence considérés par les approches décrites dans la section 1.3. Dans la suite de cette thèse, nous nous intéresserons davantage aux aspects sémantiques (normalisation forte et cohérence) soulevés par cette approche, et nous laisserons de côté les problèmes pratiques liés à la décidabilité de la vérification de type. Toutefois, il serait sans doute intéressant de définir un fragment du calcul implicite (sans doute accompagné d'annotations supplémentaires) pour lequel la vérification de type serait décidable, et d'étudier dans quelle mesure ce fragment pourrait servir de base à un système d'arguments implicites réaliste.

Pour terminer ce chapitre, il est important de préciser que le Calcul des Constructions implicite est un formalisme reposant sur une présentation non-stratifiée (contrairement aux TAS) et dont les règles de typage sont dans une large mesure indépendantes de la hiérarchie d'univers. Pour cette raison, cette approche semble être facilement généralisable à tous les systèmes de types purs (sans annotations de types sous les abstractions), ce qui pourrait donner lieu à des développements intéressants sur une théorie des *domain-free pure type systems* étendus à l'aide d'un produit implicite, que l'on pourrait qualifier naturellement de *systèmes de types purs implicites*.

[42] Ce qui semble relativement clair, puisque ce formalisme contient le système F à la Curry (et même $F\eta$) ainsi que la théorie des types de Martin-Löf sans annotations de types sous les abstractions, qui sont tous les deux des systèmes pour lesquels la vérification de type est indécidable [62, 21].

Chapitre 2

Le Calcul des Constructions implicite

2.1 Syntaxe

D'un point de vue syntaxique, le Calcul des Constructions implicite (CCI) — ou calcul implicite — est une variante du Calcul des Constructions avec univers (CCω)[1] dans laquelle on distingue deux formes de produits : le *produit explicite* et le *produit implicite*, notés respectivement $\Pi x : T \,.\, U$ et $\forall x : T \,.\, U$. Une autre différence syntaxique avec CCω est que le calcul implicite utilise une λ-abstraction à la Curry (*i.e.* sans annotation de type) contrairement à la tradition des PTS où domine un usage exclusif de la λ-abstraction à la Church (*i.e.* avec annotation de type). La syntaxe complète du calcul est récapitulée dans la figure 2.1.

Sortes	s	::=	$\mathsf{Prop} \mid \mathsf{Set} \mid \mathsf{Type}_i$	$(i > 0)$
Termes	M, N, T, U	::=	x	(Variable)
		$\mid$	s	(Sorte)
		$\mid$	$\Pi x : T \,.\, U$	(Produit explicite)
		$\mid$	$\forall x : T \,.\, U$	(Produit implicite)
		$\mid$	$\lambda x \,.\, M$	(Abstraction)
		$\mid$	$(M\ N)$	(Application)

FIG. 2.1 – Syntaxe du Calcul des Constructions implicite

2.1.1 Les sortes

Le Calcul des Constructions implicite est bâti sur le même ensemble de *sortes* (ou *univers*) que le Calcul des Constructions Inductives [63, 53]. Cet ensemble (dénombrable), noté $\mathcal{S}$, est défini par

$$\mathcal{S} \;=\; \{\mathsf{Prop};\ \mathsf{Set}\} \;\cup\; \{\mathsf{Type}_i \mid i > 0\}.$$

[1] C'est-à-dire du Calcul des Constructions [20, 26] auquel on a rajouté une hiérarchie infinie d'univers prédicatifs emboîtés, notés Type_i (pour $i \in \{1;\ 2;\ 3;\ \ldots\}$), dans laquelle la sorte Type_1 joue le rôle de la sorte Type du Calcul des Constructions originel. Le Calcul des Constructions avec univers est lui-même un sous-système du Calcul des Constructions Étendu (ECC) introduit par Luo [42], qui comporte en outre un constructeur de *sigma-types* — types sommes dépendantes — lequel est absent du Calcul des Constructions comme du Calcul des Constructions implicite.

Les sortes imprédicatives Prop et Set Comme dans le Calcul des Constructions Inductives [63, 53], le calcul implicite comporte deux sortes imprédicatives Prop et Set : la première (Prop) est la sorte des types propositionnels tandis que la seconde (Set) est la sorte des types de données imprédicatifs. Bien entendu, cette distinction est ici purement formelle puisque, du point de vue des règles de typage, les sortes Prop et Set sont isomorphes. Toutefois, nous verrons au chapitre 6 qu'il peut être intéressant d'interpréter ces deux univers de manière différente.

La hiérarchie des univers prédicatifs Type_i La famille de sortes $(\mathsf{Type}_i)_{i>0}$ correspond à la famille d'univers prédicatifs[2] incorporée au Calcul des Constructions par Luo [42]. Cette famille forme une *hiérarchie d'univers* puisque Prop et Set sont des objets de type Type_1, Type_1 est de type Type_2, et ainsi de suite. De plus, cette hiérarchie est *cumulative*, puisque tous les objets de type Prop ou Set sont également de type Type_1, ceux de type Type_1 sont également de type Type_2, etc.

2.1.2 Les termes

Dans ce qui suit, on désigne par $\mathcal{V}$ l'ensemble (dénombrable) des variables et par Λ_{CCI} l'ensemble des termes du calcul implicite. On trouvera une correspondance informelle entre les constructions du calcul implicite et les constructions analogues en théorie des ensembles dans la figure 2.2 (où nous avons fait figurer les dépendances par des variables en indice).

Calcul implicite		**Interprétation naïve**	
Produit explicite	$\Pi x : T \,.\, U_x$	Produit cartésien	$\prod_{x \in T} U_x$
Produit implicite	$\forall x : T \,.\, U_x$	Intersection	$\bigcap_{x \in T} U_x$
Abstraction	$\lambda x \,.\, M_x$	Fonction	$x \mapsto M_x$
Application	$(M\ N)$	Application	$M(N)$

FIG. 2.2 – Signification informelle des termes de CCI

Insistons sur le fait que cette correspondance est pour le moment purement informelle : en particulier, la notation fonctionnelle $(x \mapsto M_x)$ ne fait pas de sens en théorie des ensembles lorsqu'elle n'est pas accompagnée du domaine de définition correspondant.

Notations et abréviations Dans l'étude du calcul implicite, nous reprendrons la plupart des notations et conventions du λ-calcul pur. L'ensemble des variables libres d'un terme M est noté $FV(M)$, sachant que les constructions $\Pi x : T \,.\, U$ et $\forall x : T \,.\, U$ lient toutes les occurrences de la variable x dans leur sous-terme U (mais aucune des occurrences de la variable x dans T) de même que la construction $\lambda x \,.\, M$ lie toutes les occurrences libres de x dans son sous-terme M. L'opération (externe) de substitution est notée $M\{x := N\}$, et désigne le terme obtenu en remplaçant dans M toutes les occurrences libres de la variable x par le terme N. Conformément à l'usage, nous identifierons les termes *α-convertibles*, et nous renommerons

[2] Nous reprenons ici la convention de [64] consistant à numéroter les univers prédicatifs à partir de l'entier 1. On désignera parfois la sorte (imprédicative) Prop par Type_0.

silencieusement les variables liées chaque fois que cela sera nécessaire, de manière à éviter les phénomènes de capture de variable.

On notera l'application associativement à gauche, en supprimant les parenthèses extérieures chaque fois que cela sera possible

$$M\ N_1\ N_2\ \cdots\ N_p \quad \equiv \quad (\cdots((M\ N_1)\ N_2)\ \cdots\ N_p)$$

et on factorisera les abstractions consécutives[3]

$$\lambda x_1 x_2 \ldots x_n \,.\, M \quad \equiv \quad \lambda x_1 \,.\, \lambda x_2 \,.\, \ldots \lambda x_n \,.\, M$$

de même que les produits consécutifs de même nature lorsque les types T sur lesquels portent les quantifications sont les mêmes :

$$\begin{aligned} \Pi x_1, x_2, \ldots, x_n : T \,.\, U &\equiv \Pi x_1 : T \,.\, \Pi x_2 : T \,.\, \cdots \Pi x_n : T \,.\, U \\ \forall x_1, x_2, \ldots, x_n : T \,.\, U &\equiv \forall x_1 : T \,.\, \forall x_2 : T \,.\, \cdots \forall x_n : T \,.\, U \end{aligned}$$

L'application a priorité sur toute autre construction. Ainsi, la notation $\lambda x \,.\, y\ z$ désigne le terme $\lambda x \,.\, (y\ z)$ et non $((\lambda x \,.\, y)\ z)$. Enfin, le produit explicite non-dépendant $\Pi x : T \,.\, U$ (*i.e.* tel que $x \notin FV(U)$) est noté $T \to U$. La notation "flèche" est associative à droite

$$T_1 \to T_2 \to \cdots \to T_n \to U \quad \equiv \quad T_1 \to (T_2 \to \cdots \to (T_n \to U) \cdots)$$

et prioritaire vis-à-vis des lieurs Π, $\forall$ et λ.

2.2 Réduction

2.2.1 Les règles β et η

Comme dans la théorie du λ-calcul pur, la notion de calcul afférente au Calcul des Constructions implicite est basée sur les règles de β et de η-réduction. Ces règles de réduction sont construites à partir des relations binaires $\triangleright_\beta$ (*β-réduction à la racine*) et $\triangleright_\eta$ (*η-réduction à la racine*) définies par

$$\begin{aligned} (\lambda x \,.\, M)\ N \quad &\triangleright_\beta \quad M\{x := N\} \\ \lambda x \,.\, M\ x \quad &\triangleright_\eta \quad M \quad (\text{si } x \notin FV(M)) \end{aligned}$$

lorsque M et N décrivent l'ensemble Λ_{CCI} (avec la restriction $x \notin FV(M)$ dans le cas de la notion de η-réduction). On définit également la relation $\triangleright_{\beta\eta}$ comme étant la réunion des relations $\triangleright_\beta$ et $\triangleright_\eta$. Autrement dit :

$$M \triangleright_{\beta\eta} M' \qquad \Leftrightarrow \qquad M \triangleright_\beta M' \quad \text{ou} \quad M \triangleright_\eta M'$$

Pour chaque règle $R \in \{\beta;\ \eta;\ \beta\eta\}$ et pour chaque couple de termes (M, M') tels que $M \triangleright_R M'$, on dit que M est un *R-radical*, et que M' est le *R-contracté* de M.

Soit $R \in \{\beta;\ \eta;\ \beta\eta\}$. On appelle

– *R-réduction en une étape* la clôture contextuelle de la relation $\triangleright_R$, notée $\to_R$;

[3] Il nous arrivera fréquemment de séparer les variables par des virgules dans les abstractions groupées (*i.e.* $\lambda x_1, \ldots, x_n \,.\, M$) de manière à améliorer la lisibilité.

– *R-réduction en un nombre arbitraire d'étapes* la clôture réflexive et transitive de la relation $\rightarrow_R$, notée $\twoheadrightarrow_R$ (*i.e.* la relation de préordre engendrée par $\rightarrow_R$) ;
– *R-convertibilité* la clôture réflexive, symétrique et transitive de la relation $\rightarrow_R$, notée $=_R$ (*i.e.* la relation d'équivalence engendrée par $\rightarrow_R$).

Les règles β, η et $\beta\eta$ sont confluentes et satisfont la propriété de Church-Rosser :

Proposition 2.2.1 (Confluence) — *Pour chacune des règles $R \in \{\beta; \eta; \beta\eta\}$ on a*

$$M \twoheadrightarrow_R M_1 \quad et \quad M \twoheadrightarrow_R M_2 \qquad \Rightarrow \qquad \exists\, M' \quad M_1 \twoheadrightarrow_R M' \quad et \quad M_2 \twoheadrightarrow_R M'.$$

Proposition 2.2.2 (Church-Rosser) — *Pour chacune des règles $R \in \{\beta; \eta; \beta\eta\}$ on a*

$$M_1 =_R M_2 \qquad \Leftrightarrow \qquad \exists\, M' \quad M_1 \twoheadrightarrow_R M' \quad et \quad M_2 \twoheadrightarrow_R M'.$$

Mentionnons également une autre propriété des règles β et η qui nous servira de manière centrale dans la preuve de préservation du typage par la β-réduction :

Proposition 2.2.3 (Report de la règle η)

$$M \twoheadrightarrow_{\beta\eta} M' \qquad \Rightarrow \qquad \exists\, M_0 \quad M \twoheadrightarrow_\beta M_0 \quad et \quad M_0 \twoheadrightarrow_\eta M'.$$

(La preuve de ces trois propriétés sera effectuée au paragraphe suivant.)

2.2.2 Traductions vers le λ-calcul

En raison de l'absence d'annotation de type sous l'abstraction, la structure du calcul implicite est très proche de celle du λ-calcul pur, hormis la présence des lieurs $\Pi x : T \,.\, U$ et $\forall x : T \,.\, U$ ainsi que les constantes de sortes. Ceci suggère naturellement une traduction des termes du calcul implicite vers un λ-calcul avec constantes, laquelle nous permettra de ramener la preuve des propriétés syntaxiques du calcul implicite énoncées dans le paragraphe précédent aux preuves des propriétés correspondantes dans le λ-calcul.

Pour cela, considérons un λ-calul pur étendu par un jeu (fini ou dénombrable) de constantes inertes (c_i), que nous noterons Λ_{c_i}. Vis à vis de la $\beta\eta$-réduction, les constantes $c_i \in \Lambda_{c_i}$ se comportent exactement comme des variables libres du λ-calcul, d'où il ressort que dans Λ_{c_i}, les propriétés de confluence et de Church-Rosser sont satisfaites par les règles β, η et $\beta\eta$, de même que la propriété de report de la règle η [8].

Considérons à présent une *traduction* ϕ du calcul implicite dans Λ_{c_i}, c'est-à-dire une application qui à chaque terme $M \in \Lambda_{\mathrm{CCI}}$ associe un terme $\phi(M) \in \Lambda_{c_i}$. Pour chaque règle $R \in \{\beta; \eta; \beta\eta\}$, on dira que ϕ est

- un *morphisme* vis-à-vis de R si pour tous termes M et M' du calcul implicite, $M \rightarrow_R M'$ entraîne $\phi(M) \rightarrow_R \phi(M')$;
- un *plongement* vis-à-vis de R si ϕ est un morphisme injectif satisfaisant la propriété de *relèvement* suivante :

$$\forall\, M \in \Lambda_{\mathrm{CCI}},\ m' \in \Lambda_{c_i} \qquad \phi(M) \twoheadrightarrow_R m' \quad \Rightarrow \quad \exists\, M' \quad M \twoheadrightarrow_R M' \ \text{ et } \ \phi(M') = m'.$$

Si ϕ est un plongement vis-à-vis d'une règle $R \in \{\beta; \eta; \beta\eta\}$, alors l'arbre de R-réduction issu d'un terme M du calcul implicite est isomorphe à l'arbre de R-réduction issu de son image $\phi(M)$ dans Λ_{c_i}, d'où il ressort immédiatement que :

Lemme 2.2.4 — *S'il existe une traduction $\phi : \Lambda_{\mathrm{CCI}} \to \Lambda_{c_i}$ qui est un plongement vis-à-vis de la règle R, alors la règle R est confluente dans le calcul implicite, et satisfait la propriété de Church-Rosser.*

Lemme 2.2.5 — *S'il existe une traduction $\phi : \Lambda_{\mathrm{CCI}} \to \Lambda_{c_i}$ qui est un plongement vis-à-vis des règles β et η, alors le calcul implicite satisfait la propriété de report de la règle η par rapport à la règle β.*

Nous allons à présent définir non pas une traduction, mais deux traductions du calcul implicite vers des λ-calculs avec constantes.

La première traduction consiste simplement à interpréter les constructions $\Pi x : T \,.\, U$ et $\forall x : T \,.\, U$ comme résultant de l'application de combinateurs de types Π et $\forall$ au type T et à la fonctionnelle $\lambda x \,.\, U$. Si cette traduction est très naturelle — nous aurons l'occasion d'y revenir au chapitre 3 lors de l'étude de la sémantique du calcul implicite — elle présente le défaut structurel de n'être un plongement que vis-à-vis de la règle β.

Afin de remédier à ce problème, nous introduirons une seconde traduction qui nous permettra également d'étendre cette propriété de plongement vis-à-vis des règles β et η, ce qui achèvera la preuve des propositions 2.2.1, 2.2.2 et 2.2.3.

Traduction vers $\Lambda_{\Pi\forall s}$ Soit $\Lambda_{\Pi\forall s}$ le λ-calcul pur étendu avec des constantes inertes Π, $\forall$ ainsi que toutes les constantes de sortes $s \in \mathcal{S}$. À chaque terme $M \in \Lambda_{\mathrm{CCI}}$ on associe un terme $M^* \in \Lambda_{\Pi\forall s}$ défini récursivement par :

$$
\begin{array}{lcl}
x^* & = & x \\
s^* & = & s \\
(\Pi x : T \,.\, U)^* & = & (\Pi\ T^*\ (\lambda x \,.\, U^*)) \\
(\forall x : T \,.\, U)^* & = & (\forall\ T^*\ (\lambda x \,.\, U^*)) \\
(\lambda x \,.\, M)^* & = & \lambda x \,.\, M^* \\
(M\ N)^* & = & (M^*\ N^*)
\end{array}
$$

Lemme 2.2.6 (Substitutivité et variables libres) — *Si M et N sont des termes du calcul implicite, et si x est une variable, alors :*

1. $FV(M^*) = FV(M)$;
2. $(M\{x := N\})^* \ = \ M^*\{x := N^*\}$.

Proposition 2.2.7 — *La traduction $M \mapsto M^*$ est un plongement vis-à-vis de la règle β.*

Proposition 2.2.8 — *La traduction $M \mapsto M^*$ est un morphisme vis-à-vis de la règle η.*

Malheureusement, la traduction $M \mapsto M^*$ n'est pas un plongement vis-à-vis de la règle de η-réduction. Un contre-exemple à la propriété de relèvement est donné par le terme $M = \Pi x : s \,.\, (f\ x)$ dont la traduction $M^* = \big(\Pi\ s\ (\lambda x \,.\, (f\ x))\big)$ se η-réduit en une étape vers le terme $(\Pi\ s\ f)$ qui n'est l'image d'aucun terme $M' \in \Lambda_{\mathrm{CCI}}$ par la traduction $M \mapsto M^*$.

Afin de remédier à cette situation, il est nécessaire d'introduire une légère modification dans le schéma de traduction de manière à empêcher l'apparition au cours de la traduction d'un η-radical qui ne proviendrait pas d'un η-radical du terme initial.

Traduction vers $\Lambda_{\Pi\forall sK}$ Soit $\Lambda_{\Pi\forall sK}$ le système obtenu en ajoutant une nouvelle constante inerte K à $\Lambda_{\Pi\forall s}$.[4] À chaque terme M du calcul implicite, on associe à présent un terme $M^\dagger \in \Lambda_{\Pi\forall sK}$ défini récursivement par :

$$\begin{array}{lcl} x^\dagger & = & x \\ s^\dagger & = & s \\ (\Pi x : T \,.\, U)^\dagger & = & (\Pi\ T^\dagger\ (\lambda x \,.\, (K\ U^\dagger\ K))) \\ (\forall x : T \,.\, U)^\dagger & = & (\forall\ T^\dagger\ (\lambda x \,.\, (K\ U^\dagger\ K))) \\ (\lambda x \,.\, M)^\dagger & = & \lambda x \,.\, M^\dagger \\ (M\ N)^\dagger & = & (M^\dagger\ N^\dagger) \end{array}$$

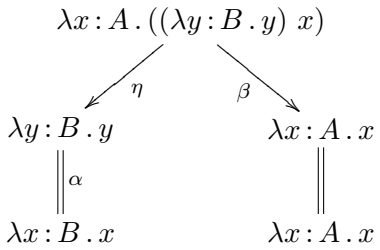
Lemme 2.2.9 (Substitutivité et variables libres) — *Si M et N sont des termes du calcul implicite, et si x est une variable, alors :*

1. $FV(M^\dagger) = FV(M)$;
2. $(M\{x := N\})^\dagger \ = \ M^\dagger\{x := N^\dagger\}$.

Proposition 2.2.10 — *La traduction $M \mapsto M^\dagger$ est un plongement vis-à-vis des trois règles β, η et $\beta\eta$.*

2.2.3 Non-confluence de la $\beta\eta$-réduction dans CCω

Du point de vue de la théorie de la réduction, la différence essentielle entre le calcul implicite et les PTS réside dans l'utilisation de la règle de η-réduction, qui n'est pas prise en compte dans le cadre des PTS. Une des raisons qui expliquent cette différence est qu'en présence de l'abstraction à la Church des PTS, la $\beta\eta$-réduction ne vérifie plus la propriété de Church-Rosser sur les termes non-typés. Ce problème est illustré par le contre-exemple suivant, dû à Nederpelt [25] :



Dans le diagramme ci-dessus, les deux chaînes de réduction ne peuvent pas converger si les termes A et B ne sont pas convertibles dans le PTS considéré. En revanche, si le terme initial $\lambda x : A \,.\, ((\lambda y : B \,.\, y)\ x)$ est bien typé, et *si le PTS est fonctionnel* [26], alors les termes A et B sont convertibles. Notons que l'hypothèse selon laquelle le PTS est fonctionnel est ici

[4] Intuitivement, la constante K correspond au combinateur $\mathbf{K} = \lambda xy \,.\, x$ de la logique combinatoire, à cette différence près que dans $\Lambda_{\Pi\forall sK}$ on n'associe aucune règle de réduction spécifique à la constante K, qui demeure inerte vis-à-vis de la $\beta\eta$-réduction.

primordiale : dans un calcul tel que CCω dans lequel on dispose d'une règle de cumulativité, la propriété de confluence de la $\beta\eta$-reduction est perdue y compris pour les termes bien typés. Un exemple de non-confluence sur les termes bien typés dans CCω est donné par le terme suivant

$$\begin{array}{rcl} \lambda x : \mathsf{Type}_1 \,.\, ((\lambda y : \mathsf{Type}_2 \,.\, y)\ x) & \to_\beta & \lambda x : \mathsf{Type}_1 \,.\, x \\ & \to_\eta & \lambda y : \mathsf{Type}_2 \,.\, y \end{array}$$

Cet exemple est important, car il montre que la notion de sous-typage (illustrée ici par l'inclusion $\mathsf{Type}_1 \subset \mathsf{Type}_2$) est incompatible avec l'annotation de type sous l'abstraction en présence de la règle $\beta\eta$.

Bien entendu, ce problème de confluence lié à la présence d'une annotation de type sous le λ n'apparaît pas dans le calcul implicite, qui est un formalisme à la Curry. Ce point a une certaine importance car, dans le calcul implicite, l'utilisation de la règle η est nécessaire pour conférer de bonnes propriétés à la relation de sous-typage induite par la présence du produit implicite (*cf* section 2.5).

2.3 Typage

2.3.1 Les ensembles Axiom et Rule

Comme dans les PTS, les règles de typage du Calcul des Constructions implicite reposent sur deux ensembles, **Axiom** $\subset \mathcal{S}^2$ et **Rule** $\subset \mathcal{S}^3$, lesquels permettent de typer respectivement les sortes et les produits dépendants. (Notons que le même ensemble **Rule** est utilisé pour former les produits explicites comme les produits implicites.)

Dans le calcul implicite, les ensembles **Axiom** et **Rule** sont définis par :

$$\begin{array}{rcl} \mathbf{Axiom} & = & \{(\mathsf{Prop}, \mathsf{Type}_1);\ (\mathsf{Set}, \mathsf{Type}_1);\ (\mathsf{Type}_i, \mathsf{Type}_{i+1}) \quad \mid i > 0\} \\ \\ \mathbf{Rule} & = & \{(s, \mathsf{Prop}, \mathsf{Prop}); (s, \mathsf{Set}, \mathsf{Set}) \quad \mid s \in \mathcal{S}\} \quad \cup \\ & & \{(\mathsf{Type}_i, \mathsf{Type}_i, \mathsf{Type}_i) \quad \mid i > 0\} \end{array}$$

Intuitivement, la relation $(s_1, s_2) \in$ **Axiom** signifie que la sorte s_1 peut être typée dans la sorte s_2, et la relation $(s_1, s_2, s_3) \in$ **Rule** signifie qu'un produit explicite $\Pi x : T \,.\, U$ ou un produit implicite $\forall x : T \,.\, U$ dont les composantes T et U habitent respectivement les sortes s_1 et s_2 peut être formé dans la sorte s_3.

2.3.2 Ordre de cumulativité

Comme dans CCω, l'ensemble des sortes est ordonné par une relation notée $s_1 \leq s_2$, appelée *ordre de cumulativité*. Cette relation, qui spécifie la relation d'inclusion entre les sortes, est définie par les inéquations :

$$\begin{array}{lll} \mathsf{Prop} \leq \mathsf{Prop} & \mathsf{Prop} \leq \mathsf{Type}_i & (i > 0) \\ \mathsf{Set} \leq \mathsf{Set} & \mathsf{Set} \leq \mathsf{Type}_i & (i > 0) \\ \mathsf{Type}_i \leq \mathsf{Type}_j & & (i, j > 0; \quad i \leq j) \end{array}$$

Remarquons que cette relation d'ordre partiel sur l'ensemble des sortes est en fait le préordre engendré par la relation $(s_1, s_2) \in$ **Axiom**. Par ailleurs, la seule paire de sortes incomparables

entre elles est la paire {Prop; Set} formée par les deux sortes imprédicatives, lesquelles se situent tout en bas de la hiérarchie d'univers.

2.3.3 Contextes de typage

Comme dans n'importe quel λ-calcul typé, la définition de la relation de typage dans le calcul implicite fait intervenir une structure auxiliaire — le *contexte de typage* — dont le rôle est d'assigner un type à chacune des variables libres rencontrées.

Une *déclaration (de type)* est un couple constitué d'une variable x et d'un terme T, que l'on note $(x : T)$. Un *contexte de typage* (ou plus simplement un *contexte*) est une liste finie (et ordonnée) de déclarations, notée

$$\Gamma \;=\; [x_1 : T_1;\; \ldots; x_n : T_n]$$

Si Γ et Δ sont deux contextes, on note $\Gamma;\Delta$ le contexte formé par la concaténation de Γ et Δ pris dans cet ordre. En particulier, si Γ est un contexte et $(x : T)$ une déclaration, $\Gamma;[x : T]$ désigne le contexte Γ dans lequel on a *chargé* l'hypothèse $(x : T)$. Le contexte vide est noté quant à lui $[\,]$. Soient Γ et Γ' deux contextes. On dit que

- Γ est un *préfixe* de Γ' s'il existe un contexte Δ tel que $\Gamma;\Delta = \Gamma'$, ce que l'on note $\Gamma \sqsubset \Gamma'$;
- Γ est un *sous-contexte* de Γ' si toute déclaration $(x : T)$ apparaissant dans Γ apparaît également dans Γ', ce que l'on note $\Gamma \subset \Gamma'$.

Notons que $\Gamma \sqsubset \Gamma'$ entraîne $\Gamma \subset \Gamma'$, la réciproque étant évidemment fausse. En particulier, les déclarations d'un sous contexte $\Gamma \subset \Gamma'$ peuvent ne pas apparaître dans le même ordre dans Γ et dans Γ'.

Si $\Gamma = [x_1 : T_1;\; \ldots;\; x_n : T_n]$ est un contexte, on note

$$\begin{array}{lcll} DV(\Gamma) & = & \{x_1;\; \ldots;\; x_n\} & \text{(variables déclarées dans } \Gamma) \\ FV(\Gamma) & = & FV(T_1) \cup \cdots \cup FV(T_n) & \text{(variables libres de } \Gamma) \end{array}$$

De même, la notation $M\{x := N\}$ est étendue aux contextes de typage en posant

$$[x_1 : T_1;\; \ldots;\; x_n : T_n]\{x := N\} \;=\; \big[x_1 : T_1\{x := N\};\; \ldots;\; x_n : T_n\{x := N\}\big],$$

cette notation n'ayant par ailleurs un sens que dans la mesure où la variable substituée n'apparaît pas dans les variables déclarées dans Γ (*i.e.* si $x \neq x_i$ pour tout $i \in \{1 \ldots n\}$). Pour chacune des trois règles de réduction β, η et $\beta\eta$, la notion de R-convertibilité est étendue aux contextes en posant

$$[x_1 : T_1; \ldots; x_n : T_n] \quad =_R \quad [x_1 : T'_1; \ldots; x_n : T'_n]$$

lorsque $T_i =_R T'_i$ pour tout $i \in [1..n]$. (Cette notation ne fait un sens que si les contextes ont même longueur, et déclarent les mêmes variables dans le même ordre.)

2.3.4 Jugements et dérivations

Le système de types du calcul implicite repose sur un ensemble de *jugements*, lesquels sont de deux formes :

- les jugements unaires notés $\Gamma \vdash$, qui se lisent : « le contexte de typage Γ est bien formé » ;

– les jugements ternaires notés $\Gamma \vdash M : T$, qui se lisent : « sous le contexte Γ, le terme M est bien formé et a pour type le terme T », ou plus simplement : « sous le contexte Γ, le terme M a le type T »

Parmi cet ensemble, on distingue un sous-ensemble appelé ensemble des *jugements dérivables* (ou encore *jugements valides*), lequel est défini à l'aide des règles d'inférence de la figure 2.3 de la page 64. Chacune de ces règles d'inférence est de la forme

$$\frac{P_1 \quad \dots \quad P_n \quad B}{C}\ (R)$$

où $P_1, \dots, P_n$ sont des jugements appelés *prémisses* de la règle (R), B une condition éventuelle appelée *condition de bord* de la règle (R), et où le jugement C situé sous le trait d'inférence est appelé la *conclusion* de la règle (R). Avec ces notations, la règle (R) se lit : « si les prémisses $P_1, \dots, P_n$ sont des jugements valides, et si la condition de bord B est satisfaite, alors le jugement C est valide ».

On appelle *arbre de dérivation* tout arbre fini π tel que
– chaque nœud de π est étiqueté par un jugement et le nom d'une règle ;
– pour tout nœud de π étiqueté par un jugement C et un nom de règle (R), dont les successeurs sont étiquetés par les jugements $P_1, \dots, P_n$, il existe une instance de la règle (R) dont les prémisses sont précisément les jugements $P_1, \dots, P_n$ et la conclusion C, et dont la condition de bord éventuelle B est satisfaite.

Si π est un arbre de dérivation et J le jugement étiquetant sa racine, on dit que π *est une dérivation de* J, ce que l'on note $\pi : J$. Lorsqu'un jugement J admet une dérivation, on dit alors que le jugement J est *dérivable*, ou qu'il est *valide*. Bien entendu, l'ensemble des jugements dérivables est le plus petit sous-ensemble de l'ensemble des jugements qui soit clos par l'ensemble des régles d'inférence de la figure 2.3.

2.3.5 Signification des règles d'inférence

Les règles de typage (Var), (Sort), (E-Prd), (I-Prd), (Lam), (App), (Conv) et (Cum) correspondent aux règles usuelles de typage de CCω dans lesquelles ont été introduites quelques modifications dues essentiellement à la présence du produit implicite et au fait que le Calcul des Constructions implicite est un formalisme à la Curry :

- La règle de formation du produit dépendant a été scindée en deux règles (E-Prd) et (I-Prd) permettant de former respectivement les produits explicites et les produits implicites. Ces deux règles partagent les mêmes prémisses et la même condition de bord.
- La règle (Lam) (formation de l'abstraction) met à présent en jeu une abstraction à la Curry. Puisqu'aucune annotation de type ne figure dans le terme $\lambda x\,.\,M$ apparaissant dans la conclusion, l'inférence du type de l'abstraction $\lambda x\,.\,M$ dans un contexte Γ nécessite de "deviner" l'assignation de type correcte $(x : T)$ qui permettra d'inférer le type de son corps M dans le contexte étendu $\Gamma; [x : T]$.
- La règle de convertibilité (Conv) prend également en compte les paires de types η-convertibles, et non plus seulement les paires de types β-convertibles comme c'est le cas dans la théorie des PTS.

Le calcul implicite s'écarte également de la présentation usuelle des PTS en remplaçant les règles (Start) et (Weak) [26] par un jugement explicite de bonne formation de contexte

$$\frac{}{[] \vdash} \text{ (WF-E)} \qquad \frac{\Gamma \vdash T : s \quad x \notin DV(\Gamma)}{\Gamma; [x : T] \vdash} \text{ (WF-S)}$$

$$\frac{\Gamma \vdash \quad (x : T) \in \Gamma}{\Gamma \vdash x : T} \text{ (Var)} \qquad \frac{\Gamma \vdash \quad (s_1, s_2) \in \mathbf{Axiom}}{\Gamma \vdash s_1 : s_2} \text{ (Sort)}$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; [x : T] \vdash U : s_2 \quad (s_1, s_2, s_3) \in \mathbf{Rule}}{\Gamma \vdash \Pi x : T \,.\, U : s_3} \text{ (E-Prd)}$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma; [x : T] \vdash U : s_2 \quad (s_1, s_2, s_3) \in \mathbf{Rule}}{\Gamma \vdash \forall x : T \,.\, U : s_3} \text{ (I-Prd)}$$

$$\frac{\Gamma; [x : T] \vdash M : U \quad \Gamma \vdash \Pi x : T \,.\, U : s}{\Gamma \vdash \lambda x \,.\, M : \Pi x : T \,.\, U} \text{ (Lam)} \qquad \frac{\Gamma \vdash M : \Pi x : T \,.\, U \quad \Gamma \vdash N : T}{\Gamma \vdash M\ N : U\{x := N\}} \text{ (App)}$$

$$\frac{\Gamma; [x : T] \vdash M : U \quad \Gamma \vdash \forall x : T \,.\, U : s \quad x \notin FV(M)}{\Gamma \vdash M : \forall x : T \,.\, U} \text{ (Gen)}$$

$$\frac{\Gamma \vdash M : \forall x : T \,.\, U \quad \Gamma \vdash N : T}{\Gamma \vdash M : U\{x := N\}} \text{ (Inst)}$$

$$\frac{\Gamma \vdash M : s_1 \quad s_1 \leq s_2}{\Gamma \vdash M : s_2} \text{ (Cum)} \qquad \frac{\Gamma \vdash M : T \quad \Gamma \vdash T' : s \quad T =_{\beta\eta} T'}{\Gamma \vdash M : T'} \text{ (Conv)}$$

$$\frac{\Gamma \vdash \lambda x \,.\, (M\ x) : T \quad x \notin FV(M)}{\Gamma \vdash M : T} \text{ (Ext)}$$

$$\frac{\Gamma; [x : T] \vdash M : U \quad x \notin FV(M) \cup FV(U)}{\Gamma \vdash M : U} \text{ (Str)}$$

Fig. 2.3 – Règles de typage du Calcul des Constructions implicite

défini par les règles d'inférence (WF-E) (bonne formation du contexte vide) et (WF-S) (bonne formation d'un contexte dans lequel on a ajouté une déclaration supplémentaire), suivant ainsi les présentations habituelles de ECC et du Calcul des Constructions Inductives [53, 63].

Enfin, le calcul implicite introduit quatre nouvelles règles spécifiques (Inst), (Gen), (Ext) et (Str) qui nécessitent davantage de commentaires.

Règles (Gen) et (Inst) Les règles d'introduction et d'élimination du produit implicite (Gen) (pour "généralisation") et (Inst) (pour "instanciation") sont au produit implicite ce que les règles (Lam) et (App) sont au produit explicite. À la différence de ces dernières, elles ne s'accompagnent pas par l'apparition d'une λ-abstraction ou d'une application au niveau du terme M figurant dans la conclusion ; aussi les appellera-t-on *règles silencieuses*. Formellement, ces deux règles sont analogues aux règles d'introduction et d'élimination du polymorphisme dans les langages de la famille ML (ainsi que dans le système F à la Curry [61]), règles dont nous avons d'ailleurs conservé le nom.

Notons également la présence d'une condition de bord $x \notin FV(M)$ dans la règle (Gen), dont la justification première est d'assurer la préservation de la consistance du scope des variables.[5] D'un point de vue sémantique, cette restriction a en revanche une conséquence importante, puisqu'elle restreint la possibilité de construire un terme de type produit implicite, ce que l'on peut résumer de la manière suivante :

> *Pour généraliser de manière implicite la preuve d'un type $P(x)$ par rapport à x, il faut que la preuve de $P(x)$ ne dépende pas explicitement de la variable x.*

De manière générale, il y aura toujours moins de termes habitant un produit implicite $\forall x : T \,.\, U$ que de termes habitant le produit explicite correspondant $\Pi x : T \,.\, U$ (*cf* paragraphe 2.6.2).

La règle (Ext) L'objet de la règle (Ext) (pour "extensionnalité") est de forcer dans le calcul la propriété de préservation du typage par η-réduction (en conjonction avec la règle (Conv) qui prend également en compte la règle η). Nous verrons au paragraphe 2.5 que l'intérêt de cette règle réside essentiellement dans le fait qu'elle confère de bonnes propriétés à la relation de sous-typage, telles que la contravariance et la covariance respectives des première et seconde composantes des produits dépendants vis-à-vis du sous-typage.[6]

L'idée d'introduire "de force" la règle η afin d'améliorer les propriétés du sous-typage n'est pas nouvelle, et a déjà été proposée dans le cadre du système F à la Curry [61], lequel est par ailleurs un sous-système du Calcul des Constructions implicite.

La règle (Str) La présence de la dernière règle — appelée (Str) pour "renforcement" (*strengthening* en anglais) — peut paraître surprenante, puisque la règle correspondante est admissible dans CCω, ECC, le Calcul des Constructions Inductives et plus généralement dans tous les PTS fonctionnels. En revanche dans le calcul implicite, la propriété de renforcement n'est

[5] En effet, si on décharge l'hypothèse $(x : T)$ même lorsque la variable x apparaît libre dans M, la consistance du scope est perdue, puisque dans la conclusion figure une occurrence libre d'une variable qui n'est plus déclarée dans le contexte. Bien entendu, ce problème ne survient jamais dans le cadre d'une introduction de produit explicite, puisque la décharge d'une hypothèse $(x : T)$ est systématiquement accompagée par l'apparition d'un λ-lieur qui en conserve la trace dans le terme de preuve.

[6] Cette propriété est une des propriétés fondamentales que l'on attend d'un système de types muni d'une relation de sous-typage.

pas admissible sans l'ajout d'une règle spécifique, en raison de la présence du *produit implicite non-dépendant*. Afin d'éclairer ce point, nous allons temporairement admettre certains résultats métathéoriques qui seront établis dans la section suivante.

Signification du produit implicite non-dépendant Considérons un produit implicite non-dépendant $\forall x : T \,.\, U$ (c'est-à-dire tel que $x \notin FV(U)$) que nous supposons bien formé dans un certain contexte Γ. Pour n'importe quel terme M de type U dans Γ, il est possible d'affaiblir le jugement $\Gamma \vdash M : U$ afin d'obtenir $\Gamma; [x : T] \vdash M : U$. En appliquant la règle de généralisation, il vient alors $\Gamma \vdash M : \forall x : T \,.\, U$, ce qui montre que n'importe quel terme de type U est également un terme de type $\forall x : T \,.\, U$. Intuitivement, nous n'avons rien fait d'autre qu'introduire une quantification "inutile" sur une variable x qui n'apparaît pas dans U.

Réciproquement, supposons que M est un terme de type $\forall x : T \,.\, U$ (avec $x \notin FV(U)$) dans le contexte Γ. Si nous connaissons un terme N de type T dans Γ, nous pouvons instancier la variable x par N — à l'aide de la règle (Inst) — et ainsi dériver que M a également le type $U\{x := N\}$, qui est ici identique à U puisque $x \notin FV(U)$. En d'autres termes, nous venons de montrer qu'un produit implicite non-dépendant $\forall x : T \,.\, U$ a les mêmes habitants que U, *pourvu que l'on soit capable de prouver que le type T est habité dans le contexte correspondant.*

L'introduction d'une règle spécifique de renforcement nous permet d'étendre l'équivalence de types

$$M \ \in \ U \quad \Leftrightarrow \quad M \ \in \ \forall x : T \,.\, U \qquad\qquad (\text{si } x \notin FV(U))$$

à tous les produits implicites non-dépendants $\forall x : T \,.\, U$, *y compris lorsque le type T est vide.* Pour cela, on raisonne de la manière suivante : si $\Gamma \vdash M : \forall x : T \,.\, U$, alors la règle admissible d'affaiblissement nous permet de dériver $\Gamma; [p : T] \vdash M : \forall x : T \,.\, U$, où $p : T$ est une hypothèse supplémentaire ajoutée au contexte. Dans ce contexte étendu, il est alors possible d'instancier x par p pour dériver $\Gamma; [p : T] \vdash M : U$. Enfin, la règle de renforcement nous permet de faire disparaître l'hypothèse intermédiaire $p : T$ puisqu'on a manifestement $p \notin FV(M)$ et $p \notin FV(U)$, ce qui nous permet de dériver le jugement souhaité $\Gamma \vdash M : U$.

Autrement dit, la règle de renforcement nous donne la possibilité d'introduire une hypothèse temporaire pour établir un jugement, et d'*effacer* ensuite cette hypothèse dans la mesure où elle n'apparaît plus ni dans le terme ni dans le type de la conclusion, même si cette hypothèse a été effectivement utilisée à une *position implicite* (typiquement lors d'une instanciation) dans l'arbre de dérivation.

Nous verrons dans la seconde partie de cette thèse que la règle de renforcement est de loin la règle la plus difficile à interpréter. Pour cette raison, nous définirons au paragraphe 2.4.3 une restriction du calcul implicite — le *calcul implicite restreint* — qui reprend toutes les règles du calcul implicite à l'exception de la règle de renforcement. Nous aurons l'occasion de voir dans les pages qui viennent que le calcul implicite restreint satisfait essentiellement les mêmes propriétés syntaxiques que le calcul complet, telles que la propriété de préservation du typage par la $\beta\eta$-réduction pour ne citer que cet exemple.

Le calcul implicite restreint jouera un rôle important dans la seconde partie de cette thèse, puisque les deux premiers modèles que nous construirons (dans les chapitres 5 et 6) lui seront consacrés.[7] Ce n'est donc qu'au chapitre 7 consacré à la preuve du théorème de normalisation

[7] Nous verrons en particulier que ces premiers modèles invalident tous les deux la règle de renforcement.

forte que nous modéliserons enfin le calcul complet (c'est-à-dire le calcul muni sa règle de renforcement).[8] Dans la suite de ce chapitre, nous aurons l'occasion de revenir sur ce point lorsque nous traiterons les problèmes de cohérence relative.

2.4 Propriétés du typage

Dans cette section sont exposées les principales propriétés métathéoriques du Calcul des Constructions implicite, dont la plus importante est sans doute la propriété de préservation du typage pour la $\beta\eta$-réduction ($\beta\eta$-*subject reduction* en anglais).

Dans le cadre du calcul implicite, ce résultat est assez difficile à établir, essentiellement à cause de la présence des règles (Str) et (Ext). Pour résoudre le problème syntaxique posé par la règle de renforcement, nous montrerons que la plupart des propriétés syntaxiques du formalisme complet se déduisent des propriétés du formalisme privé de la règle de renforcement (*i.e.* le calcul implicite restreint) à l'aide d'un lemme d'élimination de la règle de renforcement (lemme 2.4.17). En pratique, nous effectuerons donc la plupart des preuves dans le calcul implicite restreint, et il nous suffira d'utiliser cette propriété d'élimination du renforcement pour en déduire chacune des propriétés correspondantes dans le formalisme complet.

Le problème posé par la règle (Ext) est cependant beaucoup plus délicat à résoudre, principalement en raison du fait que le terme figurant dans la prémisse est structurellement plus grand que celui qui figure dans la conclusion. Pour contourner ce problème, nous serons amenés à ne considérer (au paragraphe 2.4.4) que des dérivations d'une forme particulière — les dérivations η-directes — dans lesquelles l'utilisation de la règle (Ext) est interdite à certaines positions particulièrement sensibles dans les dérivations. Cette restriction induira une nouvelle forme de jugement (notée $\Gamma \vdash_d M : T$) plus faible que la forme restreinte (notée $\Gamma \vdash_r M : T$), dont nous verrons qu'elle a cependant de bonnes propriétés de clôture. Nous montrerons dans un premier temps la propriété de préservation du typage par la β-réduction pour les jugements issus de dérivations η-directes, et nous verrons que ce résultat peut être étendu à toutes les dérivations du calcul implicite restreint en utilisant le lemme de report de la règle de η-réduction (lemme 2.2.3). La propriété de préservation du typage par la β-réduction dans le calcul complet (muni de sa règle de renforcement) s'en déduira alors de manière immédiate.

Les différentes étapes de la démonstration sont récapitulées dans la figure 2.4 où nous avons représenté par des flèches les principales dépendances entre les lemmes intermédiaires.

2.4.1 Propriétés élémentaires dans CCI

Les deux lemmes qui suivent ont la même structure, et se prouvent par une récurrence mutuelle sur la structure des dérivations de $\Gamma \vdash$ et $\Gamma \vdash M : T$. Dans les deux cas, il est nécessaire de renforcer l'hypothèse de récurrence $Q(\Gamma, M, T)$ associée au jugement de typage $\Gamma \vdash M : T$ (second item) en posant

$$Q'(\Gamma, M, T) \quad \equiv \quad P(\Gamma) \ \wedge \ Q(\Gamma, M, T),$$

où $P(\Gamma)$ désigne l'hypothèse de récurrence associée au jugement de bonne formation de contexte $\Gamma \vdash$ (premier item).

[8] La preuve de cohérence du calcul implicite que nous allons présenter dans cette thèse est donc une preuve indirecte, basée sur le théorème de normalisation forte. Notons qu'à l'heure actuelle, nous ne disposons toujours pas d'un modèle du calcul implicite complet qui nous permettrait d'obtenir une preuve de cohérence directe.

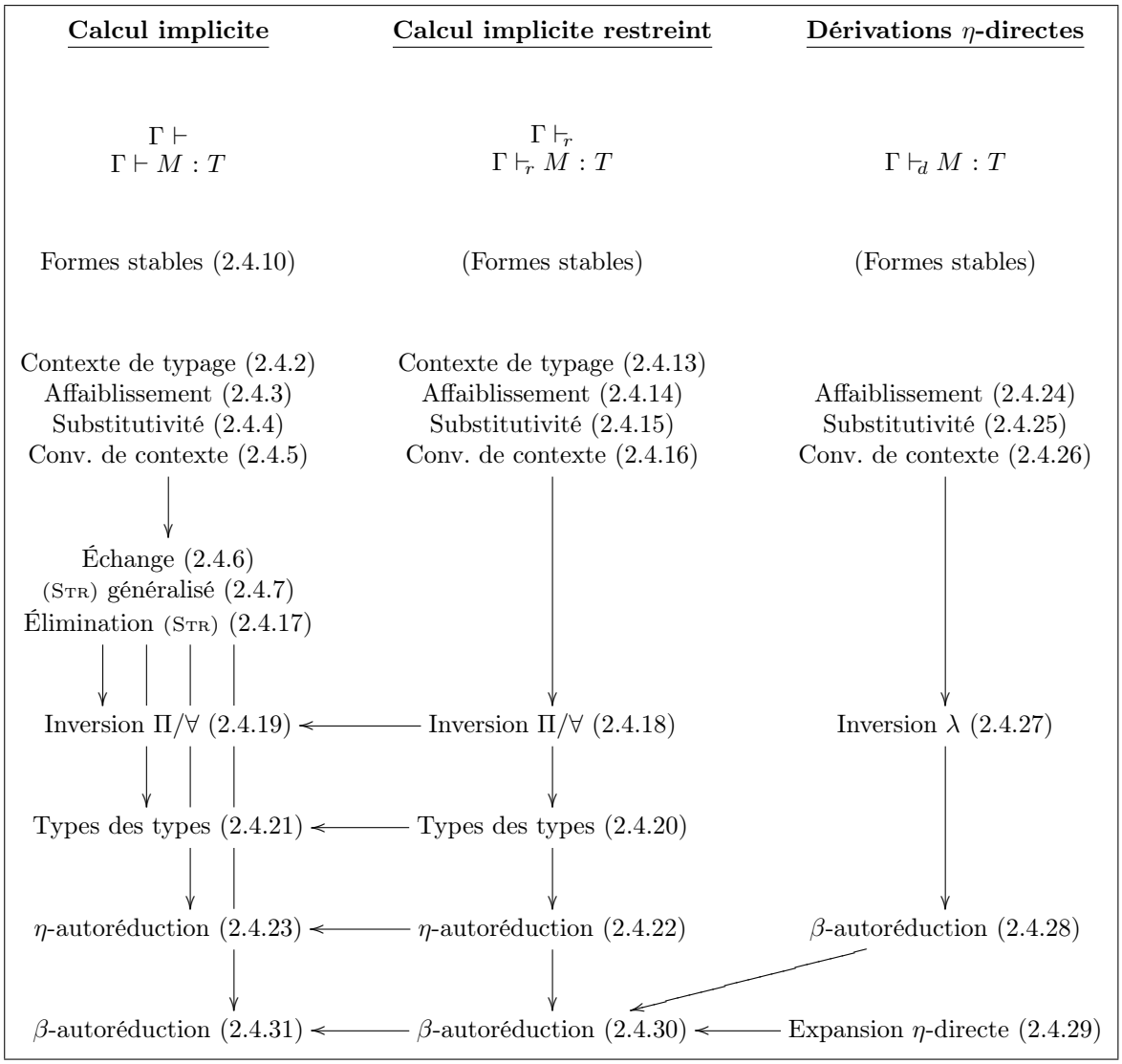


FIG. 2.4 – Architecture de la preuve de $\beta\eta$-autoréduction dans CCI et CCI$^-$

Lemme 2.4.1 (Déclaration des variables) — *Soit* $\Gamma = [x_1 : T_1; \ldots; x_n : T_n]$ *un contexte.*

1. *Si* Γ *est bien formé, alors* $FV(T_i) \subset \{x_1; \ldots; x_{i-1}\}$ *pour tout* $i \in [1..n]$ *;*
2. *Si* $\Gamma \vdash M : T$*, alors* $FV(M) \subset DV(\Gamma)$ *et* $FV(T) \subset DV(\Gamma)$*.*

Lemme 2.4.2 (Bonne formation de contexte) — *Soit* Γ *un contexte.*

1. *Si* Γ *est bien formé, alors tout préfixe de* Γ *est bien formé ;*
2. *Si* $\Gamma \vdash M : T$ *alors le contexte* Γ *est bien formé.*

Lemme 2.4.3 (Affaiblissement) — *Soit* $\Gamma \vdash M : T$ *un jugement valide. Pour tout contexte* Γ' *bien formé tel que* $\Gamma \subset \Gamma'$ *on a* $\Gamma' \vdash M : T$*.*

Preuve. Par récurrence sur la structure de la dérivation de $\Gamma \vdash M : T$. □

Lemme 2.4.4 (Substitutivité) — *Si* $\Gamma_1 \vdash N_0 : T_0$ *est un jugement valide, alors pour tout contexte* Γ_2 *et pour tous termes* M *et* T *:*

1. $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash \quad \Rightarrow \quad \Gamma_1; (\Gamma_2\{x_0 := N_0\}) \vdash$
2. $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash M : T \quad \Rightarrow \quad \Gamma_1; (\Gamma_2\{x_0 := N_0\}) \vdash M\{x_0 := N_0\} : T\{x_0 := N_0\}$

Preuve. Par récurrence mutuelle sur la structure des dérivations de $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash$ et $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash M : T$, en utilisant la même technique que pour les lemmes 2.4.1 et 2.4.2. Le seul point délicat correspond au cas de la règle (Var), qui nécessite l'emploi du lemme d'affaiblissement. □

Lemme 2.4.5 (Conversion de contexte) — *Soient* Γ *et* Γ' *des contextes tels que* $\Gamma =_{\beta\eta} \Gamma'$*. Si* $\Gamma \vdash M : T$ *et si* Γ' *est bien formé, alors* $\Gamma' \vdash M : T$*.*

Preuve. Par récurrence sur la structure de la dérivation de $\Gamma \vdash M : T$, en utilisant les lemmes 2.4.2, 2.4.3 et la règle (Conv) dans le cas de la règle (Var). □

Nous allons terminer ce paragraphe par deux lemmes qui sont des conséquences de la règle de renforcement :

Lemme 2.4.6 (Échange) — *Si* $x_1 \notin FV(T_2)$*, alors :*

1. $\Gamma_1; [x_1 : T_1;\ x_2 : T_2]; \Gamma_2 \vdash \quad \Rightarrow \quad \Gamma_1; [x_2 : T_2;\ x_1 : T_1]; \Gamma_2 \vdash$
2. $\Gamma_1; [x_1 : T_1;\ x_2 : T_2]; \Gamma_2 \vdash M : T \quad \Rightarrow \quad \Gamma_1; [x_2 : T_2;\ x_1 : T_2]; \Gamma_2 \vdash M : T$

Preuve. Les points 1 et 2 se prouvent simultanément par récurrence sur la longueur du contexte Γ_2. Dans le cas de base, la preuve du premier point utilise la règle de renforcement pour montrer successivement que les contextes $\Gamma; [x_2 : T_2]$ et $\Gamma_1; [x_2 : T_2;\ x_1 : T_1]$ sont bien formés. Le second point découle du premier point par une utilisation immédiate du lemme d'affaiblissement, en remarquant que $\Gamma_1; [x_1 : T_1;\ x_2 : T_2]$ est un sous-contexte de $\Gamma_1; [x_2 : T_2;\ x_1 : T_1]$. Dans le cas récursif, on procède par inversion de la règle (WF-S) pour montrer le premier point (à l'aide de l'hypothèse de récurrence), et le passage du premier au second point s'effectue de la même manière que pour le cas de base. □

Lemme 2.4.7 (Renforcement généralisé) — *Si $x_0 \notin FV(\Gamma_2) \cup FV(M) \cup FV(T)$, alors :*

1. $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash \quad \Rightarrow \quad \Gamma_1; \Gamma_2 \vdash$
2. $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash M : T \quad \Rightarrow \quad \Gamma_1; \Gamma_2 \vdash M : T$

Preuve. 1. Supposons le contexte $\Gamma_1; [x_0 : T_0]; \Gamma_2$ bien formé. Par échanges successifs (en faisant commuter l'hypothèse $x_0 : T_0$ avec chacune des déclarations figurant dans Γ_2), il est clair que le contexte $\Gamma_1; \Gamma_2; [x_0 : T_0]$ est bien formé, d'où il ressort que le contexte $\Gamma_1; \Gamma_2$ est lui-même bien formé d'après le lemme 2.4.2.

2. Supposons $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash M : T$. D'après le lemme 2.4.2 et le premier point que nous venons d'établir, le contexte $\Gamma_1; \Gamma_2; [x_0 : T_0]$ est bien formé. Par affaiblissement (lemme 2.4.3) on a $\Gamma_1; \Gamma_2; [x_0 : T_0] \vdash M : T$, d'où l'on tire $\Gamma_1; \Gamma_2 \vdash M : T$ par application de la règle de renforcement. □

2.4.2 Formes stables

Pour tout contexte $\Delta = [x_1 : T_1; \ldots; x_n : T_n]$ et pour tout terme U on note

$$\forall \Delta \,.\, U \quad = \quad \forall x_1 : T_1 \,.\, \ldots \forall x_n : T_n \,.\, U.$$

On dit d'un terme M qu'il est :

- une *forme sorte* s'il existe un contexte Δ et une sorte s tel que

$$M \quad =_{\beta\eta} \quad \forall \Delta \,.\, s$$

- une *forme produit* s'il existe un contexte Δ, une variable x et des termes T, U tels que

$$M \quad =_{\beta\eta} \quad \forall \Delta \,.\, \Pi x : T \,.\, U$$

- une *forme stable* si M est une forme sorte ou bien une forme produit.

La terminologie de *forme stable* est justifiée par le fait que la *forme* (sorte ou produit) du type T intervenant dans un jugement $\Gamma \vdash M : T$ est préservée par les *règles de sous-typage* du calcul implicite, c'est-à-dire les règles (Cum), (Conv), (Gen), (Inst), (Ext) et (Str). Cette caractéristique importante des formes stables résulte du lemme suivant :

Lemme 2.4.8 (Préservation des formes stables)

1. *(Conversion) Si U est une forme sorte (resp. une forme produit) et si $U' =_{\beta\eta} U$, alors U' est une forme sorte (resp. une forme produit).*
2. *(Substitutivité) Si U est une forme sorte (resp. une forme produit), alors pour tout terme N et toute variable x, $U\{x := N\}$ est une forme sorte (resp. une forme produit).*
3. *(Généralisation) Si T est une forme sorte (resp. une forme produit), alors $\forall x : T \,.\, U$ est une forme sorte (resp. une forme produit).*
4. *(Renforcement) Si $\forall x : T \,.\, U$ est une forme sorte (resp. une forme produit), alors U est une forme sorte (resp. une forme produit).*

Par ailleurs, le fait d'être une forme sorte est incompatible avec le fait d'être une forme produit, et réciproquement :

Lemme 2.4.9 (Incompatibilité des formes sorte et produit) — *Si U_1 est une forme sorte et si U_2 est une forme produit, alors les termes U_1 et U_2 ne peuvent être ni égaux ni même $\beta\eta$-convertibles.*

Ce dernier résultat nous permet d'énoncer le *lemme des formes stables* :

Lemme 2.4.10 (Formes stables) — *Pour tout jugement dérivable $\Gamma \vdash M : T$*

1. *Si M est une sorte ou un produit explicite/implicite, alors T est une forme sorte.*
2. *Si M est une λ-abstraction, alors T est une forme produit.*

Preuve. Par récurrence sur la structure de la dérivation du jugement $\Gamma \vdash M : T$, on montre conjointement les trois assertions suivantes :

1. Le terme M ne contient aucun sous-terme de la forme $(M_1\ M_2)$ dont le membre gauche M_1 est une sorte, un produit explicite ou un produit implicite.
2. Si M est une sorte ou un produit explicite/implicite, alors T est une forme sorte.
3. Si M est une λ-abstraction, alors T est une forme produit.

On distingue les cas suivant la dernière règle appliquée. Les cas (Var), (Sort), (E-Prd), (I-Prd), (Lam) et (Cum) sont triviaux, tandis que les cas (Gen), (Inst), (Str) et (Conv) découlent du lemme de préservation des formes stables. Les deux seuls cas intéressants concernent donc les règles (App) et (Ext).

- (App). La conclusion, qui est de la forme $\Gamma \vdash (M\ N) : U\{x := N\}$, découle de deux prémisses $\Gamma \vdash M : \Pi x : T\,.\,U$ et $\Gamma \vdash N : T$.

  1. Supposons que $(M\ N)$ admette un sous-terme applicatif $(M_1\ M_2)$ où M_1 est une sorte ou un produit (explicite ou implicite). D'après l'hypothèse de récurrence, ce sous-terme $(M_1\ M_2)$ ne peut être ni un sous-terme de M (première prémisse) ni un sous-terme de N (seconde prémisse), d'où $M_1 = M$ et $M_2 = N$. Par conséquent, M est une sorte ou un produit, d'où il résulte (d'après l'hypothèse de récurrence sur la première prémisse) que $\Pi x : T\,.\,U$ est une forme sorte, ce qui est absurde.
  2. et 3. Immédiat.

- (Ext). La conclusion $\Gamma \vdash M : T$ provient d'une prémisse $\Gamma \vdash \lambda x\,.\,(M\ x) : T$.

  1. Immédiat, car M est un sous-terme de son η-expansé $\lambda x\,.\,(M\ x)$.
  2. Si M est une sorte ou un produit, alors $(M\ x)$ est un sous-terme de $\lambda x\,.\,(M\ x)$ dont le membre gauche est une sorte ou un produit, ce qui contredit l'hypothèse de récurrence. Ce cas est donc absurde.
  3. Immédiat d'après l'hypothèse de récurrence.

Lemme 2.4.11 — *Si π est une dérivation d'un jugement $\Gamma \vdash M : T$ dans lequel M est une sorte ou un produit, alors la dernière règle appliquée dans π ne peut pas être la règle* (Ext).

Preuve. Ce résultat est une conséquence immédiate du lemme précédent. □

2.4.3 Le calcul implicite restreint CCI^-

Définition 2.4.12 (Calcul implicite restreint) — On appelle *calcul implicite restreint* et on note CCI^- le sous-système du calcul implicite basé sur la même syntaxe et les mêmes règles de réduction, et dont le système de types est engendré par les règles de typage du calcul implicite privées de la règle de renforcement (Str).

Dans ce qui, suit on notera respectivement $\Gamma \vdash_r$ et $\Gamma \vdash_r M : T$ les jugements de bonne formation de contexte et de typage associés au calcul implicite restreint. Ainsi, un jugement $\Gamma \vdash_r$ ou $\Gamma \vdash_r M : T$ est dérivable si et seulement si dans le calcul implicite, le jugement correspondant ($\Gamma \vdash$ ou $\Gamma \vdash M : T$) admet une dérivation dans laquelle ne figure aucune instance de la règle de renforcement.

Le calcul implicite restreint vérifie essentiellement les mêmes propriétés que le calcul implicite, mises à part les propriétés dont la démonstration est directement liée à la présence de la règle de renforcement, telles que la propriété d'échange (2.4.6) ou la propriété de renforcement généralisé (2.4.7). Les quatres lemmes suivants se démontrent de la même manière que les lemmes 2.4.2, 2.4.3, 2.4.4 et 2.4.5.

Lemme 2.4.13 (Bonne formation de contexte dans CCI^-) — *Soit Γ un contexte.*

1. *Si $\Gamma \vdash_r$, alors pour tout préfixe $\Gamma' \sqsubset \Gamma$ on a $\Gamma' \vdash_r$.*
2. *Si $\Gamma \vdash_r M : T$, alors $\Gamma \vdash_r$.*

Lemme 2.4.14 (Affaiblissement dans CCI^-) — *Soit $\Gamma \vdash_r M : T$ un jugement valide dans le calcul implicite restreint. Pour tout contexte Γ' tel que $\Gamma' \vdash_r$ et $\Gamma \subset \Gamma'$ on a $\Gamma' \vdash_r M : T$.*

Lemme 2.4.15 (Substitutivité dans CCI^-) — *Si $\Gamma_1 \vdash_r N_0 : T_0$ est un jugement valide, alors pour tout contexte Γ_2 et pour tous termes M et T :*

1. $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash_r \quad \Rightarrow \quad \Gamma_1; (\Gamma_2\{x_0 := N_0\}) \vdash_r$
2. $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash_r M : T \quad \Rightarrow \quad \Gamma_1; (\Gamma_2\{x_0 := N_0\}) \vdash_r M\{x_0 := N_0\} : T\{x_0 := N_0\}$

Lemme 2.4.16 (Conversion de contexte dans CCI^-) — *Soient Γ et Γ' des contextes tels que $\Gamma =_{\beta\eta} \Gamma'$. Si $\Gamma \vdash_r M : T$ et si $\Gamma' \vdash_r$, alors $\Gamma' \vdash_r M : T$.*

Nous allons maintenant établir une propriété essentielle du calcul implicite qui va nous permettre en pratique de déduire la plupart des propriétés du calcul implicite (complet) des propriétés correspondantes dans le calcul implicite restreint. Cette propriété d'*élimination de la règle de renforcement* exprime que tous les jugements du calcul implicite peuvent être dérivés dans le calcul implicite restreint — c'est-à-dire sans jamais utiliser la règle de renforcement — à condition d'ajouter au contexte une hypothèse suffisament forte (qui, notons-le, rend ce contexte incohérent).

Proposition 2.4.17 (Élimination de la règle de renforcement) — *Pour tout jugement $\Gamma \vdash M : T$ dérivable dans* CCI*, il existe une sorte s tel que le jugement*

$$[c : \forall a : s\,.\,a]; \Gamma \vdash_r M : T \qquad (\textit{où } c \textit{ est une variable fraîche})$$

soit dérivable dans le calcul implicite restreint CCI^-.

Preuve. Soient $\Gamma \vdash M : T$ un jugement valide de CCI et π une dérivation de ce jugement. Considérons une sorte s suffisamment haut placée dans la hiérarchie d'univers de manière à avoir $s' \leq s$ pour toute sorte s' figurant dans la dérivation π. Le choix d'une telle sorte s est possible parce que

1. la dérivation π n'utilise qu'un sous-ensemble fini de l'ensemble des sortes $\mathcal{S}$;
2. l'ensemble des sortes $\mathcal{S}$ muni de l'ordre de cumulativité $\leq$ est un ensemble *dirigé*, puisque pour toute paire de sortes $\{s_1; s_2\}$, il existe une sorte s_3 telle que $s_1 \leq s_3$ et $s_2 \leq s_3$.

Soit c une variable fraîche, n'apparaissant pas dans π. On montre alors par une récurrence structurelle sur π que tout sous-arbre $\pi' \subset \pi$ satisfait les conditions suivantes :

1. si la racine de π' est étiquetée par un jugement de la forme $\Gamma' \vdash$, alors le jugement $[c : \forall a : s \,.\, a]; \Gamma' \vdash_r$ est dérivable dans CCI^- ;
2. si la racine de π' est étiquetée par un jugement de la forme $\Gamma' \vdash M' : T'$, alors le jugement $[c : \forall a : s \,.\, a]; \Gamma' \vdash_r M' : T'$ est dérivable dans CCI^-.

Le seul cas intéressant est celui où le sous-arbre $\pi' \subset \pi$ se termine par une utilisation de la règle (Str). Dans ce cas, π' est de la forme

$$\pi' \;=\; \left\{ \begin{array}{c} \vdots\ \pi_1 \\ \dfrac{\Gamma'; [x : T'] \vdash M' : U'}{\Gamma' \vdash M' : U'}\ (\textsc{Str}) \end{array} \right.$$

avec $x \notin FV(M')$ et $x \notin FV(U')$. D'après l'hypothèse de récurrence appliquée à π_1 on a

$$[c : \forall a : s \,.\, a]; \Gamma'; [x : T'] \vdash_r M' : U'.$$

D'après la forme des règles de typage dans le calcul implicite, il est clair que π_1 contient un sous-arbre π_2 qui est une dérivation de $\Gamma' \vdash T' : s'$ pour une certaine sorte s', laquelle vérifie $s' \leq s$ en vertu de l'hypothèse initiale sur s. Comme π_2 est un sous-arbre strict de π', l'hypothèse de récurrence s'applique également à π_2, d'où il ressort que

$$[c : \forall a : s \,.\, a]; \Gamma' \vdash_r T' : s',$$

ce qui nous permet de construire la dérivation

$$\dfrac{\dfrac{}{[c : \forall a : s \,.\, a]; \Gamma' \vdash_r c : \forall a : s \,.\, a}\ (\textsc{Var}) \quad \dfrac{[c : \forall a : s \,.\, a]; \Gamma' \vdash_r T' : s' \quad s' \leq s}{[c : \forall a : s \,.\, a]; \Gamma' \vdash_r T' : s}\ (\textsc{Cum})}{[c : \forall a : s \,.\, a]; \Gamma' \vdash_r c : T'}\ (\textsc{Inst}\ a := T')$$

qui est une dérivation du jugement $[c : \forall a : s \,.\, a]; \Gamma' \vdash_r c : T'$ dans CCI^-. D'après le lemme de substitutivité 2.4.15, on peut remplacer la variable $x : T'$ dans le jugement

$$[x : \forall a : s \,.\, a]; \Gamma'; [x : T'] \vdash_r M' : U'$$

par le terme $c : T'$, d'où il ressort que le jugement $[c : \forall a : s \,.\, a]; \Gamma' \vdash_r M' : U'$ est valide dans le calcul implicite restreint. □

Lemme 2.4.18 (Inversion des produits dans CCI^-) — *Si $\Gamma \vdash_r Bx : T \,.\, U : R$ (où B désigne Π ou $\forall$), alors il existe un contexte Δ et quatre sortes s_1, s_2, s_3 et s telles que*

1. $R =_{\beta\eta} \forall \Delta \,.\, s$;
2. $\Gamma; \Delta \vdash_r T : s_1$;
3. $\Gamma; \Delta; [x : T] \vdash_r U : s_2$;
4. $(s_1, s_2, s_3) \in \mathbf{Rule}$;
5. $s_3 \leq s$.

Preuve. Par récurrence sur la dérivation du jugement $\Gamma \vdash_r Bx : T \,.\, U : R$. Une telle dérivation ne peut se terminer que par l'utilisation d'une des règles (E-Prd) (si $B \equiv \Pi$), (I-Prd) (si $B \equiv \forall$), (Inst), (Gen), (Conv) ou (Cum). Notons que le cas de la règle (Ext) est ici éliminé à l'aide du lemme 2.4.11.

- (E-Prd), (I-Prd) Ce cas de base est immédiat : les sortes s_1, s_2 et s_3 sont données par les prémisses et la conclusion, et il suffit de prendre $\Delta \equiv []$ et $s \equiv s_3$.
- (Gen) La conclusion, qui est de la forme $\Gamma \vdash_r Bx : T \,.\, U : \forall a : A \,.\, R$, provient de deux prémisses $\Gamma; [a : A] \vdash_r Bx : T \,.\, U : R$ et $\Gamma \vdash_r \forall a : A \,.\, R : s'$, avec la condition de bord $a \notin FV(Bx : T \,.\, U)$. D'après l'hypothèse de récurrence appliquée à la première prémisse, il existe Δ, s_1, s_2, s_3 et s tels que

  $$R =_{\beta\eta} \forall \Delta \,.\, s\,; \qquad (s_1, s_2, s_3) \in \mathbf{Rule}\,; \qquad s_3 \leq s\,;$$
  $$\Gamma; [a : A]; \Delta \vdash_r T : s_1\,; \qquad \Gamma; [a : A]; \Delta; [x : T] \vdash_r U : s_2\,.$$

  La conclusion est immédiate en posant $\Delta' = [a : A]; \Delta$.
- (Inst) La conclusion, qui est de la forme $\Gamma \vdash_r Bx : T \,.\, U : R\{a := N\}$, provient de deux prémisses $\Gamma \vdash_r Bx : T \,.\, U : \forall a : A \,.\, R$ et $\Gamma \vdash_r N : A$. D'après l'hypothèse de récurrence appliquée à la première prémisse, il existe Δ, s_1, s_2, s_3 et s tels que

  $$\forall a : A \,.\, R =_{\beta\eta} \forall \Delta \,.\, s\,; \qquad (s_1, s_2, s_3) \in \mathbf{Rule}\,; \qquad s_3 \leq s\,;$$
  $$\Gamma; \Delta \vdash_r T : s_1\,; \qquad \Gamma; \Delta; [x : T] \vdash_r U : s_2\,.$$

  D'après la relation $\forall a : A \,.\, R =_{\beta\eta} \forall \Delta \,.\, s$, il existe un terme A' et un contexte Δ' tels que $\Delta = [a : A']; \Delta'$, $A =_{\beta\eta} A'$ et $R =_{\beta\eta} \forall \Delta' \,.\, s$. Avec ces notations, les deux jugements déduits de l'hypothèse de récurrence s'écrivent

  $$\Gamma; [a : A']; \Delta' \vdash_r T : s_1 \qquad \text{et} \qquad \Gamma; [a : A']; \Delta'; [x : T] \vdash_r U : s_2.$$

  D'après le premier de ces deux jugements, A' est un type bien formé dans Γ, ce qui nous permet d'appliquer la règle de conversion pour dériver $\Gamma \vdash_r N : A'$. En substituant N à a dans les jugements ci-dessus (lemme 2.4.15), il vient

  $$\Gamma; \Delta'\{a := N\} \vdash_r T : s_1 \qquad \text{et} \qquad \Gamma; \Delta'\{a := N\}; [x : T] \vdash_r U : s_2$$

  (la substitution étant inopérante dans T et U puisque $a \notin FV(T)$ et $a \notin FV(U)$). On pose alors $\Delta'' = \Delta'\{a := N\}$, et on conclut en remarquant que $R\{a := N\} =_{\beta\eta} \forall \Delta'' \,.\, s$.
- (Conv) Ce cas résulte immédiatement de l'hypothèse de récurrence appliquée à la première prémisse.
- (Cum) Ce cas résulte immédiatement de l'hypothèse de récurrence. Il est toutefois nécessaire de remplacer la sorte s par la sorte $s' \geq s$. Bien entendu, on a toujours $s_3 \leq s'$ puisque la relation de cumulativité est transitive. □

On remarquera que dans le lemme ci-dessus, le processus d'inversion fait apparaître un contexte Δ dans les jugements

$$\Gamma; \Delta \vdash_r T : s_1 \qquad \text{et} \qquad \Gamma; \Delta; [x : T] \vdash_r U : s_2,$$

alors qu'aucune des hypothèses introduites par Δ n'est utilisée dans les termes T et U. Bien entendu, ces hypothèses ne sont pas effaçables en l'absence de la règle de renforcement et, dans le calcul implicite complet, la présence de la règle (Str) permet naturellement de simplifier l'énoncé du lemme d'inversion des produits de la manière suivante :

Lemme 2.4.19 (Inversion des produits dans CCI**)** — *Si* $\Gamma \vdash Bx{:}T\,.\,U : R$ *(où* B *désigne* Π *ou* $\forall$*), alors il existe un contexte* Δ *et quatre sortes* s_1*,* s_2*,* s_3 *et* s *telles que*

1. $R =_{\beta\eta} \forall \Delta\,.\,s$;
2. $\Gamma \vdash T : s_1$;
3. $\Gamma; [x : T] \vdash U : s_2$;
4. $(s_1, s_2, s_3) \in \mathbf{Rule}$;
5. $s_3 \leq s$.

Preuve. Supposons $\Gamma \vdash Bx{:}T\,.\,U : R$. Par élimination de la règle de renforcement (lemme 2.4.17), le jugement $[c : \forall a : s_0\,.\,a]; \Gamma \vdash_r Bx{:}T\,.\,U : R$ est dérivable dans le calcul implicite restreint quitte à ajouter en début de contexte une hypothèse de la forme $c : \forall a : s_0\,.\,a$ (où s_0 est une sorte suffisamment élevée dans la hiérarchie d'univers). D'après le lemme précédent, il existe donc un contexte Δ et des sortes s_1, s_2, s_3, s tels que

$$\begin{array}{lll} R =_{\beta\eta} \forall \Delta\,.\,s\,; & (s_1, s_2, s_3) \in \mathbf{Rule}\,; & s_3 \leq s\,; \\ [c : \forall a : s_0\,.\,a]; \Gamma; \Delta \vdash_r T : s_1\,; & [c : \forall a : s_0\,.\,a]; \Gamma; \Delta; [x : T] \vdash_r U : s_2\,. & \end{array}$$

Bien entendu, les deux jugements de typage ci-dessus sont également dérivables dans le calcul implicite complet, où il est maintenant possible d'appliquer la règle de renforcement généralisée (lemme 2.4.7) sur l'hypothèse $c : \forall a : s_0\,.\,a$ ainsi que sur toutes les hypothèses figurant dans le contexte Δ, d'où il ressort que $\Gamma \vdash T : s_1$ et $\Gamma; [x : T] \vdash U : s_2$. □

Lemme 2.4.20 (Type des types dans CCI$^-$**)** — *Si* $\Gamma \vdash_r M : T$*, alors il existe une sorte* s *telle que* $\Gamma \vdash_r T : s$.

Preuve. Par récurrence sur la structure de la dérivation de $\Gamma \vdash_r M : T$, en distinguant les cas suivant la dernière règle appliquée.

- (Var) La conclusion $\Gamma \vdash_r x : T$ provient d'une prémisse $\Gamma \vdash_r$. Comme $(x : T) \in \Gamma$, il existe des contextes Γ_1 et Γ_2 tels que $\Gamma = \Gamma_1; [x : T]; \Gamma_2$. D'après le lemme 2.4.13, le contexte $\Gamma_1; [x : T]$ est bien formé dans CCI$^-$, d'où il existe par inversion de la règle (WF-S) une sorte s telle que $\Gamma_1 \vdash_r T : s$. On conclut alors à l'aide du lemme d'affaiblissement.
- (Sort), (E-Prd), (I-Prd), (Cum) Ces cas sont immédiats, car le type T est une sorte, et toutes les sortes du calcul implicite sont typables par une autre sorte dans n'importe quel contexte bien formé.
- (Lam), (Gen), (Conv). Le jugement recherché est donné par la seconde prémisse.

- (App) La conclusion de la dérivation $\Gamma \vdash_r M\ N : U\{x := N\}$ provient de deux prémisses $\Gamma \vdash_r M : \Pi x : T\,.\,U$ et $\Gamma \vdash_r N : T$. D'après l'hypothèse de récurrence appliquée à la première prémisse, il existe une sorte s telle que $\Gamma \vdash_r \Pi x : T\,.\,U : s$. D'après le lemme d'inversion des produits dans CCI$^-$ (lemme 2.4.18), il existe des sortes s_1, s_2 et s_3 (le contexte Δ donné par ce lemme étant forcément vide) telles que $\Gamma \vdash_r T : s_1$, $\Gamma; [x : T] \vdash_r U : s_2$ et $(s_1, s_2, s_3) \in \mathbf{Rule}$. En appliquant le lemme de substitutivité aux jugements $\Gamma; [x : T] \vdash_r U : s_2$ et $\Gamma \vdash_r N : T$, il vient $\Gamma \vdash_r U\{x := N\} : s_2$.
- (Inst) Ce cas se traite de la même façon que pour la règle (App). □

Lemme 2.4.21 (Type des types dans CCI**)** — *Si $\Gamma \vdash M : T$, alors il existe une sorte s telle que $\Gamma \vdash T : s$.*

Preuve. Ce lemme découle immédiatement du lemme précédent, en utilisant le lemme d'élimination de la règle de renforcement pour se ramener à CCI$^-$, puis la règle de renforcement généralisée pour revenir dans CCI. □

Proposition 2.4.22 (Préservation du typage par la η-réduction dans CCI$^-$**)** — *Si le jugement $\Gamma \vdash_r M : T$ est dérivable dans* CCI$^-$ *et si $M \to_\eta M'$, alors $\Gamma \vdash_r M' : T$ est dérivable également.*

Preuve. Le cas de base correspondant à une étape de η-réduction à la racine est traité à l'aide de la règle (Ext). Le cas général est ensuite prouvé par récurrence sur la dérivation de $\Gamma \vdash_r M : T$. Le seul cas délicat est le cas de la règle (App), en raison de la substitution effectuée sur le type. Dans ce cas, la conclusion $\Gamma \vdash_r M\ N : U\{x := N\}$ provient de deux prémisses $\Gamma \vdash_r M : \Pi x : T\,.\,U$ et $\Gamma \vdash_r N : T$. Considérons un terme R tel que $M\ N \to_\eta R$ (une étape de η-réduction). On distingue les deux cas suivants :

- Soit $R = M'\ N$, et $M \to_\eta M'$. D'après l'hypothèse de récurrence appliquée à la première prémisse, on a $\Gamma \vdash_r M' : \Pi x : T\,.\,U$, d'où l'on tire $\Gamma \vdash_r M'\ N : U\{x := N\}$ à l'aide de la règle (App).
- Soit $R = M\ N'$, et $N \to_\eta N'$. D'après l'hypothèse de récurrence appliquée à la seconde prémisse, on a $\Gamma \vdash_r N' : T$, d'où l'on tire $\Gamma \vdash_r M\ N' : U\{x := N'\}$ à l'aide de la règle (App). Comme par hypothèse, $\Gamma \vdash_r M\ N : U\{x := N\}$, il existe d'après le lemme 2.4.20 une sorte s telle que $\Gamma \vdash_r U\{x := N\} : s$. Comme les types $U\{x := N\}$ et $U\{x := N'\}$ sont η-convertibles, on dérive le jugement recherché $\Gamma \vdash_r M\ N' : U\{x := N\}$ à l'aide de la règle (Conv).

Notons que le problème de conversion rencontré ci-dessus (lorsque $N \to_\eta N'$) ne peut pas survenir dans le cas de la règle (Inst), puisque l'argument implicite $N : T$ n'est pas conservé dans le terme figurant en conclusion. □

Proposition 2.4.23 (Préservation du typage par la η-réduction dans CCI**)** — *Si le jugement $\Gamma \vdash M : T$ est dérivable dans* CCI *et si $M \to_\eta M'$, alors $\Gamma \vdash M' : T$ est dérivable également.*

Preuve. Ce résultat découle de la proposition précédente en se ramenant à CCI$^-$ par élimination de la règle de renforcement. □

2.4.4 Dérivations η-directes

Intuitivement, l'arbre de dérivation d'un jugement $\Gamma \vdash_r M : T$ (nous ne considérerons ici que des jugements du calcul implicite restreint) peut être décomposé en deux morceaux. Le premier morceau part de la racine (la conclusion) et correspond à la déstructuration progressive du terme M, à l'exception des η-expansions éventuelles dues à l'utilisation de la règle (Ext). Lorsque sont atteints les nœuds étiquetés par les règles (Var) et (Sort) — une fois que le terme M a été complètement déstructuré — commence la seconde partie de l'arbre de dérivation, qui correspond à la déstructuration du contexte éventuellement enrichi par les passages sous les lieurs.

Dans ce qui suit, nous dirons qu'une dérivation d'un jugement $\Gamma \vdash_r M : T$ (dans CCI^-) est *η-directe* si elle ne contient aucune instance de la règle (Ext) dans la première partie de l'arbre de dérivation (celle qui déstructure le terme M). En revanche, une dérivation η-directe peut faire intervenir la règle (Ext), dans la mesure où celle-ci n'est utilisée que dans la seconde partie de la dérivation (celle qui déstructure le contexte).

Plus formellement, une dérivation du jugement $\Gamma \vdash_r M : T$ est *η-directe* si l'une des conditions suivantes est satisfaite :

- la dernière règle appliquée est la règle (Var) ou (Sort) ;
- la dernière règle appliquée est la règle (E-Prd) ou (I-Prd), et les dérivations des deux prémisses sont η-directes ;
- la dernière règle est (Lam), et la dérivation de la prémisse $\Gamma; [x : T] \vdash M : U$ est η-directe ;
- la dernière règle est (App), et les dérivations des deux prémisses sont η-directes ;
- la dernière règle est (Gen) et la dérivation de la prémisse $\Gamma; [x : T] \vdash M : U$ est η-directe ;
- la dernière règle est (Inst) et la dérivation des deux prémisses sont η-directes ;
- la dernière règle est (Conv) et la dérivation de la prémisse $\Gamma \vdash M : T$ est η-directe ;
- la dernière règle est (Cum) et la dérivation de la prémisse $\Gamma \vdash T : s_1$ est η-directe ;

(Il n'y a pas à considérer ici le cas de la règle (Str), puisque nous nous plaçons dans CCI^-.)

Lorsqu'un jugement $\Gamma \vdash_r M : T$ admet une dérivation η-directe, on le note $\Gamma \vdash_d M : T$. De manière équivalente, la notation $\Gamma \vdash_d M : T$ peut être définie comme une nouvelle forme de jugement engendrée par les règles d'inférence de la figure 2.5 (à la page 78), dans laquelle nous avons distingué les instances du jugement η-direct $\Gamma \vdash_d M : T$ de celles du jugement de typage standard $\Gamma \vdash_r M : T$ dans CCI^- en les encadrant. Remarquons que dans les dérivations η-directes, la règle (Ext) est autorisée

- dans toutes les dérivations des jugements de bonne formation de contexte ;
- dans la dérivation de la seconde prémisse de la règle (Lam) (pour le typage du produit explicite $\Pi x : T \,.\, U$) ;
- dans la dérivation de la seconde prémisse de la règle (Gen) (pour le typage du produit implicite $\forall x : T \,.\, U$) ;
- dans la dérivation de la seconde prémisse de la règle (Conv) (pour le typage du type T').

Notons également que la règle $(\text{Inst})_d$ effectue une légère entorse au schéma informel que nous avons exposé plus haut, puisqu'on impose que la dérivation de typage de l'argument implicite $\Gamma \vdash N : T$ soit également η-directe, bien que cette partie de la dérivation ne concerne pas la déstructuration du terme M. (Nous en verrons la raison lors du traitement du cas de la règle (Inst) dans la démonstration du lemme 2.4.27.)

Le nouveau jugement $\Gamma \vdash_d M : T$ a de bonnes propriétés de clôture. En particulier, les lemmes d'affaiblissement, de substitutivité et de conversion de contexte valent toujours lors-

$$\frac{\Gamma \vdash_r \quad (x : T) \in \Gamma}{\boxed{\Gamma \vdash_d x : T}}\ (\textsc{Var'}) \qquad \frac{\Gamma \vdash_r \quad (s_1, s_2) \in \mathbf{Axiom}}{\boxed{\Gamma \vdash_d s_1 : s_2}}\ (\textsc{Sort'})$$

$$\frac{\boxed{\Gamma \vdash_d T : s_1} \quad \boxed{\Gamma; [x : T] \vdash_d U : s_2} \quad (s_1, s_2, s_3) \in \mathbf{Rule}}{\boxed{\Gamma \vdash_d \Pi x : T \,.\, U : s_3}}\ (\textsc{E-Prd'})$$

$$\frac{\boxed{\Gamma \vdash_d T : s_1} \quad \boxed{\Gamma; [x : T] \vdash_d U : s_2} \quad (s_1, s_2, s_3) \in \mathbf{Rule}}{\boxed{\Gamma \vdash_d \forall x : T \,.\, U : s_3}}\ (\textsc{I-Prd'})$$

$$\frac{\boxed{\Gamma; [x : T] \vdash_d M : U} \quad \Gamma \vdash_r \Pi x : T \,.\, U : s}{\boxed{\Gamma \vdash_d \lambda x \,.\, M : \Pi x : T \,.\, U}}\ (\textsc{Lam'})$$

$$\frac{\boxed{\Gamma \vdash_d M : \Pi x : T \,.\, U} \quad \boxed{\Gamma \vdash_d N : T}}{\boxed{\Gamma \vdash_d M\ N : U\{x := N\}}}\ (\textsc{App'})$$

$$\frac{\boxed{\Gamma; [x : T] \vdash_d M : U} \quad \Gamma \vdash_r \forall x : T \,.\, U : s \quad x \notin FV(M)}{\boxed{\Gamma \vdash_d M : \forall x : T \,.\, U}}\ (\textsc{Gen'})$$

$$\frac{\boxed{\Gamma \vdash_d M : \forall x : T \,.\, U} \quad \boxed{\Gamma \vdash_d N : T}}{\boxed{\Gamma \vdash_d M : U\{x := N\}}}\ (\textsc{Inst'})$$

$$\frac{\boxed{\Gamma \vdash_d M : s_1} \quad s_1 \leq s_2}{\boxed{\Gamma \vdash_d M : s_2}}\ (\textsc{Cum'}) \qquad \frac{\boxed{\Gamma \vdash_d M : T} \quad \Gamma \vdash_r T' : s \quad T =_{\beta\eta} T'}{\boxed{\Gamma \vdash_d M : T'}}\ (\textsc{Conv'})$$

(Pas de règle $(\textsc{Ext})_d$)

FIG. 2.5 – Règles d'inférence définissant le jugement $\Gamma \vdash_d M : T$

qu'on remplace $\Gamma \vdash_r T : U$ par $\Gamma \vdash_d T : U$. Bien entendu, la propriété de préservation du typage par la η-réduction est automatiquement perdue lorsqu'on se restreint uniquement aux dérivations η-directes.

Les lemmes suivants se prouvent de la même manière que leurs homologues 2.4.3/2.4.14, 2.4.4/2.4.15 et 2.4.5/2.4.16, lesquels ont été établis dans les paragraphes 2.4.1 et 2.4.3.

Lemme 2.4.24 (Affaiblissement η-direct) — *Soit $\Gamma \vdash_d M : T$ un jugement η-direct. Pour tout contexte Γ' bien formé tel que $\Gamma \subset \Gamma'$ on a $\Gamma' \vdash_d M : T$.*

Lemme 2.4.25 (Substitutivité η-directe) — *Si $\Gamma_1 \vdash_d N_0 : T_0$, alors pour tout contexte Γ_2 et pour tous termes M et N on a :*

$$\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash_d M : T \quad \Rightarrow \quad \Gamma_1; (\Gamma_2\{x_0 := N_0\}) \vdash_d M\{x_0 := N_0\} : T\{x_0 := N_0\}$$

Lemme 2.4.26 (Conversion de contexte η-directe) — *Soient Γ et Γ' des contextes tels que $\Gamma =_{\beta\eta} \Gamma'$. Si $\Gamma \vdash_d M : T$ et si Γ' est bien formé dans* CCI$^-$*, alors $\Gamma' \vdash_d M : T$.*

Ces premiers lemmes étant établis, nous pouvons à présent énoncer le lemme d'inversion de la λ-abstraction.

Lemme 2.4.27 (Inversion η-directe de la λ-abstraction) — *Si $\Gamma \vdash_d \lambda x \,.\, M : R$, alors il existe un contexte Δ et deux termes T, U tels que :*

1. $R =_{\beta\eta} \forall \Delta \,.\, \Pi x : T \,.\, U$;
2. $\Gamma; \Delta; [x : T] \vdash_d M : U$.

Preuve. Par récurrence sur la dérivation η-directe du jugement $\Gamma \vdash_d \lambda x \,.\, M : R$. La dernière règle appliquée peut être l'une des règles (Lam), (Gen), (Inst) ou (Conv), le cas embarrassant de la règle (Ext) étant supprimé par l'hypothèse selon laquelle la dérivation considérée est η-directe.

- (Lam) Ce cas — qui est le cas de base — est immédiat.
- (Gen) La conclusion, qui est de la forme $\Gamma \vdash_d \lambda x \,.\, M : \forall a : A \,.\, R$, provient de deux prémisses $\Gamma; [a : A] \vdash_d \lambda x \,.\, M : R$ (avec $a \notin FV(M)$) et $\Gamma \vdash_r \forall a : A \,.\, R : s$. D'après l'hypothèse de récurrence appliquée à la première prémisse, il existe un contexte Δ et des termes T et U tels que $R =_{\beta\eta} \forall \Delta \,.\, \Pi x : T \,.\, U$ et $\Gamma; [a : A]; \Delta; [x : T] \vdash_d M : U$. On conclut alors en posant $\Delta' = [a : A]; \Delta$, et en remarquant que $\forall a : A \,.\, R =_{\beta\eta} \forall \Delta' \,.\, \Pi x : T \,.\, U$.
- (Inst) La conclusion, qui est de la forme $\Gamma \vdash_d \lambda x \,.\, M : R\{a := N\}$, provient de deux prémisses $\Gamma \vdash_d \lambda x \,.\, M : \forall a : A \,.\, R$ et $\Gamma \vdash_d N : A$. D'après l'hypothèse de récurrence appliquée à la première prémisse, il existe un contexte Δ et des termes T et U tels que $\Gamma; \Delta; [x : T] \vdash_d M : U$ et $\forall a : A \,.\, R =_{\beta\eta} \forall \Delta \,.\, \Pi x : T \,.\, U$. De cette dernière égalité découle l'existence d'un terme A' et d'un contexte Δ' tels que $A' =_{\beta\eta} A$ et $\Delta \equiv [a : A']; \Delta'$. Comme $\Gamma; [a : A']; \Delta'; [x : T] \vdash_d M : U$, le contexte $\Gamma; [a : A']; \Delta'$ est bien formé dans CCI$^-$, et il existe une sorte s telle que $\Gamma \vdash_r A' : s$. En appliquant la règle (Conv') au jugement $\Gamma \vdash_d N : A$ il vient $\Gamma \vdash_d N : A'$, ce qui nous permet ensuite d'appliquer le lemme de substitutivité η-directe (2.4.25) au jugement $\Gamma; [a : A']; \Delta'; [x : T] \vdash_d M : U$ (en remplaçant a par N) pour dériver le jugement

$$\Gamma; (\Delta'\{a := N\}); [x : T\{a := N\}] \vdash_d M : U\{a := N\}$$

(la substitution étant inopérante sur M puisque $a \notin FV(M)$). On conclut alors en posant $\Delta'' = \Delta'\{a := N\}$, $T'' = T\{a := N\}$, $U'' = U\{a := N\}$, et en remarquant que $R\{a := N\} =_{\beta\eta} \forall \Delta'' \,.\, \Pi x : T'' \,.\, U''$.

- (Conv) Le résultat recherché découle directement de l'hypothèse de récurrence appliquée à la première prémisse. □

Lemme 2.4.28 (Préservation du typage η-direct par β-réduction) — *Si* $\Gamma \vdash_d M : T$ *et si* $M \to_\beta M'$, *alors* $\Gamma \vdash_d M' : T$.

Preuve. Par récurrence sur la dérivation de $\Gamma \vdash_d M : T$. Le seul cas intéressant est celui de la règle (App). Dans ce cas, la conclusion $\Gamma \vdash_d M\ N : U\{x := N\}$ provient de deux prémisses $\Gamma \vdash_d M : \Pi x : T \,.\, U$ et $\Gamma \vdash_d N : T$. Soit R un terme tel que $M\ N \to_\beta R$. Trois cas sont possibles :

1. $R = M'\ N$ et $M \to_\beta M'$. Dans ce cas, l'hypothèse de récurrence appliquée à la première prémisse donne $\Gamma \vdash_d M' : \Pi x : T \,.\, U$. On conclut en appliquant la règle (App), ce qui nous donne le jugement recherché $\Gamma \vdash_d M'\ N : U\{x := N\}$.
2. $R = M\ N'$ et $N \to_\beta N'$. Dans ce cas, l'hypothèse de récurrence appliquée à la seconde prémisse donne $\Gamma \vdash_d N' : T$. En appliquant la règle (App), on dérive alors le jugement $\Gamma \vdash_d M\ N' : U\{x := N'\}$. Comme par ailleurs $\Gamma \vdash_d M\ N : U\{x := N\}$, il existe une sorte s (d'après le lemme 2.4.20) telle que $\Gamma \vdash_r U\{x := N\} : s$. En appliquant la règle de conversion η-directe au jugement $\Gamma \vdash_d M\ N' : U\{x := N'\}$, on obtient alors $\Gamma \vdash_d M\ N' : U\{x := N\}$.
3. $M = \lambda x \,.\, M_0$ et $R = M_0\{x := N\}$ (cas de la β-réduction à la racine). D'après le lemme d'inversion η-directe de l'abstraction appliqué au jugement $\Gamma \vdash_d \lambda x \,.\, M_0 : \Pi x : T \,.\, U$ (première prémisse), il existe des termes T' et U' (le contexte Δ donné par ce lemme étant alors vide) tels que $\Pi x : T \,.\, U =_{\beta\eta} \Pi x : T' \,.\, U'$ et $\Gamma; [x : T'] \vdash_d M_0 : U'$. D'après la confluence de la $\beta\eta$-réduction, la relation de conversion $\Pi x : T \,.\, U =_{\beta\eta} \Pi x : T' \,.\, U'$ entraîne que $T =_{\beta\eta} T'$ et $U =_{\beta\eta} U'$. Comme par ailleurs, $\Gamma \vdash_d N : T$ (seconde prémisse), il existe d'après le lemme 2.4.20 une sorte s telle que $\Gamma \vdash_r T : s$, ce qui prouve que le contexte $\Gamma; [x : T]$ est bien formé dans CCI^-. En appliquant le lemme de conversion de contexte η-directe (2.4.26), il vient alors $\Gamma; [x : T] \vdash_d M_0 : U'$, ce qui nous permet d'utiliser le lemme 2.4.25 pour montrer que $\Gamma \vdash_d M_0\{x := N\} : U'\{x := N\}$. On conclut alors que $\Gamma \vdash_d M_0\{x := N\} : U\{x := N\}$ en appliquant la règle de conversion, dont l'utilisation est justifiée par le lemme 2.4.20 appliqué à l'hypothèse initiale $\Gamma \vdash_d M\ N : U\{x := N\}$. □

2.4.5 Préservation du typage par la $\beta\eta$-réduction

Dans les paragraphes qui précèdent, nous avons étudié les propriétés de trois formes de jugements de typage, qui sont :

- les jugements de typage respectifs $\Gamma \vdash M : T$ et $\Gamma \vdash_r M : T$ du calcul implicite complet et du calcul implicite restreint, dont nous avons montré qu'ils satisfont tous les deux la propriété de préservation du typage par la η-réduction (propositions 2.4.23 et 2.4.22) ;
- le jugement de typage η-direct $\Gamma \vdash_d M : T$, dont nous avons montré qu'il satisfait la propriété de préservation du typage par la β-réduction. (En revanche, cette forme de jugement n'est pas préservée par η-réduction.)

À l'aide de ces différents résultats, il est maintenant possible d'établir la propriété de préservation du typage par la β-réduction dans CCI$^-$ comme dans CCI. Pour cela, nous devons d'abord montrer le résultat suivant :

Lemme 2.4.29 (Expansion η-directe) — *Si $\Gamma \vdash_r M : T$, alors il existe un terme M_0 tel que $M_0 \twoheadrightarrow_\eta M$ et $\Gamma \vdash_d M_0 : T$.*

Preuve. Par récurrence sur la dérivation de $\Gamma \vdash_r M : T$. L'abandon de chacune des instances de la règle (Ext) dans la partie de la dérivation déstructurant le terme M se traduit par une étape de η-expansion dans le terme M_0. Pour traiter le cas des règles (App) et (Inst), on utilise la règle de conversion η-directe (Conv') afin d'empêcher que les η-expansions effectuées dans le terme ne se répercutent dans son type par le biais de la substitution. (Ce qui est possible dans la mesure où la règle (Conv') n'impose pas que la dérivation de sa seconde prémisse $\Gamma \vdash_r T' : s$ soit η-directe.) □

Remarquons que dans le lemme ci-dessus, les η-expansions sont effectuées dans le terme M, mais pas dans le type T ni dans le contexte Γ. Ceci est dû au fait que dans les dérivations η-directes, la règle (Ext) n'est interdite que dans le fragment de la dérivation correspondant à la destructuration effective du terme M, tout en étant autorisée partout ailleurs (et en particulier dans la dérivation de la seconde prémisse de la règle de conversion).

Proposition 2.4.30 (Préservation du typage par la β-réduction dans CCI$^-$) — *Si le jugement $\Gamma \vdash_r M : T$ est dérivable (dans CCI$^-$) et si $M \twoheadrightarrow_\beta M'$, alors $\Gamma \vdash_r M' : T$.*

Preuve. Supposons $\Gamma \vdash_r M : T$ et $M \twoheadrightarrow_\beta M'$. D'après le lemme d'expansion η-directe, il existe un terme M_0 tel que $\Gamma \vdash_d M_0 : T$ et $M_0 \twoheadrightarrow_\eta M$. Comme $M_0 \twoheadrightarrow_{\beta\eta} M'$, il existe en vertu du lemme de report de la règle η un terme M_1 tel que $M_0 \twoheadrightarrow_\beta M_1$ et $M_1 \twoheadrightarrow_\eta M'$. D'après la propriété de préservation du typage η-direct par la β-réduction, on a $\Gamma \vdash_d M_1 : T$ d'où $\Gamma \vdash_r M_1 : T$. D'après la propriété de préservation du typage par la η-réduction dans CCI$^-$, on a par conséquent $\Gamma \vdash_r M' : T$. □

Proposition 2.4.31 (Préservation du typage par la β-réduction dans CCI) — *Si le jugement $\Gamma \vdash M : T$ est dérivable (dans CCI) et si $M \twoheadrightarrow_\beta M'$, alors $\Gamma \vdash M' : T$.*

Preuve. Cette proposition découle de la proposition précédente à l'aide du lemme d'élimination de la règle de renforcement et de la règle de renforcement généralisée. □

Proposition 2.4.32 (Préservation du typage par la $\beta\eta$-réduction) — *Si $\Gamma \vdash M : T$ (resp. $\Gamma \vdash_r M : T$) et $M \twoheadrightarrow_{\beta\eta} M'$, alors $\Gamma \vdash M' : T$ (resp. $\Gamma \vdash_r M' : T$).*

Preuve. Découle des propositions 2.4.31, 2.4.30, 2.4.23 et 2.4.22. □

2.5 Sous-typage

2.5.1 La relation de sous-typage

Un des aspects les plus intéressants du Calcul des Constructions implicite est la riche relation de sous-typage induite par le produit implicite. Cette relation notée $\Gamma \vdash T \leq T'$ peut être définie directement à partir de la relation de typage comme la "macro" suivante :

$$\Gamma \vdash T \leq T' \quad \equiv \quad \Gamma; [x : T] \vdash x : T' \qquad (x \text{ variable fraîche})$$

Remarquons que cette définition exprime à l'intérieur du formalisme lui-même la signification exacte qu'on souhaite donner à la relation de sous-typage, c'est-à-dire : T est un sous-type de T' dans Γ si tout terme de type T (représenté dans sa généralité par une variable $x : T$ ajoutée au contexte) est également un terme de type T'.

Lemme 2.5.1 (Préordre de sous-typage) — *Les règles d'inférence suivantes sont admissibles :*

$$\frac{\Gamma \vdash T : s}{\Gamma \vdash T \leq T}\ (\textsc{SubRefl}) \qquad \frac{\Gamma \vdash T_1 \leq T_2 \quad \Gamma \vdash T_2 \leq T_3}{\Gamma \vdash T_1 \leq T_3}\ (\textsc{SubTrans})$$

$$\frac{\Gamma \vdash M : T \quad \Gamma \vdash T \leq T'}{\Gamma \vdash M : T'}\ (\textsc{Sub})$$

Preuve. Les fragments de dérivation établissant l'admissibilité de ces trois règles sont donnés par la figure 2.6 dans laquelle on a distingué l'usage des règles admissibles de substitutivité (Subst) et d'affaiblissement (Weak) en les représentant avec un double trait d'inférence. □

Une propriété fondamentale de la relation de sous-typage est que la formation des produits explicites et implicites agit de manière contravariante vis-à-vis des domaines et de manière covariante vis-à-vis des codomaines :

Lemme 2.5.2 (Sous-typage dans les produits) — *Les règles d'inférence suivantes sont admissibles :*

$$\frac{\Gamma \vdash T' \leq T \quad \Gamma; [x : T'] \vdash U \leq U'}{\Gamma \vdash \Pi x : T \,.\, U \leq \Pi x : T' \,.\, U'}\ (\textsc{SubEPrd})$$

$$\frac{\Gamma \vdash T' \leq T \quad \Gamma; [x : T'] \vdash U \leq U'}{\Gamma \vdash \forall x : T \,.\, U \leq \forall x : T' \,.\, U'}\ (\textsc{SubIPrd})$$

Preuve. Les fragments de dérivation établissant l'admissibilité de ces deux règles sont donnés par la figure 2.7. □

Dans la dérivation de la règle (SubEPrd), la règle (Ext) joue un rôle essentiel puisqu'elle permet d'accéder à la fonctionnalité de la variable f de type $\Pi x : T \,.\, U$ grâce à la η-expansion $f \leftarrow_\eta \lambda x \,.\, (f\ x)$. Sans une telle η-expansion, il aurait été impossible de déstructurer la variable f afin de pouvoir travailler séparément sur les composantes T' et U' du produit explicite $\Pi x : T' \,.\, U'$. C'est d'ailleurs pour obtenir cette propriété que la règle (Ext) a été incorporée au système de types du calcul implicite.

Remarquons également que la structure de la dérivation de la règle (SubIPrd) est très similaire à celle de la règle (SubEPrd). En particulier, les règles (Gen) et (Inst) jouent dans la dérivation de (SubIPrd) le même rôle que les règles (Lam) et (App) dans la dérivation de (SubEPrd). En revanche, ces deux règles ne font pas apparaître de η-radical, ce qui explique l'absence de la règle (Ext) dans la dérivation de (SubIPrd).

Signalons enfin que les règles de cumulativité, de conversion, de généralisation et d'instanciation peuvent être reformulées comme des règles particulières de sous-typage :

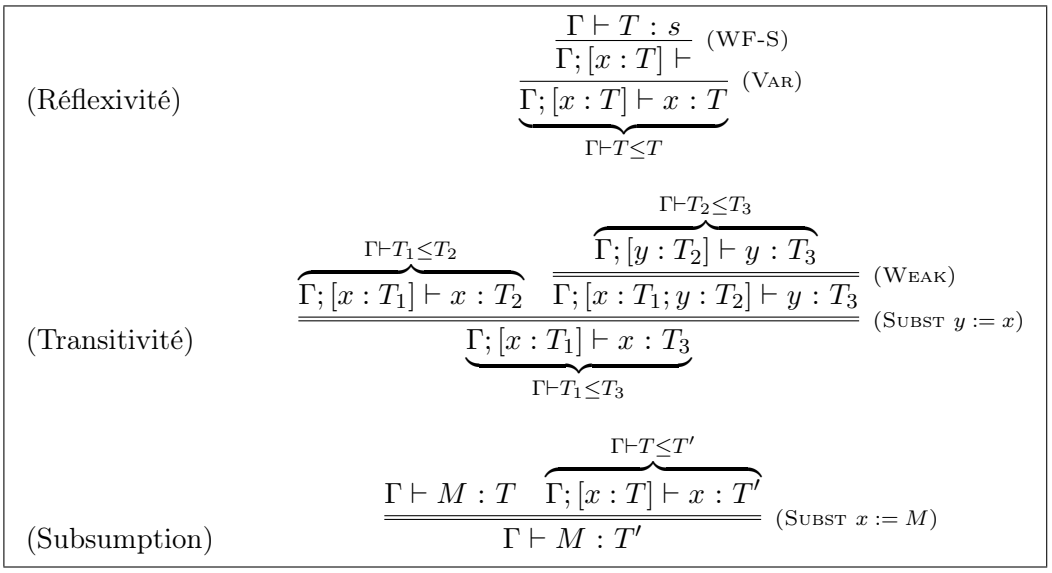


Fig. 2.6 – Admissibilité des règles de réflexivité, transitivité et subsumption

Lemme 2.5.3 — *Les règles d'inférence suivantes sont admissibles :*

$$\frac{\Gamma \vdash T : s \quad \Gamma \vdash T' : s' \quad T =_{\beta\eta} T'}{\Gamma \vdash T \leq T'} \text{ (SubConv)} \qquad \frac{\Gamma \vdash \qquad s_1 \leq s_2}{\Gamma \vdash s_1 \leq s_2} \text{ (SubCum)}$$

$$\frac{\Gamma; [x : T] \vdash U \leq V \quad x \notin FV(U)}{\Gamma \vdash U \leq \forall x : T \,.\, V} \text{ (SubGen)} \qquad \frac{\Gamma \vdash \forall x : T \,.\, U : s \quad \Gamma \vdash N : T}{\Gamma \vdash \forall x : T \,.\, U \leq U\{x := N\}} \text{ (SubInst)}$$

Preuve. L' admissibilité des règles (SubConv), (SubCum) et (SubInst) est immédiate. Pour établir la règle (SubGen), il est d'abord nécessaire d'utiliser la règle de renforcement (Str) pour montrer que U est un type bien formé dans le contexte Γ, ce qui permet ensuite de faire commuter (par affaiblissement) les deux hypothèses $x : T$ et $y : U$ dans le contexte de la prémisse $\Gamma; [x : T;\ y : U] \vdash y : V$ ($\equiv\ \Gamma; [x : T] \vdash U \leq V$) avant d'appliquer la règle (Gen).[9] □

2.5.2 Équivalence de types

La notion de sous-typage du Calcul des Constructions implicite induit naturellement une notion d'*équivalence de type* notée $\Gamma \vdash T \sim T'$, qui est simplement la clôture symétrique de la relation de sous-typage (le contexte Γ étant fixé) :

$$\Gamma \vdash T \sim T' \qquad \equiv \qquad \Gamma \vdash T \leq T' \quad \text{et} \quad \Gamma \vdash T' \leq T.$$

Lemme 2.5.4 (Commutation des produits) — *Soient Γ un contexte, T_1 et T_2 des types bien formés dans le contexte Γ, et U un type bien formé dans le contexte $\Gamma; [x_1 : T_1;\ x_2 : T_2]$. On a les équivalences de types suivantes :*

$$\Gamma \ \vdash\ \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U \ \sim\ \forall x_2 : T_2 \,.\, \forall x_1 : T_1 \,.\, U \qquad \textit{(commutation implicite/implicite)}$$

$$\Gamma \ \vdash\ \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U \ \sim\ \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U \qquad \textit{(commutation explicite/implicite)}$$

[9] Dans la preuve d'admissibilité de la règle (SubGen) il est possible de se passer de la règle de renforcement en remplaçant la condition de bord $x \notin FV(U)$ par une nouvelle prémisse $\Gamma \vdash U : s$.

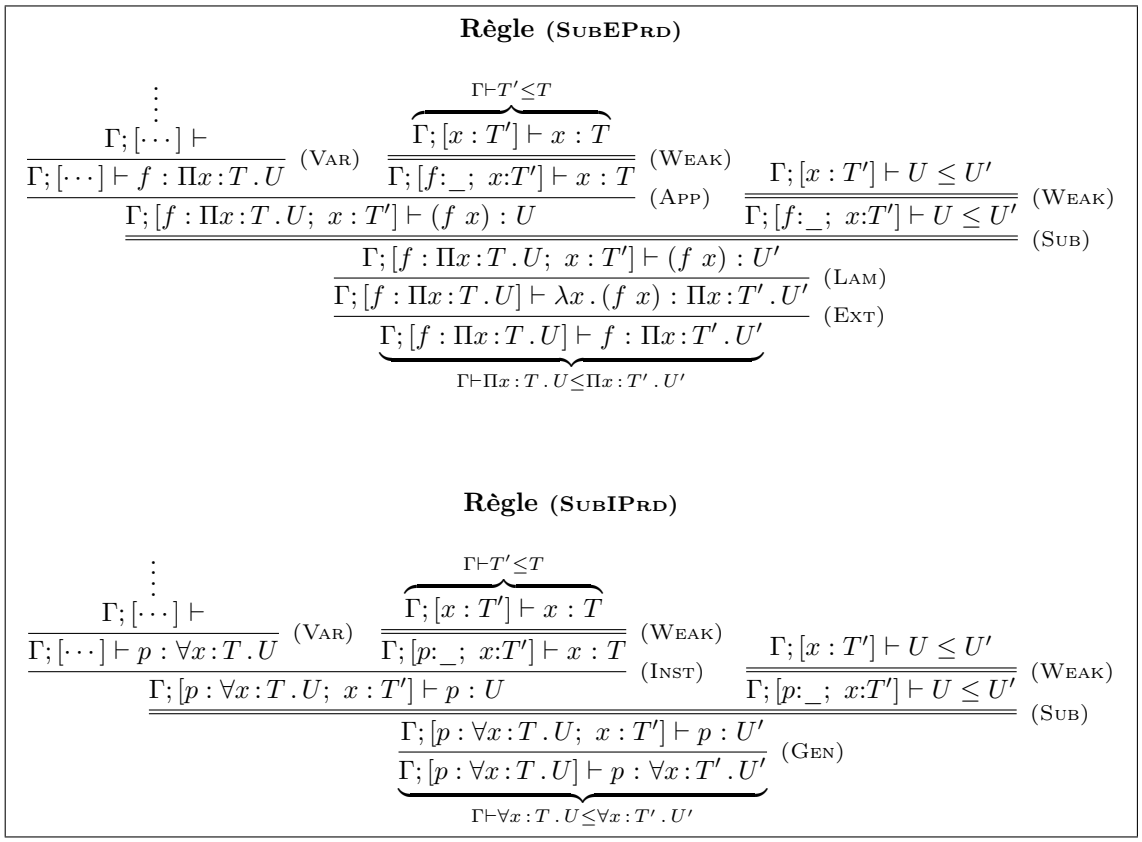


FIG. 2.7 – Admissibilité des règles de sous-typage dans les produits

Preuve. Voir figure 2.8 pour les fragments de dérivation correspondants. □

Pour terminer cette étude du sous-typage, mentionons le fait (déjà annoncé lors de la discussion du paragraphe 2.3.5 au sujet du produit implicite non-dépendant) que la règle de renforcement (Str) permet de montrer l'équivalence de types suivante :

Lemme 2.5.5 (Produit implicite non dépendant) — *Si* $\forall x : T \,.\, U$ *est un produit implicite non-dépendant bien formé dans* Γ*, alors* U *est un type bien formé dans* Γ *et*

$$\Gamma \quad \vdash \quad \forall x : T \,.\, U \quad \sim \quad U.$$

Preuve. Voir figure 2.9 pour les fragments de dérivation correspondants. □

2.6 Résultats de cohérence relative

2.6.1 Cohérence relative du calcul implicite

Dans le calcul implicite, la proposition "fausse" peut être représentée de deux manières différentes : soit sous sa forme explicite $\Pi A : \mathsf{Prop} \,.\, A$ (le "faux" explicite), soit sous sa forme implicite $\forall A : \mathsf{Prop} \,.\, A$ (le "faux" implicite). Comme dans le Calcul des Constructions, une preuve

Dérivation du jugement $\Gamma \vdash \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U \;\leq\; \forall x_2 : T_2 \,.\, \forall x_1 : T_1 \,.\, U$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash p : \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U}\,(\textsc{Var}) \quad \dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash x_1 : T_1}\,(\textsc{Var})}{\Gamma ; [\cdots] \vdash p : \forall x_2 : T_2 \,.\, U}\,(\textsc{Inst}) \quad \dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash x_2 : T_2}\,(\textsc{Var})}{\Gamma ; [p : \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U ;\; x_2 : T_2 ;\; x_1 : T_1] \vdash p : U}\,(\textsc{Inst})}{\Gamma ; [p : \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U ;\; x_2 : T_2] \vdash p : \forall x_1 : T_1 \,.\, U}\,(\textsc{Gen})}{\underbrace{\Gamma ; [p : \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U] \vdash p : \forall x_2 : T_2 \,.\, \forall x_1 : T_1 \,.\, U}_{\Gamma \vdash \forall x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U \leq \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U}}\,(\textsc{Gen})$$

Dérivation du jugement $\Gamma \vdash \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U \;\leq\; \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash p : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U}\,(\textsc{Var}) \quad \dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash x_1 : T_1}\,(\textsc{Var})}{\Gamma ; [\cdots] \vdash (p\ x_1) : \forall x_2 : T_2 \,.\, U}\,(\textsc{App}) \quad \dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash x_2 : T_2}\,(\textsc{Var})}{\Gamma ; [p : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U ;\; x2 : T_2 ;\; x_1 : T_1] \vdash (p\ x_1) : U}\,(\textsc{Inst})}{\Gamma ; [p : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U ;\; x_2 : T_2] \vdash \lambda x_1 \,.\, (p\ x_1) : \Pi x_1 : T_1 \,.\, U}\,(\textsc{Lam})}{\Gamma ; [p : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U ;\; x_2 : T_2] \vdash p : \Pi x_1 : T_1 \,.\, U}\,(\textsc{Ext})}{\underbrace{\Gamma ; [p : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U] \vdash p : \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U}_{\Gamma \vdash \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U \leq \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U}}\,(\textsc{Gen})$$

Dérivation du jugement $\Gamma \vdash \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U \;\leq\; \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash p : \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U}\,(\textsc{Var}) \quad \dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash x_2 : T_2}\,(\textsc{Var})}{\Gamma ; [\cdots] \vdash p : \Pi x_1 : T_1 \,.\, U}\,(\textsc{Inst}) \quad \dfrac{\dfrac{\vdots}{\Gamma ; [\cdots] \vdash}}{\Gamma ; [\cdots] \vdash x_1 : T_1}\,(\textsc{Var})}{\Gamma ; [p : \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U ;\; x_1 : T_1 ;\; x_2 : T_2] \vdash (p\ x_1) : U}\,(\textsc{App})}{\Gamma ; [p : \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U ;\; x_1 : T_1] \vdash (p\ x_1) : \forall x_2 : T_2 \,.\, U}\,(\textsc{Gen})}{\Gamma ; [p : \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U] \vdash \lambda x_1 \,.\, (p\ x_1) : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U}\,(\textsc{Lam})}{\underbrace{\Gamma ; [p : \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U] \vdash p : \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U}_{\Gamma \vdash \forall x_2 : T_2 \,.\, \Pi x_1 : T_1 \,.\, U \leq \Pi x_1 : T_1 \,.\, \forall x_2 : T_2 \,.\, U}}\,(\textsc{Ext})$$

FIG. 2.8 – Règles de commutation implicite/implicite et explicite/implicite

Dérivation du jugement $\Gamma \vdash U \leq \forall x : T \,.\, U$ **(avec** $x \notin FV(U)$**)**

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \forall x : T \,.\, U : s}{\Gamma ; [x : T] \vdash U : s_2}\ (\textsc{InvEPrd})^*}{\Gamma \vdash U : s_2}\ (\textsc{Str})}{\vdots}$$

$$\underbrace{\dfrac{\dfrac{\Gamma ; [p : U ;\ x : T] \vdash}{\Gamma ; [p : U ;\ x : T] \vdash p : U}\ (\textsc{Var})}{\Gamma ; [p : U] \vdash p : \forall x : T \,.\, U}\ (\textsc{Gen})}_{\Gamma \vdash U \leq \forall x : T \,.\, U}$$

* (InvEPrd) $\equiv$ lemme d'inversion du produit explicite (2.4.19)

Dérivation du jugement $\Gamma \vdash \forall x : T \,.\, U \leq U$ **(avec** $x \notin FV(U)$**)**

$$\dfrac{\dfrac{\vdots}{\Gamma ; [p : \forall x : T \,.\, U ;\ x : T] \vdash p : \forall x : T \,.\, U}\ (\textsc{Var}) \qquad \dfrac{\vdots}{\Gamma ; [p : \forall x : T \,.\, U ;\ x : T] \vdash x : T}\ (\textsc{Var})}{\Gamma ; [p : \forall x : T \,.\, U ;\ x : T] \vdash p : U}\ (\textsc{Inst})$$

$$\underbrace{\dfrac{\Gamma ; [p : \forall x : T \,.\, U ;\ x : T] \vdash p : U}{\Gamma ; [p : \forall x : T \,.\, U] \vdash p : U}\ (\textsc{Str})}_{\Gamma \vdash \forall x : T \,.\, U \leq U}$$

FIG. 2.9 – Règle d'équivalence de types du produit implicite non-dépendant

du faux explicite $\Pi A : \mathsf{Prop} \,.\, A$ est une fonction prenant une proposition en argument et retournant une preuve de cette proposition. Au contraire, une preuve du faux implicite $\forall A : \mathsf{Prop} \,.\, A$ est une preuve de n'importe quel proposition, puisque le faux implicite est l'intersection de tous les types propositionnels. Cependant, ces deux représentations de la proposition fausse sont prouvablement équivalentes :

$$\begin{aligned} \lambda f \,.\, f\ (\forall A : \mathsf{Prop} \,.\, A) \quad &: \quad (\Pi A : \mathsf{Prop} \,.\, A) \to (\forall A : \mathsf{Prop} \,.\, A) \\ \lambda p, A \,.\, p \quad &: \quad (\forall A : \mathsf{Prop} \,.\, A) \to (\Pi A : \mathsf{Prop} \,.\, A) \end{aligned}$$

Remarquons que la seconde preuve permet de prouver plus généralement l'implication

$$\lambda p, x \,.\, p \quad : \quad (\forall x : T \,.\, U) \to (\Pi x : T \,.\, U),$$

qui exprime qu'un produit explicite $\Pi x : T \,.\, U$ a au moins autant d'habitants que le produit implicite $\forall x : T \,.\, U$ correspondant. Nous verrons au paragraphe suivant qu'un des intérêts majeurs de la règle de renforcement est d'autoriser la construction d'un terme de preuve réfutant (dans le formalisme lui-même) la réciproque de l'implication ci-dessus.

La propriété de *cohérence relative* du calcul implicite — à savoir que sa cohérence logique peut se déduire de la propriété de normalisation forte — est une conséquence des deux lemmes suivants :

Lemme 2.6.1 — *Dans le contexte vide, le type d'un terme en forme normale de tête faible est une forme stable.*

Preuve. Soit $[] \vdash M : T$ un jugement valide dans le contexte vide. Si M est en forme normale de tête faible, alors M est

- soit de la forme $\lambda x\,.\,M'$. Dans ce cas, le type T est une forme produit (lemme 2.4.10) donc une forme stable.
- soit de la forme $(H\ N_1 \dots N_n)$, où H est une variable, une sorte, un produit explicite ou un produit implicite. Le cas où x est une variable est exclu puisque le contexte de typage est vide. Dans le cas où H est une sorte ou un produit, H ne peut pas se trouver en position fonctionnelle dans une application (*cf* preuve du lemme 2.4.10), d'où $n = 0$ et $M = H$. D'après le lemme des formes stables, le type de M est une forme sorte, donc une forme stable. □

Lemme 2.6.2 (Preuves du faux implicite) — *Dans le contexte vide, le faux implicite* $\forall A : \mathsf{Prop}\,.\,A$ *n'a pas de preuve en forme normale de tête faible.*

Preuve. Ce résultat est une conséquence immédiate du lemme précédent puisque le faux implicite $\forall A : \mathsf{Prop}\,.\,A$ n'est ni une forme sorte, ni une forme produit. □

Insistons sur le fait que le lemme ci-dessus repose sur le lemme des formes stables (2.4.10), qui est vrai dans le calcul implicite complet, muni de sa règle de renforcement.

Proposition 2.6.3 (Cohérence relative) — *Si tous les termes bien typés du Calcul des Constructions implicite sont fortement normalisables, alors le système logique sous-jacent est cohérent.*

Preuve. Ce résultat découle de la propriété de préservation du typage par la β-réduction et du lemme précédent. □

Remarquons que l'hypothèse de normalisation forte n'est pas strictement nécessaire pour établir le lemme précédent. On peut la remplacer par une hypothèse plus faible telle que « tous les termes bien typés de CCI ont une forme normale » (hypothèse de normalisation faible), ou même par « tous les termes bien typés de CCI ont une forme normale de tête faible ».

2.6.2 Cohérence logique et règle de renforcement

La cohérence logique du Calcul des Constructions implicite peut paraître suspecte en raison de la présence de la règle de renforcement (Str) qui autorise l'introduction de *paralogismes temporaires* dans les dérivations. Cette règle permet en effet d'utiliser puis d'effacer du contexte une hypothèse quelconque — et éventuellement fausse ! — du moment que la variable de preuve correspondante n'apparaît ni dans le terme ni dans le type du jugement considéré.

Aussi surprenant que cela puisse paraître, la règle de renforcement ne met pas en danger la cohérence du calcul implicite. En effet, nous montrerons au chapitre 7 que tous les termes bien typés du calcul implicite sont fortement normalisables, d'où il découlera que le calcul implicite est cohérent, en vertu de la proposition 2.6.3.

Une tentative d'attaque de la règle (Str) Afin de se convaincre de l'inocuité de la règle de renforcement, il est intéressant d'étudier l'exemple suivant, qui est une tentative (avortée) de construction de paradoxe basée sur l'utilisation de cette règle.

Considérons le produit implicite non-dépendant $\forall x : \mathsf{False} \,.\, \mathsf{False}$, où False désigne le "faux" implicite $\forall A : \mathsf{Prop} \,.\, A$. D'après le lemme 2.5.5 établi grâce à la règle de renforcement, ce type a les mêmes habitants que le type False lui-même, et cela dans n'importe quel contexte bien formé. Par conséquent, l'implication

$$(\forall x : \mathsf{False} \,.\, \mathsf{False}) \to \mathsf{False},$$

est prouvable dans le contexte vide, par le terme $\lambda p \,.\, p$. La dérivation ci-dessous montre comment l'utilisation de la règle de renforcement permet de prouver cette proposition :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[\cdots] \vdash p : \forall x : \mathsf{False} \,.\, \mathsf{False}}\,(\textsc{Var}) \quad \dfrac{}{[\cdots] \vdash x : \mathsf{False}}\,(\textsc{Var})}{[p : \forall x : \mathsf{False} \,.\, \mathsf{False};\ x : \mathsf{False}] \vdash p : \mathsf{False}}\,(\textsc{Inst})}{[p : \forall x : \mathsf{False} \,.\, \mathsf{False}] \vdash p : \mathsf{False}}\,(\textsc{Str})}{[] \vdash \lambda p \,.\, p : (\forall x : \mathsf{False} \,.\, \mathsf{False}) \to \mathsf{False}}\,(\textsc{Lam})$$

Autrement dit, nous venons de prouver la *négation* de la proposition $\forall x : \mathsf{False} \,.\, \mathsf{False}$, ce qui peut paraître très douteux puisque la quantification explicite analogue $\Pi x : \mathsf{False} \,.\, \mathsf{False}$ (également notée $\mathsf{False} \to \mathsf{False}$) est quant à elle trivialement prouvable par le terme $\lambda x \,.\, x$.

En fait, ceci n'a rien d'étonnant puisque dans ce cas précis, les prouvabilités des quantifications explicite et implicite ne sont pas équivalentes. En effet, s'il est possible d'appliquer la règle (Lam) au jugement valide

$$[x : \mathsf{False}] \vdash x : \mathsf{False}$$

afin de dériver le jugement

$$[] \vdash \lambda x \,.\, x : \Pi x : \mathsf{False} \,.\, \mathsf{False}$$

établissant que $\lambda x \,.\, x$ est une preuve de la proposition $\Pi x : \mathsf{False} \,.\, \mathsf{False}$, il n'est en revanche pas possible d'appliquer la règle (Gen) — qui est l'équivalent implicite de la règle (Lam) — à ce même jugement, en raison de la présence d'une condition de bord $x \notin FV(x)$ qui n'est pas satisfaite.

Autrement dit, le terme de preuve $\lambda x \,.\, x$ établissant la proposition $\Pi x : \mathsf{False} \,.\, \mathsf{False}$ ($\equiv \mathsf{False} \to \mathsf{False}$) ne nous donne aucune indication permettant de construire une preuve de $\forall x : \mathsf{False} \,.\, \mathsf{False}$ par simple transposition, pour la simple raison que le calcul implicite ne comporte pas d'abstraction implicite, et que l'utilisation de la règle (Gen) est soumise à une condition de bord qui n'existe pas dans le cas de la règle (Lam). (On notera que ce point confirme la profonde dissymétrie entre les constructions $\Pi x : T \,.\, U$ et $\forall x : T \,.\, U$.)

Par conséquent, il n'y a donc *a priori* rien d'incohérent à ce que dans le calcul implicite, la quantification explicite $\Pi x : \mathsf{False} \,.\, \mathsf{False}$ soit prouvable, alors que la quantification implicite analogue $\forall x : \mathsf{False} \,.\, \mathsf{False}$ est réfutable. Le théorème de normalisation forte établi au chapitre 7 confirmera cette impression *a posteriori*, puisqu'il nous permettra d'établir que la proposition $\forall x : \mathsf{False} \,.\, \mathsf{False}$ n'a effectivement pas de preuve dans le contexte vide. En effet, cette proposition qui n'est pas une forme stable ne peut pas avoir de preuve en forme normale de tête faible dans le contexte vide, en vertu du lemme 2.6.1.

Équivalence de CCI **et de** CCI$^-$ **vis-à-vis de la normalisation** Comme la cohérence logique du calcul implicite repose sur la propriété de normalisation forte, il est naturel de se demander si la présence de la règle de renforcement (Str) n'est pas susceptible de mettre en danger cette propriété. La réponse est clairement négative ainsi que l'indique le résultat suivant :

Proposition 2.6.4 — *Le Calcul des Constructions implicite satisfait la propriété de normalisation forte (resp. la propriété de normalisation faible) si et seulement si le Calcul des Constructions implicite restreint satisfait cette propriété également.*

Preuve. La partie directe est triviale puisque CCI$^-$ $\subset$ CCI, et la réciproque découle immédiatement du lemme 2.4.17 d'élimination de la règle de renforcement. □

Il est intéressant de s'attarder quelques instants sur la partie réciproque de la proposition énoncée ci-dessus, qui repose sur l'argument suivant. Si $\Gamma \vdash M : T$ est un jugement dérivable dans CCI alors, d'après le lemme d'élimination de la règle de renforcement, il existe une sorte s telle que le jugement $[c : \forall a : s\,.\,a]; \Gamma \vdash_r M : T$ est dérivable dans CCI$^-$, ce qui entraîne que M est fortement normalisable (resp. faiblement normalisable), sous l'hypothèse que tous les termes bien typés dans CCI$^-$ sont fortement normalisables (resp. faiblement normalisables).

Techniquement, la démonstration du lemme d'élimination de la règle de renforcement consiste à remplacer chacune des instances de la règle de renforcement

$$\frac{\Gamma; [x : T] \vdash M : U}{\Gamma \vdash M : U} \quad (x \notin FV(M) \cup FV(U))$$

par une application du lemme de substitutivité dans CCI$^-$ en remplaçant la variable x par un terme N de type T (sachant que d'après l'hypothèse $x \notin FV(M) \cup FV(U)$, la substitution est inopérante sur M comme sur U). Bien entendu, ceci n'est possible que dans la mesure où on a étendu le contexte avec une hypothèse suffisamment forte pour que tous les types — ou du moins tous les types intervenant dans la dérivation — soient habités, sans quoi il ne serait pas possible de constuire le terme N de type T dont l'existence est cruciale pour légitimer l'utilisation du lemme de substitutivité.

Par rapport aux modèles usuels — typiquement ceux que l'on utilise pour prouver un résultat de cohérence — les modèles de normalisation interprètent tous les types par des ensembles habités, ce qui est essentiel pour pouvoir interpréter tous les contextes, y compris les contextes incohérents. Il est donc clair que si $\mathcal{M}$ est un modèle de normalisation forte du calcul implicite *restreint*, il sera toujours possible d'appliquer dans $\mathcal{M}$ la technique décrite ci-dessus pour interpréter la règle de renforcement à l'aide d'une substitution adéquate, qui sera légitimée par le fait que dans $\mathcal{M}$, n'importe quel type est habité. Autrement dit :

> *Tout modèle de normalisation du calcul implicite restreint est également un modèle de normalisation du calcul implicite complet.*

Nous aurons l'occasion de revenir sur ce point au chapitre 3 lorsque nous introduirons les notions de modèles restreints et de modèles non restreints dans le cadre de l'interprétation du Calcul des Constructions implicite (*cf* paragraphe 3.3.5).

2.7 Exemples

Dans cette section, nous montrons l'expressivité du Calcul des Constructions implicite en comparant les codages imprédicatifs des types de données usuels dans le Calcul des Constructions et dans le calcul implicite respectivement. Afin d'illustrer les exemples qui suivent, nous supposerons définis certains types de données de base tels que le type des entiers ou des booléens, et nous travaillerons dans le contexte suivant :

$$\begin{array}{lcllcl} \mathsf{nat} & : & \mathsf{Set} & \mathsf{bool} & : & \mathsf{Set} \\ \mathsf{O} & : & \mathsf{nat} & \mathsf{true} & : & \mathsf{bool} \\ \mathsf{S} & : & \mathsf{nat} \to \mathsf{nat} & \mathsf{false} & : & \mathsf{bool} \end{array}$$

que nous considérerons tantôt comme un contexte du Calcul des Constructions, tantôt comme un contexte du calcul implicite.

2.7.1 Les listes

Le premier exemple sur lequel nous allons travailler est celui des listes, dont nous allons définir le codage imprédicatif dans le Calcul des Constructions puis dans le Calcul des Constructions implicite. Cet exemple est particulièrement intéressant car il fait apparaître dans le calcul implicite un phénomène nouveau par rapport au Calcul des Constructions, lié à la propagation du sous-typage à travers certains constructeurs de types (tels que les listes). Par ailleurs, nous verrons apparaître au paragraphe suivant — lorsque nous passerons aux *listes dépendantes* — un autre phénomène qui est le *partage de constructeurs*, c'est à dire la possibilité de définir deux types inductifs partageant les mêmes constructeurs (typiquement les listes et les listes dépendantes).

Listes à la Church Dans le Calcul des Constructions, le constructeur de type des listes est habituellement défini par

$$\begin{array}{rcl} \mathsf{list} & : & \mathsf{Set} \to \mathsf{Set} \\ & := & \lambda A : \mathsf{Set} \,.\, \Pi X : \mathsf{Set} \,.\, X \to (A \to X \to X) \to X \end{array}$$

tandis que les constructeurs (de valeurs) nil et cons qui lui sont associés sont donnés par :

$$\begin{array}{rcl} \mathsf{nil} & : & \Pi A : \mathsf{Set} \,.\, \mathsf{list}\ A \\ & := & \lambda A : \mathsf{Set} \,.\, \lambda X : \mathsf{Set} \,.\, \lambda x : X \,.\, \lambda \_ : (A \to X \to X) \,.\, x \end{array}$$

$$\begin{array}{rcl} \mathsf{cons} & : & \Pi A : \mathsf{Set} \,.\, A \to \mathsf{list}\ A \to \mathsf{list}\ A \\ & := & \lambda A : \mathsf{Set} \,.\, \lambda a : A \,.\, \lambda l : \mathsf{list}\ A \,.\, \lambda X : \mathsf{Set} \,.\, \lambda x : X \,.\, \lambda f : (A \to X \to X) \,.\, f\ a\ (l\ X\ x\ f) \end{array}$$

Le principal intérêt de cette définition des listes est qu'elle est *polymorphe*, ce qui nous donne la possibilité d'utiliser les trois nouvelles constantes list, nil et cons pour travailler sur des listes d'objets de n'importe quel type, que ce soit les listes d'entiers naturels (type list nat), les listes de booléens (type list bool) ou même les listes de listes (type list (list A)).

Malheureusement, le polymorphisme du constructeur de listes a aussi un coût, puisqu'il oblige de spécifier à chaque utilisation des constructeurs nil et cons le type considéré par le

biais d'une application supplémentaire. Par exemple, la liste de booléens [true; false; false] s'écrit-elle dans le Calcul des Constructions

$$\mathsf{cons\ bool\ true\ (cons\ bool\ false\ (cons\ bool\ false\ (nil\ bool)))} \quad : \quad \mathsf{list\ bool}.$$

Cependant, les listes que nous venons de définir ont un bon comportement calculatoire, puisqu'il est possible de définir par exemple une fonction polymorphe length qui à chaque liste associe sa longueur

$$\begin{array}{rcl} \mathsf{length} & : & \Pi A : \mathsf{Set}\,.\,\mathsf{list}\ A \to \mathsf{nat} \\ & := & \lambda A : \mathsf{Set}\,.\,\lambda l : \mathsf{list}\ A\,.\,l\ \mathsf{nat}\ \mathsf{O}\ (\lambda \_ : A\,.\,\lambda x : \mathsf{nat}\,.\,\mathsf{S}\ x) \end{array}$$

et de vérifier qu'elle a le comportement attendu vis-à-vis de la β-réduction :

$$\mathsf{length\ bool\ (cons\ bool\ true\ (cons\ bool\ false\ (cons\ bool\ false\ (nil\ bool))))} \quad \twoheadrightarrow_\beta \quad \mathsf{S\ (S\ (S\ O))}.$$

Listes à la Curry Dans le Calcul des Constructions implicite, il est possible de tirer parti de l'existence du produit implicite afin d'obtenir un comportement polymorphe "implicite" bien plus proche du polymorphisme à la ML. Pour cela, on définit les constantes list, nil et cons de la manière suivante :

$$\begin{array}{rcl} \mathsf{list} & : & \mathsf{Set} \to \mathsf{Set} \\ & := & \lambda A\,.\,\forall X : \mathsf{Set}\,.\,X \to (A \to X \to X) \to X \\ \\ \mathsf{nil} & : & \forall A : \mathsf{Set}\,.\,\mathsf{list}\ A \\ & := & \lambda x, \_\,.\,x \\ \\ \mathsf{cons} & : & \forall A : \mathsf{Set}\,.\,A \to \mathsf{list}\ A \to \mathsf{list}\ A \\ & := & \lambda a, l\,.\,\lambda x, f\,.\,f\ a\ (l\ x\ f) \end{array}$$

Remarquons que la définition des constructeurs nil et cons est maintenant identique à la définition usuelle des constructeurs nil et cons du λ-calcul pur (avec la définition habitielle des listes). Par ailleurs, nil et cons sont maintenant polymorphes au sens du calcul implicite, ce qui nous permet de réécrire la liste précédente d'une manière bien plus concise :

$$\mathsf{cons\ true\ (cons\ false\ (cons\ false\ nil))} \quad : \quad \mathsf{list\ bool}.$$

De plus, il est toujours possible de programmer la fonction length

$$\begin{array}{rcl} \mathsf{length} & : & \forall A : \mathsf{Set}\,.\,\mathsf{list}\ A \to \mathsf{nat} \\ & := & \lambda l\,.\,l\ \mathsf{O}\ (\lambda \_, x\,.\,\mathsf{S}\ x) \end{array}$$

à cette différence près que son "code" est maintenant exactement le même que celui de la fonction length qu'on aurait définie en λ-calcul pur. Cette fonction a toujours le comportement calculatoire attendu :

$$\mathsf{length\ (cons\ true\ (cons\ false\ (cons\ false\ nil)))} \quad \twoheadrightarrow_\beta \quad \mathsf{S\ (S\ (S\ O))}.$$

En revanche, l'utilisation du calcul implicite fait apparaître une propriété très intéressante des listes vis-à-vis du sous-typage, qui est que le type list A se comporte de manière covariante vis-à-vis du type A :

Proposition 2.7.1 (Covariance du type des listes) — *Pour tout contexte* Γ *et pour tous termes* A *and* B *de type* Set *dans* Γ *on a :*

$$\Gamma \vdash A \leq B \quad \Rightarrow \quad \Gamma \vdash \mathsf{list}\ A \leq \mathsf{list}\ B.$$

Preuve. Cette propriété résulte des propriétés de covariance/contravariance dans les produits, et est rendue possible par le fait que la seule occurrence de la variable A dans la définition du type $\mathsf{list}\ A$ figure en position positive (non stricte). □

2.7.2 Les listes dépendantes (vecteurs)

L'exemple du paragraphe précédent n'est en fait pas très novateur, car un examen attentif des définitions des constantes list, nil et cons dans le calcul implicite montre que nous avons utilisé le produit implicite précisément dans le cas où celui-ci correspondait à la règle de formation du produit dépendant *imprédicatif*. Par conséquent, le même exemple aurait pu être défini exactement de la même manière dans le système du cube des TAS correspondant au système $F\omega$ dans le cube de Barendregt.

En fait, la situation devient bien plus intéressante lorsqu'on considère non plus le type des listes, mais le type des *listes dépendantes*. Le type des listes dépendantes — que nous appellerons également *vecteurs* — diffère du type des listes par le fait qu'il dépend également d'un entier naturel n indiquant la longueur des listes avec lesquelles on travaille. Autrement dit, on ne travaillera plus seulement avec un type $\mathsf{list}\ A$ des listes à valeurs dans A, mais avec une famille de types $\mathsf{vect}\ A\ n$ dont les éléments sont les listes à valeurs dans A et dont la longueur est précisément l'entier n paramétrant cette famille de types.

Listes dépendantes à la Church Dans le Calcul des Constructions, le constructeur de type vect et les constructeurs de vecteurs nil' et cons' associés sont définis par :

$$\begin{array}{lcl}
\mathsf{vect} & : & \mathsf{Set} \to \mathsf{nat} \to \mathsf{Set} \\
& := & \lambda A : \mathsf{Set}\,.\,\lambda n : \mathsf{nat}\,.\,\Pi X : \mathsf{nat} \to \mathsf{Set}\,. \\
& & \qquad X\ \mathsf{O} \to (\Pi p : \mathsf{nat}\,.\,A \to X\ p \to X\ (\mathsf{S}\ p)) \to X\ n \\
\mathsf{nil}' & : & \Pi A : \mathsf{Set}\,.\,\mathsf{vect}\ A\ \mathsf{O} \\
& := & \lambda A : \mathsf{Set}\,.\,\lambda X : \mathsf{nat} \to \mathsf{Set}\,. \\
& & \qquad \lambda x : X\ \mathsf{O}\,.\,\lambda _ : (\Pi p : \mathsf{nat}\,.\,A \to X\ p \to X\ (\mathsf{S}\ p))\,.\,x \\
\mathsf{cons}' & : & \Pi A : \mathsf{Set}\,.\,\Pi n : \mathsf{nat}\,.\,A \to \mathsf{vect}\ A\ n \to \mathsf{vect}\ A\ (\mathsf{S}\ n) \\
& := & \lambda A : \mathsf{Set}\,.\,\lambda n : \mathsf{nat}\,.\,\lambda a : A\,.\,\lambda v : \mathsf{vect}\ A\ n\,.\,\lambda X : \mathsf{nat} \to \mathsf{Set}\,. \\
& & \qquad \lambda x : X\ \mathsf{O}\,.\,\lambda f : (\Pi p : \mathsf{nat}\,.\,A \to X\ p \to X\ (\mathsf{S}\ p))\,.\,f\ n\ a\ (v\ X\ x\ f)
\end{array}$$

Notons que dans le Calcul des Constructions, l'introduction d'une nouvelle famille de types $\mathsf{vect}\ A\ n$ pour représenter les listes dépendantes s'accompagne par la définition de nouveaux constructeurs nil' et cons' différents des constructeurs nil et cons utilisés pour les listes. Par ailleurs, la finesse de typage ainsi accrue coûte une application supplémentaire à chaque utilisation du constructeur cons' auquel il faut maintenant fournir, en plus du type A,

la longueur de la liste à laquelle il s'applique. Par exemple, la liste de booléens [true; false; false] s'écrit à présent

$$\begin{array}{l}\mathsf{cons'\ bool\ (S\ (S\ O))\ true\ (cons'\ bool\ (S\ O)\ false\ (cons'\ bool\ O\ false\ (nil'\ bool)))}\\ \quad :\quad \mathsf{vect\ bool\ (S\ (S\ (S\ O)))}.\end{array}$$

Un autre aspect intéressant de ce codage des listes dépendantes est la possibilité de programmer la fonction length de deux manières très différentes. La première

$$\begin{array}{rcl}\mathsf{length'} & : & \Pi A : \mathsf{Set}\,.\,\Pi n : \mathsf{nat}\,.\,\mathsf{vect}\ A\ n \to \mathsf{nat}\\ & := & \lambda A : \mathsf{Set}\,.\,\lambda n : \mathsf{nat}\,.\,\lambda v : \mathsf{vect}\ A\ n\,.\\ & & \qquad v\ (\lambda\_ : \mathsf{nat}\,.\,\mathsf{nat})\ \mathsf{O}\ (\lambda\_ : \mathsf{nat}\,.\,\lambda\_ : A\,.\,\lambda x : \mathsf{nat}\,.\,\mathsf{S}\ x)\end{array}$$

reprend l'algorithme de la fonction length définie sur les listes, en itérant la fonction successeur S à chaque nœud cons' rencontré, à partir de l'entier O. Cependant, l'utilisation de cette fonction length'

$$\begin{array}{l}\mathsf{length'\ bool\ (S\ (S\ (S\ O)))}\\ \quad\mathsf{(cons'\ bool\ (S\ (S\ O))\ true}\\ \quad\quad\mathsf{(cons'\ bool\ (S\ O)\ false}\\ \quad\quad\quad\mathsf{(cons'\ bool\ O\ false\ (nil'\ bool))))}\quad \twoheadrightarrow_\beta \quad \mathsf{S\ (S\ (S\ O))}\end{array}$$

montre que le calcul reconstruit précisément le second argument S (S (S O)) passé à la fonction length', qui n'est autre que la longueur de la liste dépendante passée en troisième argument. Ceci suggère une nouvelle implémentation de la fonction length qui est la suivante :

$$\begin{array}{rcl}\mathsf{length''} & : & \Pi A : \mathsf{Set}\,.\,\Pi n : \mathsf{nat}\,.\,\mathsf{vect}\ A\ n \to \mathsf{nat}\\ & := & \lambda A : \mathsf{Set}\,.\,\lambda n : \mathsf{nat}\,.\,\lambda\_ : \mathsf{vect}\ A\ n\,.\,n\end{array}$$

Si d'un point de vue *extensionnel*, les fonctions length' et length'' sont équivalentes — elles donnent le même résultat pour un même vecteur passé en argument — ces deux fonctions sont très différentes *intensionnellement* :

- La fonction length' effectue le calcul de longueur en *temps linéaire* puisqu'elle construit un nœud successeur S pour chaque nœud cons' rencontré, sans tenir compte de son second argument n, lequel contient pourtant le résultat recherché.
- La fonction length'' effectue le calcul en *temps constant*, puisqu'elle se contente de renvoyer son second argument n en ignorant complètement le vecteur v passé en troisième argument.

Listes dépendantes à la Curry Dans le Calcul des Constructions implicite, il est possible de définir le constructeur de type des listes dépendantes de la manière suivante :

$$\begin{array}{rcl}\mathsf{vect} & : & \mathsf{Set} \to \mathsf{nat} \to \mathsf{Set}\\ & := & \lambda A, n\,.\,\forall X : \mathsf{nat} \to \mathsf{Set}\,.\,X\ \mathsf{O} \to (\forall p : \mathsf{nat}\,.\,A \to X\ p \to X\ (\mathsf{S}\ p)) \to X\ n\end{array}$$

Notons que ce codage des listes dépendantes n'a pas d'équivalent dans le cube des TAS à cause de la quantification interne $\forall\, p : \mathsf{nat}$ introduisant un *type dépendant implicite*.[10] De plus,

[10]Rappelons que le type nat est par hypothèse un habitant de la sorte imprédicative Set, et donc que la variable p : nat se trouve au même niveau (dans la hiérarchie d'univers) que les termes de preuve. La quantification p : nat n'est donc pas imprédicative.

la définition des constructeurs nil et cons afférents est rigoureusement la même que dans le cas des listes

$$\begin{array}{lcl} \mathsf{nil} & : & \forall A : \mathsf{Set} \,.\, \mathsf{vect}\ A\ \mathsf{O} \\ & := & \lambda x, \_ \,.\, x \end{array}$$

$$\begin{array}{lcl} \mathsf{cons} & : & \forall A : \mathsf{Set} \,.\, \forall n : \mathsf{nat} \,.\, A \to \mathsf{vect}\ A\ n \to \mathsf{vect}\ A\ (\mathsf{S}\ n) \\ & := & \lambda a, l \,.\, \lambda x, f \,.\, f\ a\ (l\ x\ f) \end{array}$$

à cette différence près qu'on leur a assigné des types différents.[11]

En d'autres termes, les types des listes $\mathsf{list}\ A$ et des listes dépendantes $\mathsf{vect}\ A\ n$ partagent les mêmes constructeurs dans le calcul implicite. Les listes et les listes dépendantes se construisent de la même manière, la seule différence étant apportée par le typage, qui est effectué de manière plus fine dans le cas des listes dépendantes. Ainsi, la liste de booléens

$$\mathsf{cons\ true\ (cons\ false\ (cons\ false\ nil))}$$

a donc à la fois les types $\mathsf{list\ bool}$ et $\mathsf{vect\ bool\ (S\ (S\ (S\ O)))}$.

Dans le Calcul des Constructions implicite, il est même possible d'utiliser la notion de sous-typage pour dériver que le type des vecteurs est un sous-type du type des listes :

Proposition 2.7.2 — *Pour tout contexte* Γ *et pour tous termes* A *et* n *tels que* $\Gamma \vdash A : \mathsf{Set}$ *et* $\Gamma \vdash n : \mathsf{nat}$*, le jugement de sous-typage suivant est dérivable :*

$$\Gamma\ \vdash\ \mathsf{vect}\ A\ n\ \leq\ \mathsf{list}\ A.$$

Par ailleurs, le type des listes dépendantes se comporte de manière covariante vis-à-vis de l'argument de type A, le paramètre de longueur n étant quant à lui fixé :

Proposition 2.7.3 — *Pour tout contexte* Γ*, pour tous termes* A*,* B *de type* Set *dans* Γ *et pour tout terme* n *de type* nat *dans* Γ*, on a :*

$$\Gamma \vdash A \leq B \quad \Rightarrow \quad \Gamma\ \vdash\ \mathsf{vect}\ A\ n\ \leq\ \mathsf{vect}\ B\ n.$$

Notons que la relation de sous-typage $\mathsf{vect}\ A\ n\ \leq\ \mathsf{list}\ A$ entraîne que la fonction length définie précédemment sur les listes

$$\begin{array}{lcl} \mathsf{length} & : & \forall A : \mathsf{Set} \,.\, \mathsf{list}\ A \to \mathsf{nat} \\ & := & \lambda l \,.\, l\ \mathsf{O}\ (\lambda \_, x \,.\, \mathsf{S}\ x) \end{array}$$

a également le type $\forall A : \mathsf{Set} \,.\, \forall n : \mathsf{nat} \,.\, \mathsf{vect}\ A\ n \to \mathsf{nat}$ (en utilisant la propriété de contravariance dans les types flèche). Cette fonction length commune aux listes et aux listes dépendantes dans le calcul implicite est l'équivalent (pour le cas des listes dépendantes) de la fonction length' définie ci-dessus dans le cadre du Calcul des Constructions. En revanche, il n'est maintenant plus possible d'écrire un équivalent implicite de la fonction length'' qui se contentait de retourner la valeur de son second argument. En effet, dans le type de la fonction length

$$\forall A : \mathsf{Set} \,.\, \forall n : \mathsf{nat} \,.\, \mathsf{vect}\ A\ n \to \mathsf{nat},$$

[11] On remarquera également que la constante nil — égale par définition au terme $\lambda x, \_ \,.\, x$ — a également le type $\forall X : \mathsf{Set} \,.\, X \to (X \to X) \to X$ qui est le type des entiers de Church du calcul implicite. Dans le Calcul des Constructions implicite comme dans le λ-calcul pur, la constante nil et l'entier de Church 0 sont identiques.

l'entier n est maintenant un "argument implicite" qui ne correspond plus à aucun argument réel de la fonction. On peut se servir de cet entier figurant dans le type pour établir le typage correct d'une expression où figure la fonction length, mais l'entier n en lui-même n'a plus d'existence au niveau calculatoire.

2.7.3 Égalité de Leibniz

Dans ce paragraphe, nous allons revenir au monde propositionnel en comparant les codages de l'égalité de Leibniz dans le Calcul des Constructions et dans le calcul implicite.

Dans le Calcul des Constructions, l'égalité (au sens de Leibniz) est définie par :

$$\begin{array}{rcl} \mathsf{eq} & : & \Pi A : \mathsf{Set} \,.\, A \to A \to \mathsf{Prop} \\ & := & \lambda A : \mathsf{Set} \,.\, \lambda x, y : A \,.\, \Pi P : A \to \mathsf{Prop} \,.\, P\ x \to P\ y \end{array}$$

Cette définition imprédicative de la relation d'égalité nous permet de dériver immédiatement les propriétés de réflexivité, de symétrie et de transitivité :

$$\begin{array}{rcl} \mathsf{eq\_refl} & : & \Pi A : \mathsf{Set} \,.\, \Pi x : A \,.\, \mathsf{eq}\ A\ x\ x \\ & := & \lambda A : \mathsf{Set} \,.\, \lambda x : A \,.\, \lambda P : A \to \mathsf{Prop} \,.\, \lambda p : P\ x \,.\, p \end{array}$$

$$\begin{array}{rcl} \mathsf{eq\_sym} & : & \Pi A : \mathsf{Set} \,.\, \Pi x, y : A \,.\, \mathsf{eq}\ A\ x\ y \to \mathsf{eq}\ A\ y\ x \\ & := & \lambda A : \mathsf{Set} \,.\, \lambda x, y : A \,.\, \lambda e : \mathsf{eq}\ A\ x\ y \,.\, e\ (\lambda z : A \,.\, \mathsf{eq}\ A\ z\ x)\ (\mathsf{eq\_refl}\ A\ x) \end{array}$$

$$\begin{array}{rcl} \mathsf{eq\_trans} & : & \Pi A : \mathsf{Set} \,.\, \Pi x, y, z : A \,.\, \mathsf{eq}\ A\ x\ y \to \mathsf{eq}\ A\ y\ z \to \mathsf{eq}\ A\ x\ z \\ & := & \lambda A : \mathsf{Set} \,.\, \lambda x, y, z : A \,.\, \lambda e_1 : \mathsf{eq}\ A\ x\ y \,.\, \lambda e_2 : \mathsf{eq}\ A\ y\ z \,. \\ & & \quad \lambda P : A \to \mathsf{Prop} \,.\, \lambda p : P\ x \,.\, e_2\ P\ (e_1\ P\ p) \end{array}$$

Bien que les preuves ci-dessous soient de taille raisonnable, leur comportement calculatoire est noyé sous un grand nombre de λ-abstractions et d'applications "administratives". Un des avantages du calcul implicite est de permettre de distinguer parmi ces λ-abstractions et ces applications celles d'entre elles qui ont un réel contenu calculatoire.

Pour cela, considérons l'égalité de Leibniz dans le Calcul des Constructions implicite, laquelle est définie par :

$$\begin{array}{rcl} \mathsf{eq} & : & \Pi A : \mathsf{Set} \,.\, A \to A \to \mathsf{Prop} \\ & := & \lambda A \,.\, \lambda x, y \,.\, \forall P : A \to \mathsf{Prop} \,.\, P\ x \to P\ y \end{array}$$

Le contenu calculatoire des termes de preuve des propriétés de réflexivité, de symétrie et de transitivité de l'égalité

$$\begin{array}{rcl} \mathsf{eq\_refl} & : & \forall A : \mathsf{Set} \,.\, \forall x : A \,.\, \mathsf{eq}\ A\ x\ x \\ & := & \lambda p \,.\, p \end{array}$$

$$\begin{array}{rcl} \mathsf{eq\_sym} & : & \forall A : \mathsf{Set} \,.\, \forall x, y : A \,.\, \mathsf{eq}\ A\ x\ y \to \mathsf{eq}\ A\ y\ x \\ & := & \lambda e \,.\, e\ (\lambda p \,.\, p) \end{array}$$

$$\begin{array}{rcl} \mathsf{eq\_trans} & : & \forall A : \mathsf{Set} \,.\, \forall x, y, z : A \,.\, \mathsf{eq}\ A\ x\ y \to \mathsf{eq}\ A\ y\ z \to \mathsf{eq}\ A\ x\ z \\ & := & \lambda e_1, e_2, p \,.\, e_2\ (e_1\ p) \end{array}$$

apparaît alors beaucoup plus clairement : la preuve de réflexivité est la fonction identité, la preuve de transitivité n'est autre que le combinateur de composition de fonctions, et la preuve de symétrie est la fonction retournant son argument appliqué à la fonction identité.

Remarquons également que dans les types des termes eq_refl, eq_sym et eq_trans, la quantification sur le paramètre de type $A : \mathsf{Set}$ est implicite, alors qu'elle est explicite dans le cas de la définition de la relation d'égalité eq elle-même. La raison en est que l'argument $A : \mathsf{Set}$ est utilisé explicitement pour définir la famille des *types* correspondant à l'égalité (par la quantification $\forall P : A \to \mathsf{Prop}$), alors que ce même paramètre A n'est jamais utilisé dans les *termes* eq_refl, eq_sym et eq_trans servant à prouver des égalités.

Différences avec les systèmes d'arguments implicites Avant de passer à l'exemple suivant, il est intéressant de remarquer que dans la plupart des systèmes d'arguments implicites mis en œuvre dans les systèmes d'aide à la démonstration basés sur la théorie des types, l'argument de type A dans l'expression $\mathsf{eq}\ A\ x\ y$ est traité généralement de manière implicite. En pratique, ceci permet d'offrir à l'utilisateur une syntaxe concrète "$x = y$" bien plus confortable dans laquelle il n'est pas nécessaire de rappeler le type de x et de y qui, dans cette situatuon, est facilement inférable (en recalculant par exemple le type de x).

Dans le codage imprédicatif que nous venons de présenter, l'argument A est implicite dans les preuves des propositions exprimant la réflexivité, la symétrie et la transitivité de l'égalité, mais est *explicite* dans le constructeur de type lui-même

$$\mathsf{eq} \quad : \quad \Pi A : \mathsf{Set}\,.\, A \to A \to \mathsf{Prop}.$$

Dans le calcul implicite, il semble impossible de définir un codage de l'égalité de Leibniz dans lequel on pourrait remplacer le produit dépendant explicite ci-dessus par un produit dépendant implicite. Si tel était le cas, cela signifierait que les propositions exprimant l'égalité de deux mêmes termes M_1 et M_2 dans deux types différents A et A' (l'un étant sous-type de l'autre) seraient convertibles, et auraient pour ainsi dire la même signification.

Pour comprendre le problème soulevé par une telle identification, nous allons temporairement considérer un langage muni de types enregistrements (ou "*records*") avec sous-typage par oubli de champs. Considérons par exemple deux types définis par

$$A = \{\ell_1 : \mathsf{nat}\} \qquad \text{et} \qquad A' = \{\ell_1 : \mathsf{nat};\ \ell_2 : \mathsf{bool}\}.$$

Intuitivement, A' est un sous-type de A, en ce sens que tout objet de type A' peut-être vu comme un objet de type A, simplement en oubliant la présence du champ ℓ_2. Considérons maintenant deux programmes M_1 et M_2 de type A' définis par

$$M_1 = \{\ell_1 = 3;\ \ell_2 = \mathsf{true}\} \qquad \text{et} \qquad M_2 = \{\ell_1 = 3;\ \ell_2 = \mathsf{false}\}.$$

Clairement, ces deux objets sont différents dans A' car les champs $M_1.\ell_2$ et $M_2.\ell_2$ diffèrent. Considérons à présent les termes M_1 et M_2 en tant qu'objets de type A (qui est un sur-type de A'). Par définition, les prédicats de type $A \to \mathsf{Prop}$ ne peuvent accéder qu'au champ ℓ_1, et ne peuvent donc pas distinguer les termes M_1 et M_2. Au sens de l'égalité de Leibniz, les termes M_1 et M_2, qui sont différents dans A', deviennent égaux dans A.

Cet exemple montre en quoi la signification de l'égalité dépend en général du type dans lequel on l'observe, et justifie intuitivement le fait que dans un formalisme muni d'un riche sous-typage tel que le calcul implicite, il soit nécessaire de conserver de manière explicite l'argument

de type A dans la proposition $\mathsf{eq}\ A\ x\ y$, ne serait-ce que pour éviter des identifications qui seraient dommageables à la cohérence de la théorie. Bien entendu, on pourrait objecter que l'exemple que nous venons de donner manque de pertinence dans le cadre d'un formalisme dépourvu de types enregistrements, mais nous verrons au chapitre 5 que ce phénomène lié à l'interprétation de l'égalité en présence de sous-typage peut déjà s'observer dans le modèle sans avoir à ajouter au calcul implicite la moindre construction syntaxique supplémentaire (*cf* paragraphe 5.2.5).[12]

Pour revenir à la question des arguments implicites, il semble donc clair que les informations qui sont susceptibles d'être supprimées dans les termes sans mettre en danger la cohérence de la théorie n'ont que peu de rapports avec les sous-expressions qui peuvent être reconstruites automatiquement par les mécanismes d'arguments implicites tels qu'ils sont implémentés dans les systèmes actuels (qui, rappelons-le, conservent ces arguments implicites dans la représentation interne des termes).

2.7.4 Quantifications existentielles

Afin de renchérir sur les remarques effectuées dans le paragraphe précédent, nous allons maintenant étudier le cas du codage imprédicatif de la quantification existentielle. Dans le Calcul des Constructions, le quantificateur existentiel ex et le constructeur d'introduction $\mathsf{ex\_intro}$ correspondant sont définis par :

$$\begin{array}{lcl}
\mathsf{ex} & : & \Pi A : \mathsf{Set} \,.\, (A \to \mathsf{Prop}) \to \mathsf{Prop} \\
 & := & \lambda A : \mathsf{Set} \,.\, \lambda P : A \to \mathsf{Prop} \,.\, \Pi X : \mathsf{Prop} \,.\, (\Pi x : A \,.\, P\ x \to X) \to X \\
 & & \\
\mathsf{ex_intro} & : & \Pi A : \mathsf{Set} \,.\, \Pi P : A \to \mathsf{Prop} \,.\, \Pi x : A \,.\, P\ x \to \mathsf{ex}\ A\ P \\
 & := & \lambda A : \mathsf{Set} \,.\, \lambda P : A \to \mathsf{Prop} \,.\, \lambda x_0 : A \,.\, \lambda p_0 : P\ x_0 \,. \\
 & & \quad \lambda X : \mathsf{Prop} \,.\, \lambda f : (\Pi x : A \,.\, P\ x \to X) \,.\, f\ x_0\ p_0
\end{array}$$

Pour un type $A : \mathsf{Set}$ et un prédicat $P : A \to \mathsf{Prop}$ donnés, on peut alors construire une preuve du type $\mathsf{ex}\ A\ P$ (également noté $\exists x : A \,.\, P\ x$) dès qu'on dispose d'un témoin $x_0 : A$ et d'une preuve $p_0 : P\ x_0$ en construisant l'application

$$\mathsf{ex\_intro}\ A\ P\ x_0\ p_0 \quad : \quad \exists x : A \,.\, P\ x.$$

À ce stade, on peut alors se demander quels arguments parmi les termes A, P, x_0 et p_0 vont devenir implicites dans la preuve ci-dessus lorsque nous nous intéresserons au codage équivalent dans le Calcul des Constructions implicite. Aussi surprenant que cela puisse paraître, il n'y a pas une réponse à cette question, mais deux réponses différentes, lesquelles correspondent aux deux manières d'encoder le quantificateur existentiel dans le calcul implicite.

Quantificateur existentiel faible La première façon d'encoder le quantificateur existentiel dans le calcul implicite consiste à remplacer chaque occurrence du symbole Π par le symbole $\forall$ dans la définition du constructeur de type ex telle qu'elle a été donnée dans le Calcul

[12] Dans la suite de cette thèse, je ne considérerai pas davantage la question des types enregistrements. Cependant, les quelques discussions que j'ai eues avec Jean-Yves Girard à ce sujet m'ont convaincues que le paradigme de sous-typage qui sera présenté dans le chapitre 5 (et qui est formellement très proche de celui qui est utilisé en Ludique [30]) est parfaitement à même de prendre en compte le sous-typage sur les types enregistrements tel qu'il est décrit ci-dessus, et cela de manière très naturelle.

des Constructions. On obtient alors la définition du *quantificateur existentiel faible* du calcul implicite :

$$\begin{array}{lcl} \mathsf{ex} & : & \Pi A : \mathsf{Set} \,.\, (A \to \mathsf{Prop}) \to \mathsf{Prop} \\ & := & \lambda A \,.\, \lambda P \,.\, \forall X : \mathsf{Prop} \,.\, (\forall x : A \,.\, P\ x \to X) \to X \\ \\ \mathsf{ex\_intro} & : & \forall A : \mathsf{Set} \,.\, \forall P : A \to \mathsf{Prop} \,.\, \forall x : A \,.\, P\ x \to \mathsf{ex}\ A\ P \\ & := & \lambda p, f \,.\, f\ p \end{array}$$

Si $A : \mathsf{Set}$ est un type et $P : A \to \mathsf{Prop}$ un prédicat sur le type A, on peut former une preuve du type $\mathsf{ex}\ A\ P$ (noté également $\exists x : A \,.\, P\ x$) en appliquant le terme $\mathsf{ex\_intro}$ au à un terme de preuve $p_0 : P\ x_0$

$$\mathsf{ex\_intro}\ p_0 \quad : \quad \exists x : A \,.\, P\ x$$

sans que le témoin d'existence x_0 (*i.e.* tel que $p_0 : P\ x_0$) ne figure dans cette preuve ! Comme cette preuve ne dépend pas du témoin lui-même, mais seulement du fait que la proposition $P\ x$ est prouvable pour un certain terme $x : A$, on parle de *quantificateur existentiel faible*. Insistons sur le fait que le terme de preuve $p_0 : P\ x_0$ peut très bien ne pas faire intervenir le témoin x_0 lui-même, auquel cas il est impossible de récupérer ce témoin à partir de la preuve $\mathsf{ex\_intro}\ p_0 \ : \ \exists x : A \,.\, P\ x$.[13]

Dans le cadre de cette définition du quantificateur existentiel, le schéma d'élimination correspondant

$$\frac{p \ : \ \exists x : A \,.\, P\ x \qquad f \ : \ \forall x : A \,.\, P\ x \to X}{(p\ f) \ : \ X}$$

comporte une seconde prémisse $\forall x : A \,.\, P\ x \to X$ construite à partir d'une quantification implicite, laquelle est *a priori* plus difficile à établir que la quantification explicite correspondante $\Pi x : A \,.\, P\ x \to X$.[14] Intuitivement, cette seconde prémisse demande une preuve de l'implication $P\ x \to X$ qui soit *uniforme* vis-à-vis de la variable $x : A$, l'uniformité de la preuve étant ici exprimée par l'utilisation du produit implicite.

Quantificateur existentiel fort La discussion ci-dessus suggère la possibilité de définir un autre quantficateur existentiel ex^* pour lequel on a le schéma d'élimination

$$\frac{p \ : \ \exists^* x : A \,.\, P\ x \qquad f \ : \ \Pi x : A \,.\, P\ x \to X}{(p\ f) \ : \ X}$$

dans lequel la seconde prémisse repose sur une quantification explicite plus faible que la quantification implicite correspondante. Ce quantificateur ex^* et le constructeur d'introduction $\mathsf{ex}^*\mathsf{\_intro}$ qui lui correspond sont naturellement définis par :

$$\begin{array}{lcl} \mathsf{ex}^* & : & \Pi A : \mathsf{Set} \,.\, (A \to \mathsf{Prop}) \to \mathsf{Prop} \\ & := & \lambda A \,.\, \lambda P \,.\, \forall X : \mathsf{Prop} \,.\, (\Pi x : A \,.\, P\ x \to X) \to X \\ \\ \mathsf{ex}^*\mathsf{\_intro} & : & \forall A : \mathsf{Set} \,.\, \forall P : A \to \mathsf{Prop} \,.\, \Pi x : A \,.\, P\ x \to \mathsf{ex}^*\ A\ P \\ & := & \lambda x, p \,.\, \lambda f \,.\, f\ x\ p \end{array}$$

[13] Notons que dans les systèmes d'arguments implicites intégrés à l'heure actuelle dans les vérificateurs de preuves en théorie des types, le témoin est rarement un argument implicite, ne serait-ce que parce qu'il est très difficile à reconstruire mécaniquement à partir du type de la preuve p (ce qui nécessite de mettre en œuvre un mécanisme d'unification d'ordre supérieur assez coûteux, et qui est susceptible de boucler).

[14] Rappelons qu'on a trivialement l'implication $(\forall x : A \,.\, P\ x \to X) \to (\Pi x : A \,.\, P\ x \to X)$ — prouvée par le terme $\lambda f, x \,.\, f$ — mais que l'implication réciproque est en générale fausse dans le calcul implicite.

Remarquons que maintenant, le constructeur d'introduction $\mathsf{ex}^*\_\mathsf{intro}$ requiert un argument de plus qui est le témoin $x_0 : A$ pour lequel on a une preuve $p_0 : P\ x_0$:

$$\mathsf{ex}^*\_\mathsf{intro}\ x_0\ p_0 \quad : \quad \exists^* x : A\,.\,P\ x.$$

D'un point de vue logique, la quantification existentielle forte implique la quantification existentielle faible correspondante :

$$\lambda pf\,.\,p\ (\lambda x\,.\,f) \quad : \quad \forall A : \mathsf{Set}\,.\,\forall P : A \to \mathsf{Prop}\,.\,(\exists^* x : A\,.\,P\ x) \to (\exists x : A\,.\,P\ x).$$

Non-équivalence logique entre les deux quantificateurs existentiels Nous terminerons ce paragraphe par la description d'un contre-exemple montrant que la réciproque de l'implication ci-dessus est fausse. Pour cela, nous admettrons temporairement le théorème de normalisation forte qui sera établi au chapitre 7, d'où découle (par le biais de la proposition 2.6.3) la cohérence du Calcul des Constructions implicite. Ce résultat étant admis, considérons à présent :

1. les propositions False et True définies par $\mathsf{False} = \forall A : \mathsf{Prop}\,.\,A$ et $\mathsf{True} = \mathsf{False} \to \mathsf{False}$.
2. le type de données void défini par $\mathsf{void} = \forall A : \mathsf{Set}\,.\,A$.
3. le prédicat $P : \mathsf{void} \to \mathsf{Prop}$ défini par $P = \lambda x\,.\,\mathsf{True}$.

En vertu de la cohérence du calcul implicite, les types $\mathsf{False} : \mathsf{Prop}$ et $\mathsf{void} : \mathsf{Set}$ sont inhabités dans le contexte vide (le type void n'étant rien d'autre que le jumeau de False dans Set). En revanche, le type $\mathsf{True} : \mathsf{Prop}$ est habité par le terme clos $\mathsf{id} = \lambda x\,.\,x : \mathsf{True}$.

Lemme 2.7.4 — *La proposition $\exists x : \mathsf{void}\,.\,P\ x$ est prouvable dans le contexte vide.*

Preuve. Dans le contexte $[x : \mathsf{void}]$ le jugement $[x : \mathsf{void}] \vdash \mathsf{id} : P\ x$ est valide puisque $P\ x =_\beta \mathsf{True}$. Par conséquent, on a

$$[x : \mathsf{void}] \vdash \mathsf{ex\_intro\ id} : \mathsf{ex\ void}\ P$$

d'où l'on tire en appliquant la règle de renforcement (Str) le jugement

$$[] \vdash \mathsf{ex\_intro\ id} : \mathsf{ex\ void}\ P$$

en effaçant l'hypothèse $x : \mathsf{void}$ qui n'apparaît ni dans le terme $\mathsf{ex\_intro\ id}$ ni dans le type $\mathsf{ex\ void}\ P$ ($\equiv \exists x : \mathsf{void}\,.\,P\ x$). □

Il nous reste à présent à établir que la proposition existentielle forte correspondante n'est pas prouvable dans le contexte vide :

Lemme 2.7.5 — *La proposition $\exists^* x : \mathsf{void}\,.\,P\ x$ n'est pas prouvable dans le contexte vide.*

Preuve. Supposons que M est une preuve de la proposition $\exists^* x : \mathsf{void}\,.\,P\ x$ dans le contexte vide. Par application de la règle de conversion (Conv), on a

$$\begin{array}{rcl} [] \vdash M & : & \mathsf{ex}^*\ \mathsf{void}\ P \\ & : & \forall X : \mathsf{Prop}\,.\,(\Pi x : \mathsf{void}\,.\,P\ x \to X) \to X \\ & : & \forall X : \mathsf{Prop}\,.\,(\mathsf{void} \to \mathsf{True} \to X) \to X \end{array}$$

Soit M' le terme obtenu en remplaçant dans M toutes les occurrences de Set par Prop. Comme cette transformation commute trivialement avec la relation de typage, le terme M' est bien typé dans le contexte vide et

$$[] \quad \vdash \quad M' \quad : \quad \forall X : \mathsf{Prop} \,.\, (\mathsf{False} \to \mathsf{True} \to X) \to X.$$

(Notons que le remplacement de toutes les occurrences de Set par Prop dans le jugement précédent a transformé le terme void dans le type ci-dessus en son jumeau propositionnel False.) En instanciant la variable X par False, il vient

$$[] \quad \vdash \quad M' \quad : \quad (\mathsf{False} \to \mathsf{True} \to \mathsf{False}) \to \mathsf{False}$$

d'où l'on tire par application du terme $\lambda x, \_ \,.\, x$ de type $\mathsf{False} \to \mathsf{True} \to \mathsf{False}$ le jugement

$$[] \quad \vdash \quad M' \; (\lambda x, \_ \,.\, x) \quad : \quad \mathsf{False},$$

ce qui entre en contradiction avec la cohérence du calcul implicite. □

Il est intéressant d'analyser la technique de preuve ci-dessus et de voir pourquoi une tentative d'adaptation de cette preuve au cas de la quantification existentielle *faible* (afin d'obtenir un paradoxe !) ne peut que conduire à un échec. Informellement, la preuve du lemme précédent est basée sur une élimination de la quantification existentielle forte à l'aide du schéma

$$\frac{\exists^* x : \mathsf{void} \,.\, P \; x \qquad \Pi x : \mathsf{void} \,.\, P \; x \to X}{X}$$

dans le cas où $X = \mathsf{False}$. Cette élimination est rendue possible par le fait que la seconde prémisse $\Pi x : \mathsf{void} \,.\, P \; x \to \mathsf{False}$ — égale à $\Pi x : \mathsf{False} \,.\, P \; x \to \mathsf{False}$ modulo une identification des sortes imprédicatives Set et Prop — admet une preuve immédiate qui est le terme $\lambda x, \_ \,.\, x$.

Pour pouvoir transposer cette idée au cas de la quantification existentielle faible basée sur le schéma d'élimination

$$\frac{\exists x : \mathsf{void} \,.\, P \; x \qquad \forall x : \mathsf{void} \,.\, P \; x \to X}{X}$$

il faut d'abord construire une preuve de la seconde prémisse $\forall x : \mathsf{void} \,.\, P \; x \to X$ dans le cas où $X = \mathsf{False}$. Heureusement pour la cohérence de la théorie, les propositions

$$\Pi x : \mathsf{void} \,.\, P \; x \to \mathsf{False} \qquad \text{et} \qquad \forall x : \mathsf{void} \,.\, P \; x \to \mathsf{False}$$

ne sont pas logiquement équivalentes. La première de ces deux propositions est prouvable en effectuant une simple *abstraction* de la variable $x : \mathsf{void}$ dans le jugement

$$[x : \mathsf{void}] \quad \vdash \quad \lambda \_ \,.\, x \quad : \quad P \; x \to \mathsf{False}$$

pour obtenir une preuve $\lambda x, \_ \,.\, x$ de la proposition $\Pi x : \mathsf{void} \,.\, P \; x \to \mathsf{False}$. En revanche, il n'est pas possible de *généraliser* le jugement ci-dessus par rapport à la variable $x : \mathsf{void}$ pour prouver la proposition $\forall x : \mathsf{void} \,.\, P \; x \to \mathsf{False}$, puisque la variable x est libre dans le terme $\lambda \_ \,.\, x$.

Autrement dit, la proposition $\forall x : \mathsf{void} \,.\, P \; x \to \mathsf{False}$ n'est pas prouvable dans le calcul implicite, car la seule preuve de $P \; x \to \mathsf{False}$ que l'on peut construire dans le contexte $[x : \mathsf{void}]$ *n'est pas uniforme par rapport à la variable* x.

Remarquons également que la proposition $\forall x : \mathsf{void} \,.\, P \; x \to \mathsf{False}$ est en fait égale au produit implicite non-dépendant $\forall \_ : \mathsf{void} \,.\, \mathsf{True} \to \mathsf{False}$, lequel est habité par les mêmes termes que la proposition $\mathsf{True} \to \mathsf{False}$ qui est manifestement fausse.

2.7.5 Incohérence forte

La plupart des théories des types standard — que ce soit la théorie des types de Martin-Löf, le Calcul des Constructions avec univers ou le Calcul des Constructions inductives — reposent sur un principe d'homogénéité très fort, qui est que dans n'importe quel contexte, même incohérent, un terme donné ne peut pas être à la fois une fonction (dont le type est un produit dépendant) et un type (dont le type est une sorte).

Dans le Calcul des Constructions implicite, la possibilité de construire des intersections à n'importe quel niveau de la hiérarchie d'univers — et plus particulièrement dans les univers prédicatifs — entraîne naturellement la rupture de ce principe d'homogénéité dans les contextes fortement incohérents.

Définition 2.7.6 (Incohérence forte) — Soit Γ un contexte. On appelle *incohérence forte* dans le contexte Γ tout terme M de type $\forall T : \mathsf{Type}_1 \,.\, T$ dans le contexte Γ. Lorsqu'un tel terme existe, on dit que le contexte Γ est *fortement incohérent*.

De manière similaire, on peut définir pour chaque univers prédicatif Type_i ($i > 0$) une notion d'incohérence forte qui lui est relative. Une incohérence forte au niveau i est donc un habitant commun à tous les types de la sorte Type_i, c'est-à-dire un habitant du type $\forall T : \mathsf{Type}_i \,.\, T$. (Notons que le type $\forall T : \mathsf{Type}_i \,.\, T$ exprimant l'incohérence forte du niveau i est lui-même défini au niveau $i + 1$.)

Comme pour tout indice $i > 0$ on a les relations de sous-typage

$$\forall T : \mathsf{Type}_i \,.\, T \quad \leq \quad \forall T : \mathsf{Type}_1 \,.\, T \quad \leq \quad \forall A : \mathsf{Prop} \,.\, A$$

(d'après la relation de cumulativité), toute incohérence forte au niveau $i > 0$ est également une incohérence forte au niveau 1, mais aussi une incohérence propositionnelle, c'est-à-dire une preuve du faux implicite.

L'intérêt de la notion d'incohérence forte est d'illustrer la disparition du principe d'homogénéité entre les fonctions et les types dans les contextes fortement incohérents du calcul implicite. En effet, la proposition suivante montre que dans le calcul implicite, une incohérence forte est à la fois une fonction et un type. De plus, toute incohérence forte est un terme qui est son propre type !

Proposition 2.7.7 — *Soient Γ un contexte et M un terme. Si $\Gamma \vdash M : \forall T : \mathsf{Type}_1 \,.\, T$, alors les jugements suivants sont dérivables :*

1. $\Gamma \vdash M : \forall A : \mathsf{Prop} \,.\, A$ *(M est une preuve du faux implicite)*
2. $\Gamma \vdash M : \mathsf{Prop}$ *(M est un type)*
3. $\Gamma \vdash M : \mathsf{Prop} \to \mathsf{Prop}$ *(M est une fonction)*
4. $\Gamma \vdash M : M$ *(M a pour type M)*

Preuve. Supposons $\Gamma \vdash M : \forall T : \mathsf{Type}_1 \,.\, T$. Le premier point découle de la relation de sous-typage $\forall T : \mathsf{Type}_1 \,.\, T \leq \forall A : \mathsf{Prop} \,.\, A$ valide dans n'importe quel contexte bien formé. Les points 2 et 3 résultent d'une instanciation de la variable $T : \mathsf{Type}_1$ par les termes Prop et $\mathsf{Prop} \to \mathsf{Prop}$ respectivement. Enfin, comme $\Gamma \vdash M : \mathsf{Prop}$ et $\Gamma \vdash M : \forall A : \mathsf{Prop} \,.\, A$, il suffit d'instancier dans ce dernier jugement la variable A par le terme M lui-même pour obtenir $\Gamma \vdash M : M$. □

Afin d'illustrer cette propriété, nous avons fait figurer ci-dessous une dérivation du jugement $[x : \forall T : \mathsf{Type}_1 \,.\, T] \vdash x : x$, valide dans le calcul implicite restreint.

$$\dfrac{\dfrac{\vdots}{\dfrac{[x : \forall T : \mathsf{Type}_1 \,.\, T] \vdash}{[\cdots] \vdash x : \forall T : \mathsf{Type}_1 \,.\, T}\ (\textsc{Var})} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\dfrac{[x : \forall T : \mathsf{Type}_1 \,.\, T] \vdash}{[\cdots] \vdash x : \forall x : T \,.\, \mathsf{Type}_1}\ (\textsc{Var})} \qquad \dfrac{\vdots}{\dfrac{[x : \forall T : \mathsf{Type}_1 \,.\, T] \vdash}{[\cdots] \vdash \mathsf{Prop} : \mathsf{Type}_1}\ (\textsc{Sort})}}{[\cdots] \vdash x : \mathsf{Prop}}\ (\textsc{Inst})}{[\cdots] \vdash x : \mathsf{Type}_1}\ (\textsc{Cum})}{[x : \forall T : \mathsf{Type}_1 \,.\, T] \vdash x : x}\ (\textsc{Inst})$$

La notion d'incohérence forte jouera un rôle très important au chapitre 7 consacré à la preuve du théorème de normalisation forte, car c'est autour de cette notion que s'organisera la construction du modèle de normalisation, lequel doit être à même d'interpréter tous les contextes, y compris les contextes fortement incohérents.

Deuxième partie

Modèles cohérents

Chapitre 3

Introduction aux modèles

Dans ce chapitre, nous présentons les concepts généraux utilisés tout au long de la deuxième partie de cette thèse, qui est consacrée à la construction et à l'étude des modèles du calcul implicite dans les espaces cohérents.

Avant d'entrer dans le vif du sujet, nous commencerons par effectuer un bref rappel des principaux modèles du Calcul des Constructions avec univers (CCω) [42], qui est sans doute le formalisme le plus proche du calcul implicite de par sa structure. Loin de vouloir effectuer une revue exhaustive des travaux effectués dans ce domaine, nous nous limiterons à la présentation des deux modèles standard de CCω : le modèle ensembliste booléen dans lequel les propositions sont interprétées par des booléens [64, 5], et le modèle de CCω basé sur la notion de réalisabilité dans la catégorie des ω-sets [42].

Dans un deuxième temps, nous discuterons des limites de l'approche ensembliste naïve, dont nous pourrons constater l'inadéquation dans le cadre de l'étude sémantique d'un formalisme à la Curry tel que le calcul implicite. Nous nous tournerons alors vers la théorie des modèles du λ-calcul en introduisant une notion de λ-*modèle sous-extensionnel*, qui constitue un raffinement naturel de la notion de λ-modèle [8] permettant de contrôler le comportement de la η-réduction dans les modèles non extensionnels. À partir de cette notion, nous proposerons ensuite une axiomatisation abstraite des modèles du calcul implicite, dont nous dégagerons les propriétés générales.[1] En particulier, nous montrerons que l'existence d'un modèle du calcul implicite satisfaisant une certaine condition de restriction entraîne la cohérence logique du Calcul des Constructions implicite restreint (*cf* paragraphe 2.4.3 page 72), indépendamment de tout résultat de normalisation.[2]

Pour terminer ce chapitre introductif, nous présenterons rapidement les outils standards de la théorie des ensembles (tels que la notion d'inaccessibilité) que nous aurons l'occasion d'utiliser de manière récurrente dans les chapitres suivants.

3.1 Modèles du Calcul des Constructions avec univers

Le Calcul des Constructions avec univers (CCω) est le fragment purement fonctionnel du Calcul des Constructions étendu (ECC, [42]), dans lequel on ne considère ni les sommes

[1] La construction proprement dite de ces modèles ne sera effectuée qu'à partir du chapitre 5

[2] La cohérence du calcul implicite complet (*i.e.* avec règle de renforcement) ne sera réellement établie qu'au chapitre 7, comme une conséquence indirecte du théorème de normalisation forte.

dépendantes $\Sigma\, x : T\,.\, U$ de ECC ni le constructeur et les destructeurs qui leur sont associés. Ce formalisme est un PTS (à la Church) dont la syntaxe est donnée par

$$\begin{array}{rcl} M, N, T, U & ::= & x \\ & | & \mathsf{Prop} \quad | \quad \mathsf{Type}_i \qquad (i > 0) \\ & | & \Pi x : T\,.\, U \\ & | & \lambda x : T\,.\, M \\ & | & M\ N \end{array}$$

Les règles de bonne formation de contexte, de typage des variables, des abstractions et des applications sont données par les règles de typage standard dans les PTS [9, 26]. Les sortes sont hiérarchisées par les règles

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Prop} : \mathsf{Type}_1} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Type}_i : \mathsf{Type}_{i+1}}$$

et sont closes par formation de produit dépendant

$$\frac{\Gamma \vdash T : \mathsf{Prop} \qquad \Gamma; [x : T] \vdash U : \mathsf{Prop}}{\Gamma \vdash \Pi x : T\,.\, U : \mathsf{Prop}} \qquad \frac{\Gamma \vdash T : \mathsf{Type}_i \qquad \Gamma; [x : T] \vdash U : \mathsf{Type}_i}{\Gamma \vdash \Pi x : T\,.\, U : \mathsf{Type}_i}$$

Enfin, la sorte Prop est imprédicative

$$\frac{\Gamma \vdash T : \mathsf{Type}_i \qquad \Gamma; [x : T] \vdash U : \mathsf{Prop}}{\Gamma \vdash \Pi x : T\,.\, U : \mathsf{Prop}}$$

Par rapport au cadre strict des PTS, le Calcul des Constructions avec univers se démarque en remplaçant la règle de conversion par une règle de *cumulativité*

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash T' : s \qquad T \preceq T'}{\Gamma \vdash M : T'}$$

dont la condition de bord n'est plus une condition de β-convertibilité $T =_\beta T'$, mais une condition plus générale dite de *cumulativité*, laquelle repose sur le préordre $T \preceq T'$ défini sur l'ensemble des termes non typés à partir des règles d'inférence suivantes :

$$\frac{T =_\beta T'}{T \preceq T'} \qquad \frac{}{\mathsf{Prop} \preceq \mathsf{Type}_1} \qquad \frac{}{\mathsf{Type}_i \preceq \mathsf{Type}_{i+1}}$$

$$\frac{T \preceq T' \qquad T' \preceq T''}{T \preceq T''} \qquad \frac{U \preceq U'}{\Pi x : T\,.\, U \ \preceq\ \Pi x : T\,.\, U'}$$

Statut de la relation de cumulativité Cette présentation de la règle de cumulativité dans le Calcul des Constructions avec univers diffère sensiblement de la présentation de la règle de cumulativité dans le Calcul des Constructions implicite telle que nous l'avons effectuée au chapitre 2. (Nous faisons ici abstraction des questions de sous-typage liées au produit implicite, qui sont bien entendu spécifiques au calcul implicite.) Ces différences portent essentiellement sur deux points.

La première différence concerne le statut de la règle de cumulativité dans les deux calculs. Dans le Calcul des Constructions avec univers, la règle de cumulativité est une règle définie

en dehors de la relation de typage et sur des termes éventuellement mal typés, au même titre que la β-réduction. Dans le calcul implicite au contraire, la cumulativité est exprimée sur les sortes par une règle de (sous-)typage spécifique, et son extension à travers les produits dépendants est effectuée au moyen de la règle (Ext), qui assure dans le formalisme la propriété de préservation du typage par la η-réduction. Dans le premier cas, cette définition en dehors de la relation de typage s'explique par le fait que la relation $T \preceq T'$ a un statut très proche du statut de la relation de β-conversion : elle ne dépend pas du contexte et est décidable dans le cas où les termes T et T' ont des formes normales (ce qui est bien entendu le cas si les termes T et T' sont bien typés). Dans le calcul implicite au contraire, la relation de sous-typage est liée à un contexte particulier, et est en outre indécidable en dépit de la propriété de normalisation forte qui sera établie au chapitre 7.

La seconde différence réside dans la propagation de la relation de cumulativité à travers les produits dépendants, qui n'est pas traitée de la même façon dans les deux formalismes. Dans CCω, on ne dispose que d'une seule règle de covariance par rapport au codomaine des types fonctionnels

$$\frac{U \preceq U'}{\Pi x\!:\!T\,.\,U \;\preceq\; \Pi x\!:\!T\,.\,U'}$$

alors que dans le calcul implicite, la relation de sous-typage est plus riche puisqu'elle assure également la propriété de contravariance du sous-typage par rapport au domaine des types fonctionnels

$$\frac{T' \;\leq\; T}{\Pi x\!:\!T\,.\,U \;\leq\; \Pi x\!:\!T'\,.\,U}$$

Par exemple, $\mathsf{Prop} \to \mathsf{Type}_1$ est un sous-type de $\mathsf{Prop} \to \mathsf{Type}_2$ dans les deux formalismes, mais $\mathsf{Type}_2 \to \mathsf{Prop}$ n'est un sous-type de $\mathsf{Type}_1 \to \mathsf{Prop}$ que dans le calcul implicite.

Cette différence d'appréciation n'est pas anodine, mais fait apparaître au contraire deux conceptions, deux interprétations différentes de la théorie des types. Le Calcul des Constructions avec univers est une théorie des types à la Church qui s'interprète très naturellement dans le cadre de la théorie des ensembles. Il est donc normal que ce formalisme ne prenne en compte que la propriété de covariance sur le codomaine, qui est la seule propriété de sous-typage relative aux types fonctionnels qui ait une interprétation immédiate en théorie des ensembles.[3] Au contraire, le calcul implicite est une théorie des types à la Curry, dans laquelle l'absence d'annotation de type dans l'abstraction est une nécessité fondamentale (que nous avons déjà évoquée dans les chapitres 1 et 2). Pour cette raison, nous verrons au fil des chapitres suivants que la théorie des domaines permet bien mieux d'interpréter les termes du calcul implicite que ne saurait le faire une interprétation ensembliste au sens strict du terme (voir à ce sujet la discussion de la section 3.2.)

3.1.1 Le modèle ensembliste classique booléen

La construction du modèle classique (encore appelé modèle *proof-irrelevant*) du Calcul des Constructions avec univers repose sur une idée très simple, qui est que chaque concept de ce

[3]Cette affirmation doit être nuancée puisque les présentations plus récentes de CCω et de ECC [54] incorporent cette propriété de contravariance vis-à-vis du domaine dans la définition de la relation de cumulativité. Cependant, il semblerait que cette extension naturelle du formalisme n'ait été que très rarement étudiée sous l'angle de la sémantique dénotationnelle. Par une traduction immédiate dans le calcul implicite, on vérifiera aisément que les deux modèles présentés aux chapitres 5 et 6 interprètent sans difficulté cette extension de la relation de cumulativité dans CCω.

formalisme possède un équivalent en théorie des ensembles par lequel on peut l'interpréter. Ainsi, on interprétera :

- la relation de typage $M : T$ par la relation d'appartenance $M \in T$;
- la relation de cumulativité $T \preceq T'$ par la relation d'inclusion $T \subset T'$
- les fonctions de la théorie des types $\lambda x : T \,.\, M_x$ par les fonctions (c'est-à-dire les *applications*) de la théorie des ensembles habituellement notées $(x_{\in T} \mapsto M_x)$
- l'application de la théorie des types MN par l'application d'une fonction à son argument $M(N)$ telle qu'elle est définie en théorie des ensembles ;
- le produit dépendant de la théorie des types $\Pi x : T \,.\, U_x$ par le produit d'une famille d'ensembles $\prod_{x \in T} U_x$.

La correspondance complète entre les termes du Calcul des Constructions avec univers et les notions par lesquelles on les interprète en théorie des ensembles est donnée par la figure 3.1. Afin de mettre en valeur le naturel de cette interprétation, nous avons volontairement omis de faire figurer les "crochets du sémanticien", et nous avons indiqué les dépendances par des variables figurant en indice. Dans ce qui suit, nous nous en tiendrons à cette présentation intuitive, en laissant volontairement de côté de nombreux détails techniques. On pourra trouver une présentation formelle de ce modèle dans [5, 64] par exemple.

$$
\begin{array}{lcllcl}
\lambda x : T \,.\, M_x & = & (x_{\in T} \mapsto M_x) & M\ N & = & M(N) \\
\Pi x : T \,.\, U_x & = & \displaystyle\prod_{x \in T} U_x & \mathsf{Type}_i & = & \mathcal{U}_i \\
\mathsf{Prop} & = & \{\mathbf{0};\ \mathbf{1}\} & \mathbf{0} = \varnothing & & \mathbf{1} = \{\bullet\}
\end{array}
$$

FIG. 3.1 – Interprétation des termes de CCω en théorie des ensembles

Malgré son apparente simplicité, cette interprétation effectue une opération non-triviale, puisqu'elle transforme n'importe quelle fonction $\lambda x : T \,.\, M$ de la théorie des types — c'est-à-dire un programme — en une fonction $(x_{\in T} \mapsto M_x)$ de la théorie des ensembles — c'est-à-dire un ensemble de couples argument/résultat. De la même façon, un type T est transformé en un ensemble contenant les dénotations des termes M de type T.

Autrement dit, cette interprétation fait apparaître un contenu extensionnel (répondant à la question : "quel est le résultat du calcul ?") là où dans la théorie des types d'origine ne semblait figurer qu'un contenu intensionnel (répondant à la question : "comment effectuer le calcul ?"). En particulier, deux termes β-convertibles sont systématiquement interprétés par le même ensemble. On voit donc que cette interprétation (typique en sémantique dénotationnelle) s'intéresse davantage au *résultat* du calcul plutôt qu'au calcul lui-même, et qu'elle est de ce fait bien plus apte à prouver des résultats liés à la *prouvabilité* (tels que la cohérence logique) plutôt que des résultats liés à la *calculabilité* (tels que la normalisation forte).[4]

[4]Il est toutefois possible de prouver la normalisation forte d'un calcul comme CCω avec des outils de sémantique dénotationnelle. Dans ce cas, il faut enrichir l'interprétation avec des objets syntaxiques (typiquement des candidats de réductibilité) afin de conserver dans le modèle une trace de la manière dont s'effectue le calcul. C'est là l'idée centrale des preuves de normalisation à partir des modèles, introduite par [6]. Nous verrons un tel exemple de modèle "mixte" dans le chapitre 7 consacré à la preuve de normalisation forte du calcul implicite.

Interprétation de l'imprédicativité Le modèle que nous sommes en train de décrire est un modèle classique en ce sens qu'il interprète les propositions par les booléens $\mathbf{0}$ ou $\mathbf{1}$ (et par conséquent la sorte Prop par la paire $\{\mathbf{0};\mathbf{1}\}$). Comme en théorie des types les propositions sont les types de leurs preuves, il faut donc interpréter les propositions par des ensembles dont les éléments sont les dénotations des preuves de ces propositions. Il est alors naturel de poser $\mathbf{0} = \varnothing$ et $\mathbf{1} = \{\bullet\}$ où $\bullet$ est un objet arbitraire destiné à interpréter tous les termes de preuve de CCω.

Dans CCω, l'imprédicativité de Prop résulte de ce qu'un produit dépendant $\Pi x : T \,.\, U_x$ dont toutes les composantes U_x sont des propositions est lui-même une proposition, bien que le type T puisse être arbitrairement élevé dans la hiérarchie d'univers. Dans le cadre de notre interprétation ensembliste, cette propriété d'imprédicativité se traduit naturellement de la manière suivante :

Proposition 3.1.1 (Produit ensembliste d'une famille de booléens) — *Soit T un ensemble quelconque. Si $(U_x)_{x\in T}$ est une famille d'ensembles indicée par T telle que U_x soit un booléen $\mathbf{0} = \varnothing$ ou $\mathbf{1} = \{\bullet\}$ pour tout $x \in T$, on a alors*

$$\prod_{x\in T} U_x \quad = \quad \begin{cases} \varnothing & \text{si } U_x = \varnothing \text{ pour au moins un } x \in T \\ \{(x_{\in T} \mapsto \bullet)\} & \text{si } U_x = \{\bullet\} \text{ pour tout } x \in T \end{cases}$$

où $\prod_{x\in T} U_x$ désigne le produit ensembliste standard.

Autrement dit, le produit ensembliste $\prod_{x\in T} U_x$ est lui même soit égal à l'ensemble vide, soit égal à un singleton. Dans le second cas cependant, le produit $\prod_{x\in T} U_x$ n'est pas égal au booléen $\mathbf{1} = \{\bullet\}$ lui-même, mais à un singleton dont l'unique élément est une fonction constante $(x_{\in T} \mapsto \bullet)$ associant la "preuve" $\bullet$ à tout objet de T.

Afin de faire en sorte que le produit $\prod_{x\in T} U_x$ soit systématiquement un booléen, il faut mettre en place un procédé technique permettant d'identifier n'importe quelle fonction constante $(x_{\in T} \mapsto \bullet)$ à l'objet $\bullet$ lui-même. Nous ne définirons pas ici les mécanismes d'une telle identification dont on trouvera une description détaillée dans [5].[5]

Interprétation des univers prédicatifs La suite des univers prédicatifs $(\mathsf{Type}_i)_{i>0}$ est interprétée par une suite d'ensembles $(\mathcal{U}_i)_{i>0}$ satisfaisant les conditions suivantes :

1. $\{\mathbf{0};\mathbf{1}\} \in \mathcal{U}_1$ (axiome $\mathsf{Prop} : \mathsf{Type}_1$)
2. $\{\mathbf{0};\mathbf{1}\} \subset \mathcal{U}_1$ (cumulativité $\mathsf{Prop} \preceq \mathsf{Type}_1$)
3. $\mathcal{U}_i \in \mathcal{U}_{i+1}$ (axiome $\mathsf{Type}_i : \mathsf{Type}_{i+1}$)
4. $\mathcal{U}_i \subset \mathcal{U}_{i+1}$ (cumulativité $\mathsf{Type}_i \preceq \mathsf{Type}_{i+1}$)
5. si $T \in \mathcal{U}_i$ et $U_x \in \mathcal{U}_i$ pour tout $x \in T$, alors $\prod_{x\in T} U_x \in \mathcal{U}_i$ (clôture de l'univers Type_i par les produits dépendants)

La construction de la suite d'ensembles $(\mathcal{U}_i)_{i>0}$ repose (au moins à partir de l'indice $i = 2$) sur les notions d'ensemble inaccessible et de cardinal inaccessible, sur lesquelles nous reviendrons plus en détails dans la section 3.4.

[5] Ce mécanisme consiste essentiellement à changer la représentation des fonctions en théorie des ensembles, de manière à ce que chaque fonction constante $(x_{\in T} \mapsto \bullet)$ soit codée de la même manière que l'objet $\bullet$ lui-même.

Pour terminer cette description, mentionnons que dans le cadre du modèle ensembliste booléen, la notion de cardinal inaccessible n'est vraiment utile qu'à partir du second univers prédicatif. Pour interpréter le premier univers Type_1 (qui est l'unique univers prédicatif du Calcul des Constructions), on a recours à un ensemble dont la particularité la plus notable est de ne contenir que des ensembles finis. (Ceci n'est possible que parce que les ensembles $\mathbf{0} = \varnothing$, $\mathbf{1} = \{\bullet\}$ et $\{\mathbf{0}; \mathbf{1}\}$ sont eux-mêmes finis, et que le produit fini d'une famille d'ensembles finis constitue encore un ensemble fini). Cette remarque est importante, car elle permet de voir que dans le Calcul des Constructions (avec un seul univers prédicatif), il n'est pas possible de construire un type *prouvablement* infini car dans ce formalisme, la dénotation de n'importe quel type est un ensemble fini.[6]

Validité de l'interprétation L'interprétation que nous venons de décrire ne fait un sens que dans la mesure où elle est *valide*, c'est-à-dire que dans le contexte vide, la dénotation d'un terme M de type T est bien un élément de la dénotation de T. (Cette propriété de validité s'exprime plus généralement dans n'importe quel contexte, grâce au mécanisme des valuations.) Une conséquence importante de cette propriété de validité est que la dénotation d'un type T prouvable dans le contexte vide est un ensemble non-vide (car contenant au moins la dénotation d'une preuve de T). Par contraposition, il est donc clair qu'un type dont la dénotation est l'ensemble vide ne peut pas être prouvable dans le contexte vide.

Un tel exemple de type "dénotationnellement vide" est la proposition représentant le faux $\Pi A : \mathsf{Prop} \,.\, A$ dans le Calcul des Constructions avec univers, dont la dénotation est

$$\prod_{A \in \{\varnothing; \{\bullet\}\}} A \quad = \quad \varnothing,$$

ce qui nous donne une preuve de la cohérence logique de CCω à relativement peu de frais,[7] et surtout, une preuve de cohérence logique indépendante de tout résultat de normalisation.

Remarquons cependant que la réciproque est fausse en général : une proposition non-prouvable dans le contexte vide peut très bien s'interpréter par la proposition non vide $\mathbf{1} = \{\bullet\}$. Dans le modèle classique, le contre exemple le plus intéressant (et qui donne toute sa justification à l'adjectif "classique") est le tiers-exclu

$$\Pi A : \mathsf{Prop} \,.\, A \vee \neg A$$

dont la dénotation est le booléen $\mathbf{1} = \{\bullet\}$. Ceci est dû au fait que les codages imprédicatifs des connecteurs $\neg$, $\vee$, $\wedge$ et $\Rightarrow$ s'interprètent par leurs équivalents classiques en reconstruisant les tables de vérité correspondantes. Un autre exemple de proposition "vraie" dans le modèle classique — mais non prouvable — est l'égalité des preuves

$$\Pi A : \mathsf{Prop} \,.\, \Pi x, y : A \,.\, x =_A y$$

(où $x =_A y$ désigne le codage imprédicatif de l'égalité de Leibniz sur A), ce qui découle naturellement du fait que nous avons interprété tous les termes de preuves par le même objet.

[6] Nous verrons au chapitre 9 que cette situation change dès qu'on passe au second univers prédicatif Type_2, puisque dans cet univers il est possible de construire un ensemble des entiers prédicatifs pour lesquels tous les axiomes de l'arithmétique de Heyting sont réalisés par des termes clos.

[7] Hormis l'utilisation de cardinaux inaccessibles...

3.1.2 Le modèle intuitionniste

La simplicité du modèle ensembliste booléen provient essentiellement de l'identification de tous les termes de preuve via la fonction d'interprétation. Cette simplicité du modèle classique est également sa principale limite : ce modèle interprète uniquement la notion de prouvabilité, mais n'interprète pas réellement les preuves elles-mêmes. De ce fait, le modèle classique n'explique en rien la structure intuitionniste de CCω. S'il permet de justifier la non-contradiction du tiers-exclu dans Prop (par le fait que cet axiome est valide dans le modèle), il n'explique pas pourquoi le tiers-exclu n'est pas prouvable dans CCω.

À ce titre, le modèle de CCω dans la catégorie des ω-sets [42] répond de manière bien plus satisfaisante au problème posé par l'étude de la structure intuitionniste de CCω. Contrairement au modèle ensembliste classique qui interprète les propositions par les booléens, le modèle basé sur les ω-sets interprète les propositions par les PER[8] et les termes de preuves par des fonctions récursives partielles.

Dans le modèle intuitionniste, les types ne sont pas interprétés par des ensembles seulement, mais par des ensembles munis d'une *relation de réalisabilité*, appelés ω-sets. Formellement, un ω-set est un couple $X = (|X|, \Vdash_X)$ formé par

- un ensemble $|X|$ quelconque, appelé *support* de X ;
- une relation surjective $(\Vdash_X) \subset \omega \times |X|$ appelée *relation de réalisabilité* sur X.

La relation $n \Vdash_X x$ (avec $n \in \omega$ et $x \in |X|$) se lit "n réalise x dans X", ou encore "n est un réalisateur de x dans X". Cette relation est surjective en ce sens que tout objet x du support admet au moins un réalisateur, c'est-à-dire :

$$\forall\, x \in |X| \qquad \exists\, n \in \omega \qquad n \Vdash_X x.$$

Intuitivement, les réalisateurs d'un objet $x \in |X|$ donné désignent toutes les fonctions récursives partielles susceptibles de représenter l'objet x au sein du type X. Bien entendu, on identifie ici les fonctions récursives partielles avec leurs codes de Gödel, lesquels sont donnés par une énumération fixée une fois pour toutes.

La classe des ω-sets constitue naturellement une catégorie, notée ω-**Set**, dont les morphismes $f : X \to Y$ sont les fonctions de $|X|$ dans $|Y|$ pour lesquelles il existe un entier n tel que

$$\forall\, x \in |X| \quad \forall\, p \in \omega \qquad p \Vdash_X x \quad \Rightarrow \quad n \cdot p \Vdash_Y f(x),$$

où $n \cdot p$ représente l'application de Kleene.[9] Remarquons que les morphismes de X dans Y forment également un ω-set, que l'on note $X \to Y$. Son support est l'ensemble des morphismes de X dans Y, et la relation de réalisabilité $n \Vdash_{X \to Y} f$ est donnée par

$$n \Vdash_{X \to Y} f \qquad \equiv \qquad \forall\, x, p \quad p \Vdash_X x \quad \Rightarrow \quad n \cdot p \Vdash_Y f(x).$$

Notons que cette relation est surjective puisque, par définition, les morphismes de X dans Y sont précisément les applications $f : |X| \to |Y|$ réalisées par au moins un entier n au sens de la définition de $\Vdash_{X \to Y}$.

[8] *Partial Equivalence Relations.*

[9] La notation $n \cdot p$ désigne le résultat de l'application de la n-ième fonction récursive partielle à l'entier p, dans le cas où le résultat de cette application est défini. Cette définition de la notation $n \cdot p$ est donc partielle, et chaque fois que nous l'emploierons, nous supposerons implicitement qu'elle est bien définie.

Par ailleurs, la catégorie des ω-sets dispose d'un produit cartésien (que nous ne définirons pas ici) et satisfait plus généralement tous les axiomes qui font de ω-**Set** une catégorie cartésienne fermée, dont la correspondance $(X, Y) \mapsto [X \to Y]$ est l'exponentielle.

Interprétation du produit dépendant Dans la catégorie des ω-sets, l'interprétation du produit dépendant est une généralisation immédiate de la définition de l'exponentielle $X \to Y$. Si X est un ω-set, et $(Y_x)_{x\in|X|}$ une famille de ω-sets indicée par le support de X, on note $\Pi(x \in X; Y_x)$ le ω-set dont le support est constitué de l'ensemble des fonctions $f \in \prod_{x\in|X|} |Y_x|$ pour lesquelles il existe un entier n tel que

$$\forall\, x \in |X| \quad \forall\, p \in \omega \qquad p \Vdash_X x \quad \Rightarrow \quad n \cdot p \Vdash_{Y_x} f(x),$$

et dont la relation de réalisabilité est donnée par

$$n \Vdash_{\Pi(X;Y)} f \qquad \equiv \qquad \big(\forall\, x, p \quad p \Vdash_X x \quad \Rightarrow \quad n \cdot p \Vdash_{Y_x} f(x)\big).$$

Là encore, la relation $n \Vdash_{\Pi(X;Y)} f$ est surjective précisément parce que nous n'avons considéré que les fonctions $f \in \prod_{x\in|X|} |Y_x|$ réalisées par au moins un entier $n \in \omega$. Ce point est important, car il signifie que l'inclusion

$$\big|\Pi(x \in X; Y_x)\big| \quad \subset \quad \prod_{x\in|X|} |Y_x|$$

est stricte en général. En pratique, la condition de réalisabilité imposée aux éléments du support de $\Pi(x \in X; Y_x)$ restreint considérablement sa cardinalité.

Ce point est en effet crucial, car dans le modèle ensembliste, c'est précisément le problème d'explosion combinatoire lié à la sémantique du produit dépendant qui conduit naturellement à interpréter les preuves de manière triviale et les propositions par des ensembles ayant au plus un élément. Dans la catégorie des ω-sets, cette explosion combinatoire est maîtrisée grâce à l'introduction de la notion de réalisabilité, ce qui permet alors de proposer une interprétation non triviale de l'imprédicativité, basée sur les ω-sets modestes.

Interprétation de l'imprédicativité On dit d'un ω-set X qu'il est *modeste* si la relation de réalisabilité $n \Vdash_X x$ est fonctionnelle, c'est-à-dire :

$$\forall\, x, y \in |X| \quad \forall\, n \in \omega \qquad n \Vdash_X x \quad \text{et} \quad n \Vdash_X y \quad \Rightarrow \quad x = y.$$

Les ω-sets modestes forment une sous-catégorie pleine de ω-**Set**, que nous noterons ici ω-**Mod**. La propriété essentielle des ω-sets modestes est la suivante :

Proposition 3.1.2 (Imprédicativité) — *Si $(Y_x)_{x\in|X|}$ est une famille de ω-sets modestes indicée par le support d'un ω-set X quelconque, alors le produit dépendant $\Pi(x \in X; Y_x)$ est un ω-set modeste.*

La sous-catégorie pleine des ω-sets modestes constitue donc un bon candidat pour interpréter la sorte imprédicative Prop. La seule difficulté est liée au fait que la catégorie ω-**Mod** est une grande catégorie, dont la classe des objets ne forme pas un ensemble. Heureusement, cette grande catégorie est en réalité équivalente à la petite catégorie des PER.[10] L'équivalence de catégorie entre ω-**Mod** et **PER** est définie de la manière suivante :

[10]Rappelons qu'un PER est une relation d'équivalence partielle sur ω, c'est-à-dire une relation $R \subset \omega \times \omega$ symétrique et transitive.

- Si $R \subset \omega \times \omega$ est un PER, on note $\mathrm{In}(R)$ le ω-set dont le support est $|\mathrm{In}(R)| = \omega/R$ (où ω/R désigne le quotient du domaine de R par la relation d'équivalence induite) et dont la relation de réalisabilité est donnée par la relation d'appartenance aux classes d'équivalence partielle :

$$n \Vdash_{\mathrm{In}(R)} x \quad \equiv \quad n \in x \qquad (x \in \omega/R).$$

  Le ω-set $\mathrm{In}(R)$ ainsi défini est clairement un ω-set modeste (la fonctionnalité découlant du fait que les classes d'équivalence partielles sont disjointes).

- Si X est un ω-set modeste, il est facile de remarquer que les ensembles de réalisateurs associés à deux objets du support distincts sont des ensembles d'entiers disjoints, qui forment naturellement une relation d'équivalence partielle sur ω. Plus formellement, si X est un ω-set modeste, on note $\mathrm{Out}(X)$ la relation sur ω définie par

$$(n, m) \in \mathrm{Out}(X) \quad \equiv \quad \exists\, x \in |X| \quad n \Vdash_X x \quad \text{et} \quad m \Vdash_X x.$$

  Là encore il est facile de vérifier que la relation $\mathrm{Out}(X)$ est un PER.

Intuitivement, la transformation d'un ω-set en PER consiste simplement à oublier le support de X et à ne conserver que les classes d'équivalences partielles induites par la relation de réalisabilité sur X.

Proposition 3.1.3 (Équivalence de catégories) — *Les correspondances* In : **PER** $\to$ ω-**Mod** *et* Out : ω-**Mod** $\to$ **PER** *constituent une équivalence de catégories entre les catégories* **PER** *et* ω-**Mod** *: pour tout PER R et pour tout ω-set modeste X on a en effet*

$$\mathrm{Out}(\mathrm{In}(R)) \;=\; R \qquad \textit{et} \qquad \mathrm{In}(\mathrm{Out}(X)) \;\approx\; X.$$

(Dans le second cas, il n'y a pas égalité mais seulement un isomorphisme)

Dans le modèle intuitionniste de CCω, on interprète alors Prop par l'ensemble **PER** dont les éléments (les relations d'équivalence partielles) pourront à tout moment être considérés comme des ω-sets modestes via l'équivalence de catégories décrite ci-dessus.

Interprétation des univers prédicatifs L'interprétation des univers prédicatifs suit un schéma très analogue au cas du modèle ensembliste booléen. On considère une suite de grands ensembles emboîtés $(\mathcal{U}_i)_{i>0}$ dont chacun des éléments est à lui-seul un modèle de la théorie des ensembles, et on note ω-$\mathbf{Set}_i$ $(i > 0)$ l'ensemble de tous les ω-sets dont le support est un élément de $\mathcal{U}_i$.

Comme par définition, chacun des ensembles $\mathcal{U}_i$ est stable par toutes les constructions définissables dans ZFC (et en particulier par formation de produit ensembliste arbitraire), le sous-ensemble ω-$\mathbf{Set}_i \subset \mathcal{U}_i$ est lui-même stable par formation de produit dépendant au sens des ω-sets. Intuitivement, ω-$\mathbf{Set}_i$ constitue une petite sous-catégorie cartésienne fermée de la grande catégorie des ω-sets, obtenue par restriction de cette dernière aux objets vivant dans l'univers $\mathcal{U}_i$.

Pour que l'interprétation soit complète, il faut également munir chacun des ensembles ω-$\mathbf{Set}_i$ d'une structure de ω-set (puisque ces ensembles sont destinés à interpréter les sortes, qui

sont aussi des types). Pour cela, on leur donne la structure de ω-set *grossier*, définie par la relation de réalisabilité

$$n \Vdash_{\omega\text{-}\mathbf{Set}_i} X \quad \equiv \quad \top \qquad \text{(relation pleine)}$$

dans laquelle n'importe quel entier réalise n'importe quel objet du support. On vérifie alors aisément que la suite $(\omega\text{-}\mathbf{Set}_i)_{i>0}$ est une suite de ω-sets emboîtés, puisque pour tout $i > 0$ on a

$$\omega\text{-}\mathbf{Set}_i \ \in \ |\omega\text{-}\mathbf{Set}_{i+1}| \qquad \text{et} \qquad |\omega\text{-}\mathbf{Set}_i| \ \subset \ |\omega\text{-}\mathbf{Set}_{i+1}|,$$

et que le support de chacun des ω-sets $\omega\text{-}\mathbf{Set}_i$ est lui-même un ensemble de ω-sets clos par formation de produit dépendant arbitraire.

On peut alors interpréter chacune des sortes Type_i par $\omega\text{-}\mathbf{Set}_i$ (dont le support est une petite sous-catégorie de ω-**Set**), et les sortes Prop et Set par l'ensemble **PER** des relations d'équivalences partielles (cet ensemble étant lui aussi muni d'une structure de ω-set grossier). Remarquons que dans le modèle intuitionniste, il n'est plus possible d'interpréter les types habitant la sorte Type_1 par des ensembles finis, puisque ce modèle valide le schéma d'élimination forte [63], lequel permet de montrer (dans le formalisme) que le type des entiers de Church dans Prop est infini.

Pour terminer cette présentation, il est important de noter que, contrairement au modèle ensembliste introduit dans le paragraphe précédent, le modèle basé sur les ω-sets invalide le type propositionnel $\Pi A : \mathsf{Prop} \,.\, A \vee \neg A$ (tiers-exclu), dont la dénotation est le PER vide.

3.2 Les limites des modèles purement ensemblistes

3.2.1 Church versus Curry

Dans les deux modèles de CCω que nous venons de présenter — que ce soit le modèle ensembliste booléen ou le modèle construit dans la catégorie des ω-sets — la fonction d'interprétation repose très fortement sur la présence de l'annotation de type $x : T$ pour définir la dénotation des termes de la forme $\lambda x : T \,.\, M$:

$$[\![\lambda x : T \,.\, M]\!]_\rho \ = \ \big(a_{\in [\![T]\!]_\rho} \mapsto [\![M]\!]_{\rho; x \leftarrow a}\big).$$

(Nous anticipons ici sur la section suivante, dans laquelle nous définirons la notion de valuation ainsi que les notations correspondantes.) Cette annotation de type joue ici un rôle crucial, car c'est précisément elle qui permet de déterminer le domaine de la fonction par laquelle on interprète le terme $\lambda x : T \,.\, M$.[11] Pour cette raison, les deux modèles de CCω introduits dans la section précédente sont fondamentalement des modèles de théories des types à la Church, et le schéma d'interprétation (non-typée) sur lequel ils reposent ne peut à l'évidence pas être repris tel quel pour un formalisme à la Curry tel que le calcul implicite.

[11] Rappelons qu'en théorie des ensembles, une fonction f est codée par son graphe, c'est-à-dire par l'ensemble des couples $(x, f(x))$ où x décrit le domaine de f. Bien entendu, cette définition n'a un sens que dans la mesure où le domaine de f est connu, et est lui-même un ensemble.

3.2.2 Quelle interprétation pour le sous-typage ?

En mettant (provisoirement) de côté le problème posé par l'absence d'annotation de type dans l'abstraction, il est important de s'interroger sur la signification qu'on pourrait donner au sous-typage du calcul implicite dans le cadre d'une interprétation ensembliste.[12]

Au chapitre 2, nous avons proposé une interprétation informelle des différentes constructions syntaxiques du calcul implicite (figure 2.2) dans laquelle nous avons présenté le produit implicite $\forall x : T \,.\, U$ comme l'intersection de la famille de types U_x lorsque x décrit T. Dans le cadre de ce schéma informel, nous aurions pu également présenter la relation de sous-typage $T \leq T'$ comme une simple inclusion de types $T \subset T'$.

Malheureusement, il semble difficilement envisageable d'interpréter le sous-typage par la relation d'inclusion entre ensembles. En effet, une telle interprétation ne serait pas en mesure de rendre compte de l'expressivité de cette relation dans le calcul implicite, et cela pour plusieurs raisons.

Interprétation de la contravariance Le premier problème vient de l'interprétation de la règle de contravariance du sous-typage par rapport au domaine des types flèches

$$\frac{A' \;\leq\; A}{A \to B \;\leq\; A' \to B}$$

Dans le calcul implicite, la règle ci-dessus est valide, mais elle ne l'est plus en théorie des ensembles lorsqu'on interprète le sous-typage par la relation d'inclusion. Là encore, le problème est lié à la présence d'un domaine de définition : si f est une fonction de A dans B, et si A' un sous-ensemble de A, f n'est en général pas une application de A' dans B, puisque son domaine $\mathrm{Dom}(f)$ est égal à A mais pas à A' (sauf si $A = A'$). Bien entendu, il existe une *coercition* de l'ensemble des applications de A dans B dans l'ensemble des applications de A' dans B, qui est donnée par l'opération de restriction de domaine :

$$f \;\in\; B^A \quad \Rightarrow \quad f_{|A'} \;\in\; B^{A'}.$$

(où $f_{|A'}$ est la fonction de A' dans B définie pour tout $x \in A'$ par $f_{|A'}(x) = f(x)$). Cependant, cette coercition nous fait sortir du cadre dans lequel le sous-typage est interprété par la relation d'inclusion.

Sous-typage et univers Dans les deux modèles de CCω décrits dans la section précédente, l'interprétation de chaque univers est caractérisée par une certaine "limite de cardinalité" (voir à ce sujet le paragraphe 3.4.5). La hiérarchie d'univers prédicatifs est interprétée par une suite d'ensembles de plus en plus grands, dont chaque élément est un ensemble stable par "toutes les opérations de la théorie des ensembles".[13] Dans ce schéma d'interprétation, il est clair que

[12] La notion d'"interprétation ensembliste" n'est pas (seulement) caractérisée par le fait que le modèle sous-jacent est défini dans la théorie des ensembles, mais par le fait que 1. la relation de typage est interprétée par la relation d'appartenance — ce qui signifie qu'un type est interprété par son contenu extensionnel — et 2. que les fonctions de la théorie des types sont interprétées par les fonctions de la théorie des ensembles. Bien entendu, les différents modèles que nous construirons aux chapitres 5, 6 et 7 seront construits au sein de la théorie des ensembles (et de ZFC plus particulièrement), ce qui ne fera pas de ces modèles des modèles ensemblistes pour autant.

[13] C'est-à-dire un modèle extensionnel de ZF.

tout sous-ensemble d'un élément d'un univers donné vit dans le même univers, voire dans un univers situé plus bas dans la hiérarchie. Cette propriété n'est par ailleurs rien d'autre que le versant sémantique d'un résultat syntaxique classique [42] de la métathéorie de CCω, qui est que pour tous types T et T' bien formés dans un contexte donné et tels que $T \preceq T'$, la sorte principale de T est inférieure ou égale à la sorte principale de T'.

Dans le calcul implicite, ceci n'est plus vrai en raison de la contravariance du sous-typage vis-à-vis des domaines des produits implicites. Pour cela, il suffit de considérer le jugement de sous-typage (dérivable dans le calcul implicite)

$$[] \quad \vdash \quad \underbrace{\forall T : \mathsf{Type}_2 \,.\, T \to T}_{: \mathsf{Type}_3} \quad \leq \quad \underbrace{\forall T : \mathsf{Type}_1 \,.\, T \to T}_{: \mathsf{Type}_2}$$

qui exprime que dans le contexte vide, le type $\forall T : \mathsf{Type}_2 \,.\, T \to T$ est un sous-type de $\forall T : \mathsf{Type}_1 \,.\, T \to T$. Pourtant, la sorte principale du membre gauche (Type_3) est plus élevée dans la hiérarchie d'univers que la sorte principale du membre droit (Type_2).[14] Autrement dit, dans le calcul implicite, un type A peut être un sous-type d'un type A' tout en vivant dans un univers plus élevé. On voit donc que le sous-typage du calcul implicite s'accommode mal avec le schéma standard sur lequel est basée l'interprétation des univers prédicatifs.

3.2.3 Une interprétation typée ou non-typée

Le modèle ensembliste booléen de CCω comme le modèle basé sur les ω-sets reposent tous les deux sur le principe d'une interprétation non-typée, puisque la fonction d'interprétation construit la dénotation de n'importe quel terme (éventuellement mal typé) en dehors de tout contexte de typage.[15] Les interprétations non-typées sont en général plus simples à définir et à manipuler, car leur définition s'effectue par récurrence sur la structure des termes et non par récurrence sur la structure des dérivations. La propriété de validité n'est pas donnée *a priori* par la définition, mais est démontrée *a posteriori* par une récurrence sur la structure de la dérivation de typage.

Cependant, nous avons vu au paragraphe 3.2.1 qu'une telle approche n'est pas possible pour le calcul implicite (dans le cadre d'une interprétation ensembliste). Par ailleurs, les exemples du paragraphe 3.2.2 montrent que la relation d'inclusion ne saurait interpréter de manière satisfaisante les propriétés du sous-typage.

Pour faire face à ces différents problèmes, on pourrait envisager une toute autre approche, consistant à "expliciter les constructions implicites". Dans cette perspective, le produit implicite serait interprété non plus par l'intersection d'une famille d'ensembles, mais par un espace de fonctions cachées. De la même manière, la relation de sous-typage ne serait plus interprétée par la relation d'inclusion, mais par l'existence d'une coercition cachée. D'une interprétation

[14] La sorte principale d'un type prédicatif T (défini dans un contexte Γ donné) est le plus petit univers Type_i tel que le jugement $\Gamma \vdash T : \mathsf{Type}_i$ soit dérivable.

[15] Pour qu'une telle interprétation soit réellement indépendante du typage, il faut évidemment prendre en compte le cas (issu de l'interprétation d'un terme mal typé) dans lequel une fonction est appliquée à un objet en dehors de son domaine, en retournant par exemple une valeur arbitraire destinée à représenter une telle "erreur d'exécution". On pourra également remarquer que dans le modèle ensembliste booléen par exemple, de nombreux termes mal typés ont une interprétation bien définie en raison de l'égalité des preuves dans le modèle.

non-typée on passerait alors à une interprétation typée, dans laquelle nous n'interpréterions plus des termes, mais des jugements de typage, et plus précisément leurs dérivations.[16]

Là encore, une telle approche semble vouée à l'échec, pour des raisons que nous avons déjà évoquées au chapitre 1. En effet, toute tentative d'explicitation des constructions implicites reviendrait peu ou prou à reconstruire les abstractions et applications implicites manquantes, c'est-à-dire à relever les dérivations de typage du calcul implicite dans le calcul bicolore (*i.e.* la version "explicite" du calcul implicite que nous avons présentée au chapitre 1, dans laquelle les abstractions et applications implicites sont représentées par des constructions syntaxiques spécifiques). Cependant, nous avons vu au chapitre 1 qu'un tel relèvement des dérivations n'existe pas.

Rappelons en effet que dans les *Type Assignment Systems* [27, 60] — qui sont tous des sous-systèmes du calcul implicite — la fonction d'effacement n'est pas surjective dans les systèmes du cube avec types dépendants (*cf* paragraphe 1.4.2) et ne réalise donc plus un isomorphisme entre les formalismes à la Church et les formalismes à la Curry qui se correspondent dans les deux cubes.

Intuitivement, ceci vient du fait que la fonction d'effacement peut identifier deux termes M_1 et M_2 originellement non-convertibles, c'est-à-dire $|M_1| =_\beta |M_2|$ bien que $M_1 \neq_\beta M_2$. Par le jeu des dépendances, il est possible de construire des types T_1 et T_2 ayant la même propriété, c'est-à-dire $|T_1| =_\beta |T_2|$ bien que $T_1 \neq_\beta T_2$. Par conséquent, la règle de β-conversion des TAS avec types dépendants est plus "permissive" et permet de dériver des jugements qui ne proviennent d'aucun jugement dérivable dans le système correspondant au sein du cube de Barendregt. Bien entendu, le même problème se pose dans le calcul implicite, et il n'est en général pas possible de relever un jugement du calcul implicite vers le calcul bicolore.[17]

Dans le cadre d'une interprétation typée du calcul implicite, nous serions confrontés au même problème. Techniquement, nous serions à un certain moment amenés à interpréter deux types convertibles T_1 et T_2 (dépendant de termes dont les reconstructions sont essentiellement différentes) par deux dénotations différentes $t_1 \neq t_2$, et échouerions à interpréter la règle de conversion dans le modèle.[18]

3.3 Modèles abstraits du Calcul des Constructions implicite

Dans la section précédente, nous avons présenté plusieurs raisons pour lesquelles un cadre strictement ensembliste[19] n'est pas adapté à l'interprétation du calcul implicite. De fait, la structure du calcul implicite (abstraction à la Curry, sous-typage, ambiguïté typique très forte)

[16] Les interprétations typées sont en général plus difficiles à définir que les interprétations non-typées, puisque la preuve de la correction de la fonction d'interprétation doit être effectuée conjointement avec sa définition.

[17] Cette situation est même aggravée dans le calcul implicite par le fait que le produit implicite peut être utilisé à n'importe quel niveau de la hiérarchie d'univers.

[18] Bien entendu, il serait sans doute possible de contourner cette difficulté en introduisant dans le modèle une relation d'équivalence identifiant les objets modulo leur partie implicite, et en ne considérant que des fonctions (explicites ou implicites) compatibles avec cette relation d'équivalence. Les quelques tentatives que nous avons effectuées dans ce sens se sont soldées par des échecs, la relation d'équivalence en question s'avérant redoutablement difficile à définir. Par ailleurs, il n'est pas sûr qu'un modèle basé sur ce principe soit très lisible, et la solution que nous présenterons au chapitre 5 nous semble bien plus satisfaisante à cet égard.

[19] *cf* la note 12 de la page 115.

semble bien plus proche des langages de programmation fonctionnels[20] que des théories des types traditionnelles. Avant d'aborder la question des modèles du calcul implicite en théorie des domaines, nous allons rappeler un ensemble de résultats généraux concernant la théorie des modèles du λ-calcul [8]. Ceci nous permettra d'introduire une notion abstraite de modèle du calcul implicite et de factoriser les nombreuses propriétés communes à tous les représentants de cette famille de modèles.

3.3.1 λ-modèles

Définition 3.3.1 (Valuation) — Soit $\mathcal{M}$ un ensemble.

- Une *valuation sur* $\mathcal{M}$ est une application $\rho : \mathcal{V} \to \mathcal{M}$ associant un objet $\rho(x) \in \mathcal{M}$ à toute variable $x \in \mathcal{V}$. L'ensemble de toutes les valuations sur $\mathcal{M}$ est noté $\mathbf{Val}_{\mathcal{M}}$.
- Si ρ est une valuation sur $\mathcal{M}$, x une variable et a un élément de $\mathcal{M}$, on note $\rho; x \leftarrow a$ la valuation construite en associant a à x dans ρ (effaçant ainsi la valeur précédemment associée à x).

Définition 3.3.2 (λ-modèle) — Un *λ-modèle* est un triplet $(\mathcal{M}, \cdot, [\![]\!])$ tel que

- $\mathcal{M}$ est un ensemble ayant au moins un élément ;
- $(\cdot) : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ est une opération binaire sur $\mathcal{M}$;
- $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$ est une application associant un objet $[\![M]\!]_\rho \in \mathcal{M}$ à n'importe quel terme $M \in \Lambda$ et à n'importe quelle valuation $\rho \in \mathbf{Val}_{\mathcal{M}}$, qui satisfait les axiomes suivants :
  1. $[\![x]\!]_\rho \;=\; \rho(x)$;
  2. $[\![M\ N]\!]_\rho \;=\; [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
  3. $[\![\lambda x\,.\,M]\!]_\rho \cdot a \;=\; [\![M]\!]_{\rho;x\leftarrow a}$
  4. si $[\![M]\!]_{\rho;x\leftarrow a} = [\![M']\!]_{\rho';x\leftarrow a}$ pour tout $a \in \mathcal{M}$, alors $[\![\lambda x\,.\,M]\!]_\rho = [\![\lambda x\,.\,M']\!]_{\rho'}$.

Par la suite, l'opération $a \cdot b$ sera écrite à la manière d'une application ab. Par exemple, nous écrirons $abc(de)$ pour $((a \cdot b) \cdot c) \cdot (d \cdot e)$. L'axiome 4 porte généralement le nom d'*axiome d'extensionnalité faible* (cet axiome est en effet crucial pour valider la règle ξ). Remarquons que cet axiome n'entraîne pas que la structure applicative $(\mathcal{M}, \cdot)$ est extensionnelle, c'est-à-dire :

$$(\forall x \in \mathcal{M} \quad ax = bx) \quad \Rightarrow \quad a = b \qquad \text{(extensionnalité)}$$

(Bien entendu, l'axiome d'extensionnalité implique trivialement l'axiome 4.)

Remarque 3.3.3 — Notons que notre définition des λ-modèles diffère sensiblement de celle qui est formulée dans [8]. Le lecteur pourra cependant vérifier que si $(\mathcal{M}, [\![]\!], \cdot)$ est un λ-modèle au sens de la définition précédente, l'algèbre combinatoire $(\mathcal{M}, \cdot, \mathbf{K}, \mathbf{S})$ constitue un λ-modèle au sens de [8], où $\mathbf{K}$ et $\mathbf{S}$ sont les éléments de $\mathcal{M}$ donnés par

$$\mathbf{K} = [\![\lambda xy\,.\,x]\!]_\rho \qquad \text{et} \qquad \mathbf{S} = [\![\lambda xyz\,.\,xz(yz)]\!]_\rho \qquad (\rho \text{ arbitraire})$$

Réciproquement, on peut définir dans tout λ-modèle au sens de [8] une fonction d'interprétation $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$ qui satisfait les axiomes de la définition précédente. (On vérifiera par ailleurs que ces deux correspondances sont réciproques l'une de l'autre.)

[20] Notons à ce propos que le noyau purement fonctionnel du langage ML se plonge trivialement dans la partie *prédicative* du calcul implicite, les types étant interprétés dans Type_1 et les schémas de types dans Type_2. Bien entendu, la quantification de type prénexe de ML est ici exprimée par un produit implicite.

Proposition 3.3.4 (Propriétés des λ-modèles) — *Si $(\mathcal{M}, \cdot, [\![]\!])$ est un λ-modèle, alors :*

1. *si $\rho(x) = \rho'(x)$ pour tout $x \in FV(M)$, alors $[\![M]\!]_\rho = [\![M]\!]_{\rho'}$*
2. *$[\![M\{x := N\}]\!]_\rho = [\![M]\!]_{\rho; x \leftarrow [\![N]\!]_\rho}$*
3. *si $M =_\beta M'$, alors $[\![M]\!]_\rho = [\![M']\!]_\rho$.*

Preuve. Les points 1 et 2 se prouvent par récurrence sur M, en recourant à l'axiome d'extensionnalité faible dans le cas où M est une λ-abstraction. Le point 3 est d'abord établi dans le cas particulier où $M \to_\beta M'$ à l'aide de la même technique de preuve que pour les points 1 et 2, et est ensuite étendu au cas général par une récurrence immédiate. □

D'après le premier point de la proposition ci-dessus, il est clair que la dénotation d'un terme M ne dépend pas de la valuation ρ dans le cas où M est clos. Dans ce cas particulier, nous écrirons donc plus simplement $[\![M]\!] = [\![M]\!]_\rho$ (avec ρ arbitraire).

λ-modèles triviaux et non-triviaux Un λ-modèle $(\mathcal{M}, \cdot, [\![]\!])$ est *trivial* si l'ensemble $\mathcal{M}$ est réduit à un singleton. À isomporphisme près, il n'existe qu'un seul λ-modèle trivial, qui est le triplet $(\mathcal{M}, \cdot, [\![]\!])$ défini par

1. $\mathcal{M} = \{\bullet\}$
2. $\bullet \cdot \bullet = \bullet$
3. $[\![M]\!]_\rho = \bullet$ pour tout $M \in \Lambda$ et pour l'unique valuation $\rho = (x_{\in \mathcal{V}} \mapsto \bullet)$

Le λ-modèle trivial $\mathcal{M} = \{\bullet\}$ (qui est un λ-modèle extensionnel) n'est pas très intéressant puisqu'il interprète tous les termes du λ-calcul par le même objet. En revanche, la structure d'un λ-modèle non-trivial est nécessairement très riche d'après la proposition suivante :

Proposition 3.3.5 (λ-modèle non-trivial) — *Si $(\mathcal{M}, \cdot, [\![]\!])$ est un λ-modèle non-trivial, alors*

1. *$\mathcal{M}$ est un ensemble infini*
2. *L'opération $(\cdot) : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ est non commutative et non-associative.*

(On trouvera une preuve de cette proposition dans [8].)

3.3.2 λ-modèles sous-extensionnels

La notion de λ-modèle a l'inconvénient de ne fournir en général aucun moyen de contrôle de la règle de η-conversion à travers la fonction d'interprétation (à moins bien entendu que le λ-modèle en question ne soit extensionnel). Cependant, les modèles explicites que nous construirons aux chapitres 5, 6 et 7 ne seront pas extensionnels. Afin de capturer la régularité de leur comportement vis-à-vis de la règle η, il est donc nécessaire d'introduire une notion plus faible que la notion de λ-modèle extensionnel, qui est la notion de *λ-modèle sous-extensionnel*.

Définition 3.3.6 (λ-modèle sous-extensionnel) — Un *λ-modèle sous-extensionnel* est un quadruplet $(\mathcal{M}, \cdot, [\![]\!], \leq)$ constitué d'un λ-modèle $(\mathcal{M}, \cdot, [\![]\!])$ muni d'une relation d'ordre $(\leq) \subset \mathcal{M} \times \mathcal{M}$ satisfaisant les axiomes suivants :

1. si $a \leq a'$ et $b \leq b'$, alors $ab \leq a'b'$
2. $[\![\lambda f x \,.\, f\ x]\!] \leq [\![\lambda x \,.\, x]\!]$ (sous-extensionnalité)

L'axiome 1 exprime la monotonie de l'application. Une conséquence immédiate de cet axiome est l'implication

$$a \leq b \quad \Rightarrow \quad (\forall\, c \in \mathcal{M} \quad ac \leq bc),$$

dont la réciproque est fausse en général. (Un contre exemple est donné par l'ordre stable, qui sera introduit au chapitre 4.) L'axiome 2 — l'axiome de sous-extensionnalité — est notre moyen de contrôle de la règle η. Sa conséquence la plus importante est la suivante :

Proposition 3.3.7 (Sous-extensionnalité) — *Soit $(\mathcal{M}, \cdot, [\![]\!], \leq)$ un λ-modèle sous-extensionnel. Pour tous termes M, M' et pour toute valuation ρ, on a :*

1. *si $M \twoheadrightarrow_\eta M'$, alors $[\![M]\!]_\rho \leq [\![M']\!]_\rho$*
2. *si $M =_{\beta\eta} M'$, alors il existe un objet $c \in \mathcal{M}$ tel que $[\![M]\!]_\rho \leq c$ et $[\![M']\!]_\rho \leq c$*

Preuve. Pour établir le premier point, il suffit de considérer le cas particulier dans lequel $M \to_\eta M'$ (correspondant à une seule étape de η-réduction), le cas général en découlant directement par une récurrence immédiate. Supposons donc que $M \to_\eta M'$. Il existe alors un terme N ainsi qu'un contexte $\mathcal{C}[]$ à un trou tel que $M \equiv \mathcal{C}[\lambda x\,.\,N\ x]$ et $M' \equiv \mathcal{C}[N]$ (avec $x \notin FV(N)$). Soit z une variable n'apparaissant ni libre ni liée dans M ou M'. Nous avons

$$\begin{array}{rcccccl} M & \equiv & \mathcal{C}[\lambda x\,.\,N\ x] & =_\beta & \mathcal{C}[(\lambda f x\,.\,f\ x)\ N] & =_\beta & (\lambda z\,.\,\mathcal{C}[z\ N])\ (\lambda f x\,.\,f\ x) \\ M' & \equiv & \mathcal{C}[N] & =_\beta & \mathcal{C}[(\lambda x\,.\,x)\ N] & =_\beta & (\lambda z\,.\,\mathcal{C}[z\ N])\ (\lambda x\,.\,x). \end{array}$$

À l'aide de ces deux β-équivalences ainsi que la proposition 3.3.4, il vient

$$[\![M]\!]_\rho \quad = \quad [\![\lambda z\,.\,\mathcal{C}[z\ N]]\!]_\rho \cdot [\![\lambda f x\,.\,f\ x]\!] \quad \leq \quad [\![\lambda z\,.\,\mathcal{C}[z\ N]]\!]_\rho \cdot [\![\lambda x\,.\,x]\!] \quad = \quad [\![M']\!]_\rho$$

grâce à l'axiome de sous-extensionnalité et la monotonie de l'application. Le second point est alors une conséquence immédiate du premier point et de la propriété de Church-Rosser concernant la $\beta\eta$-réduction. □

Remarque 3.3.8 N'importe quel λ-modèle extensionnel $(\mathcal{M}, \cdot, [\![]\!])$ est (implicitement) un λ-modèle sous-extensionnel. Pour lui en donner la structure complète, il suffit simplement de le munir de la relation d'ordre triviale donnée par l'égalité dénotationnelle.

Définition 3.3.9 (Ordre sur les valuations) — Soit $(\mathcal{M}, \cdot, [\![]\!], \leq)$ un λ-modèle sous-extensionnel. La relation d'ordre $(\leq)$ sur $\mathcal{M}$ s'étend naturellement aux valuations en posant :

$$\rho \leq \rho' \quad \equiv \quad (\forall\, x \in \mathcal{V} \quad \rho(x) \leq \rho'(x)).$$

Lemme 3.3.10 (Monotonie de l'interprétation) — *Dans tout λ-modèle sous-extensionnel $(\mathcal{M}, \cdot, [\![]\!], \leq)$, la fonction d'interprétation $[\![M]\!]_\rho$ est monotone par rapport à la valuation ρ, c'est-à-dire :*

$$\rho \leq \rho' \quad \Rightarrow \quad [\![M]\!]_\rho \leq [\![M]\!]_{\rho'}.$$

Preuve. Il suffit de remarquer que pour tout terme M dont les variables libres sont notées $x_1, \ldots, x_n$ et pour toute valuation ρ on a :

$$[\![M]\!]_\rho \quad = \quad [\![(\lambda x_1 \cdots x_n\,.\,M) x_1 \cdots x_n]\!]_\rho \quad = \quad [\![\lambda x_1 \cdots x_n\,.\,M]\!] \rho(x_1) \cdots \rho(x_n). \qquad \square$$

3.3.3 Modèles abstraits du calcul implicite

Définition 3.3.11 (Modèle abstrait du calcul implicite) — On appelle *modèle abstrait du calcul implicite* toute structure de la forme

$$(\mathcal{M}, \cdot, [\![]\!], \leq, \mathcal{T}, \mathsf{El}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega})$$

où

- $(\mathcal{M}, \cdot, [\![]\!], \leq)$ est un λ-modèle sous-extensionnel, et $\mathcal{T}$ un sous-ensemble de $\mathcal{M}$
- El est une application de $\mathcal{T}$ dans $\mathfrak{P}(\mathcal{M})$
- Π et $\forall$ sont deux éléments de $\mathcal{M}$, et $(\mathsf{u}_i)_{i\in I}$ une suite d'éléments de $\mathcal{T}$

et qui satisfait les axiomes 1 à 10 donnés dans la figure 3.2 page 122.

Intuitivement, un modèle abstrait[21] du calcul implicite est un λ-modèle sous-extensionnel muni des structures additionnelles suivantes :

- Un sous-ensemble $\mathcal{T} \subset \mathcal{M}$ dont les éléments sont les types du modèle, ou plus précisément les *codes* (ou *noms*) des types dans le modèle $\mathcal{M}$, en reprenant la terminologie introduite par P. Martin-Löf.
- Une fonction de *décodage* $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{M})$ associant à chaque code de type $t \in \mathcal{T}$ son contenu extensionnel $\mathsf{El}(t) \subset \mathcal{M}$, c'est-à-dire l'ensemble des objets du modèle qui ont pour type le type t.
- Des constantes Π et $\forall$ représentant les opérateurs de produits dépendants explicite et implicite respectivement, lesquels opérent sur les codes de types pour retourner de nouveaux codes de types.
- Une suite de constantes $(\mathsf{u}_i)_{i\in\omega}$ représentant les codes des univers Prop, Set, Type_1, Type_2, etc. Notons qu'ici, la même constante u_0 représente le code des sortes Prop et Set, lesquelles sont isomorphes du point de vue des règles de typage dans le calcul implicite.

Ces paramètres satisfont en outre un jeu de 10 axiomes dont la signification est la suivante :

- Les axiomes 1 à 4 donnent les propriétés principales du sous-ensemble $\mathcal{T} \subset \mathcal{M}$ et de la fonction de décodage $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{M})$. L'axiome 1 exprime que $\mathcal{T}$ est clos supérieurement dans $\mathcal{M}$ pour la relation d'ordre $\leq$. L'axiome 2 exprime que la fonction El est monotone en ce sens très particulier que si t et t' sont deux codes de types comparables, alors t et t' ont exactement les mêmes éléments (puisque $\mathsf{El}(t) = \mathsf{El}(t')$). Finalement, l'axiome 3 exprime que pour tout code de type $t \in \mathcal{T}$, l'ensemble des éléments de t est également un sous-ensemble clos supérieurement dans $\mathcal{M}$, à l'instar de l'ensemble $\mathcal{T}$ lui-même.
- Les axiomes 4 et 5 donnent la signification des constantes Π et $\forall$. Ils expriment que lorsque Πtu (ou $\forall tu$) est un code de type, alors t est un code de type et u une famille de codes de types indicée par les éléments de t. De plus, ces axiomes caractérisent les contenus de $\mathsf{El}(\Pi tu)$ et $\mathsf{El}(\forall tu)$ en fonction des contenus de $\mathsf{El}(t)$ et de $\mathsf{El}(ua)$ lorsque a décrit l'ensemble $\mathsf{El}(t)$. Notons que les axiomes 4 et 5 ne donnent la signification de Πtu et de $\forall tu$ que lorsque ces expressions désignent des codes de types, mais n'indiquent

[21] L'idée de définir une caractérisation abstraite des modèles du calcul implicite de manière à factoriser les propriétés communes aux modèles que nous présenterons dans les chapitres 5, 6 et 7 nous a été suggérée par T. Coquand et D. Friedlender [24].

Paramètres des modèles abstraits du calcul implicite

- un ensemble $\mathcal{M}$
- une opération binaire $(\cdot) : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
- une application $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$
- une relation d'ordre $(\leq) \subset \mathcal{M} \times \mathcal{M}$
- un sous-ensemble $\mathcal{T} \subset \mathcal{M}$
- une application $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{M})$
- deux constantes $\Pi, \forall \in \mathcal{M}$
- une suite de constantes $(\mathsf{u}_i)_{i\in\omega} \in \mathcal{T}^\omega$

Axiomes des λ-modèles sous-extensionnels

a. si $a \leq a'$ et $b \leq b'$, alors $a \cdot b \leq a' \cdot b'$
b. $[\![x]\!]_\rho = \rho(x)$
c. $[\![M\ N]\!]_\rho = [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
d. $[\![\lambda x\,.\,M]\!]_\rho \cdot a = [\![M]\!]_{\rho;x\leftarrow a}$
e. si $[\![M]\!]_{\rho;x\leftarrow a} = [\![M']\!]_{\rho';x\leftarrow a}$ pour tout $a \in \mathcal{M}$, alors $[\![\lambda x\,.\,M]\!]_\rho = [\![\lambda x\,.\,M']\!]_{\rho'}$
f. $[\![\lambda f x\,.\,f\ x]\!] \ \leq\ [\![\lambda x\,.\,x]\!]$

Axiomes de $\mathcal{T}$ et El

1. si $t \in \mathcal{T}$ et $t \leq t'$, alors $t' \in \mathcal{T}$
2. si $t \in \mathcal{T}$ et $t \leq t'$, alors $\mathsf{El}(t) = \mathsf{El}(t')$
3. si $t \in \mathcal{T}$, $a \in \mathsf{El}(t)$ et $a \leq a'$, alors $a' \in \mathsf{El}(t)$

Axiomes de Π et $\forall$

4. si $\Pi t u \in \mathcal{T}$, alors
   - $t \in \mathcal{T}$ et $ua \in \mathcal{T}$ pour tout $a \in \mathsf{El}(t)$
   - $\mathsf{El}(\Pi t u) = \{f \in \mathcal{M};\quad \forall\, a \in \mathsf{El}(t)\quad fa \in \mathsf{El}(ua)\}$
5. si $\forall t u \in \mathcal{T}$, alors
   - $t \in \mathcal{T}$ et $ua \in \mathcal{T}$ pour tout $a \in \mathsf{El}(t)$
   - $\mathsf{El}(\forall t u) = \{b \in \mathcal{M};\quad \forall\, a \in \mathsf{El}(t)\quad b \in \mathsf{El}(ua)\}$

Axiomes des univers

6. $\mathsf{El}(\mathsf{u}_i) \subset \mathcal{T}$
7. $\mathsf{u}_i \in \mathsf{El}(\mathsf{u}_{i+1})$
8. $\mathsf{El}(\mathsf{u}_i) \subset \mathsf{El}(\mathsf{u}_{i+1})$
9. si $t \in \mathsf{El}(\mathsf{u}_i)$ et $ua \in \mathsf{El}(\mathsf{u}_i)$ pour tout $a \in \mathsf{El}(t)$, alors $\Pi t u \in \mathsf{El}(\mathsf{u}_i)$ et $\forall t u \in \mathsf{El}(\mathsf{u}_i)$
10. si $t \in \mathsf{El}(\mathsf{u}_i)$ et $ua \in \mathsf{El}(\mathsf{u}_0)$ pour tout $a \in \mathsf{El}(t)$, alors $\Pi t u \in \mathsf{El}(\mathsf{u}_0)$ et $\forall t u \in \mathsf{El}(\mathsf{u}_0)$

FIG. 3.2 – Paramètres et axiomes des modèles abstraits du calcul implicite

aucun critère pour qu'une telle condition soit satisfaite (de tels critères sont donnés par les axiomes 9 et 10).

- Les axiomes 6-10 révèlent la structure du système de types purs implicite sous-jacent. L'axiome 6 exprime que chacune des constantes u_i désigne une sorte, car son contenu $\mathsf{El}(\mathsf{u}_i)$ est constitué d'un ensemble de codes de types. L'axiome 7 exprime les axiomes de la hiérarchie d'univers ($\mathsf{Prop} : \mathsf{Type}_1$ et $\mathsf{Type}_i : \mathsf{Type}_{i+1}$) tandis que l'axiome 8 exprime la relation de cumulativité. Finalement, l'axiome 9 exprime que le contenu de chaque sorte u_i est clos par formation de produits dépendants explicites et implicites (formés à l'aide des operateurs Π et $\forall$) tandis que l'axiome 10 exprime l'imprédicativité de la sorte u_0 (représentant à la fois les sortes syntaxiques Prop et Set).

Remarquons que le modèle trivial du λ-calcul $\mathcal{M} = \{\bullet\}$ constitue un modèle abstrait du calcul implicite, en posant :

1. $\mathcal{T} = \mathcal{M} = \{\bullet\}$
2. $\mathsf{El}(\bullet) = \mathcal{M} = \mathcal{T} = \{\bullet\}$
3. $\Pi = \forall = \mathsf{u}_i = \bullet$

Cet exemple montre que, telle que nous venons de la définir, la notion de modèle abstrait du calcul implicite ne nous permet pas de prouver grand chose concernant les propriétés logiques du calcul implicite. En revanche, nous verrons plus loin que le simple ajout d'un axiome supplémentaire nous permettra de déduire de manière directe la cohérence logique du calcul implicite restreint indépendamment de tout résultat de normalisation.

Dans toute la suite de cette section, nous supposons que $(\mathcal{M}, \cdot, [\![]\!], \leq, \mathcal{T}, \mathsf{El}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega})$ désigne un modèle abstrait du calcul implicite quelconque.

3.3.4 Extension de l'interprétation au termes du calcul implicite

Par définition, le modèle abstrait $\mathcal{M}$ est muni d'une fonction $[\![]\!] : \mathbf{Val}_\mathcal{M} \times \Lambda \to \mathcal{M}$ permettant d'interpréter les termes du λ-calcul pur, et les termes du λ-calcul pur seulement. Cependant, la présence des constantes additionnelles Π, $\forall$ et u_i nous permettent d'étendre cette fonction d'interprétation à l'ensemble des termes du calcul implicite de la manière suivante :

Proposition 3.3.12 (Interprétation étendue) — *La fonction d'interprétation* $[\![]\!] : \Lambda \times \mathbf{Val}_\mathcal{M} \to \mathcal{M}$ *peut être étendue en une fonction d'interprétation* $[\![]\!] : \Lambda_{\mathrm{CCI}} \times \mathbf{Val}_\mathcal{M} \to \mathcal{M}$ *sur les termes du calcul implicite (l'interprétation étendue étant toujours notée* $[\![M]\!]_\rho$*) qui satisfait les propriétés suivantes :*

1. $[\![x]\!]_\rho \;=\; \rho(x)$;
2. $[\![\mathsf{Prop}]\!]_\rho \;=\; [\![\mathsf{Set}]\!]_\rho \;=\; \mathsf{u}_0$
3. $[\![\mathsf{Type}_i]\!]_\rho \;=\; \mathsf{u}_i \quad (i > 0)$
4. $[\![M\ N]\!]_\rho \;=\; [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
5. $[\![\lambda x\,.\,M]\!]_\rho \cdot a \;=\; [\![M]\!]_{\rho;x\leftarrow a}$
6. $[\![\Pi x : T\,.\,U]\!]_\rho \;=\; (\Pi \cdot [\![T]\!]_\rho) \cdot [\![\lambda x\,.\,U]\!]_\rho$
7. $[\![\forall x : T\,.\,U]\!]_\rho \;=\; (\forall \cdot [\![T]\!]_\rho) \cdot [\![\lambda x\,.\,U]\!]_\rho$
8. *si* $[\![M]\!]_{\rho;x\leftarrow a} = [\![M']\!]_{\rho';x\leftarrow a}$ *pour tout* $a \in \mathcal{M}$*, alors* $[\![\lambda x\,.\,M]\!]_\rho = [\![\lambda x\,.\,M']\!]_{\rho'}$.

9. $[\![\lambda f x\,.\,f\ x]\!]_\rho \ \leq\ [\![\lambda x\,.\,x]\!]_\rho$

Preuve. Il suffit de considérer le calcul implicite comme un simple λ-calcul muni de constantes additionnelles Prop, Set, Type_i, Π et $\forall$, et de considérer les constructions $\Pi x : T\,.\,U$ et $\forall x : T\,.\,U$ comme du "sucre syntaxique" pour

$$\Pi x : T\,.\,U \quad \equiv \quad \Pi\ T\ (\lambda x\,.\,U) \qquad \text{and} \qquad \forall x : T\,.\,U \quad \equiv \quad \forall\ T\ (\lambda x\,.\,U)$$

(ce que nous avons déjà fait au paragraphe 2.2.2 en introduisant la fonction de traduction $M \mapsto M^*$ et en étudiant ses propriétés). À l'aide de cette reformulation, il suffit alors d'interpréter les constantes Prop, Set, Type_i, Π et $\forall$ par leurs homologues sémantiques en posant $[\![\mathsf{Prop}]\!] = [\![\mathsf{Set}]\!] = \mathsf{u}_0$, $[\![\mathsf{Type}_i]\!] = \mathsf{u}_i$ $(i > 0)$, $[\![\Pi]\!] = \Pi$ and $[\![\forall]\!] = \forall$. Rappelons que cette formulation n'est pas complètement fidèle vis-à-vis de la $\beta\eta$-réduction : nous avons en effet montré au paragraphe 2.2.2 que la traduction $M \mapsto M^*$ est un plongement vis-à-vis de la β-réduction, mais seulement un morphisme vis-à-vis de la η-réduction. Cependant, seule la condition de "morphisme" (satisfaite par les règles β et η) est nécessaire pour prouver les points 1 à 9. □

Proposition 3.3.13 (Propriétés de l'interprétation étendue) — *L'interprétation étendue* $[\![]\!] : \Lambda_{\mathrm{CCI}} \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$ *satisfait les propriétés suivantes :*

1. *si* $\rho(x) = \rho'(x)$ *pour tout* $x \in FV(M)$, *alors* $[\![M]\!]_\rho \ =\ [\![M]\!]_{\rho'}$
2. *si* $\rho(x) \leq \rho'(x)$ *pour tout* $x \in FV(M)$, *alors* $[\![M]\!]_\rho \ \leq\ [\![M]\!]_{\rho'}$
3. $[\![M\{x := N\}]\!]_\rho \ =\ [\![M]\!]_{\rho;x\leftarrow[\![N]\!]_\rho}$
4. *si* $M \to_\beta M'$, *alors* $[\![M]\!]_\rho = [\![M']\!]_\rho$.
5. *si* $M \to_\eta M'$, *alors* $[\![M]\!]_\rho \leq [\![M']\!]_\rho$.
6. *si* $M =_{\beta\eta} M'$, *alors il existe un objet* $c \in \mathcal{M}$ *tel que* $[\![M]\!]_\rho \leq c$ *et* $[\![M']\!]_\rho \leq c$

Preuve. Les points 1 à 5 découlent directement des propriétés définitionnelles de l'interprétation étendue. Le dernier point est une conséquence immédiate des point 4, 5 et de la propriété de Church-Rosser pour la $\beta\eta$-réduction dans le calcul implicite. □

Le dernier point de la proposition ci-dessous a une conséquence importante, liée à l'interprétation de la règle de ($\beta\eta$-)conversion dans le calcul implicite :

Proposition 3.3.14 (Interprétation de la règle de $\beta\eta$-conversion) — *Soient* M *et* M' *deux termes du calcul implicite. Pour toute valuation* ρ *on a*

$$M =_{\beta\eta} M' \quad et \quad [\![M]\!]_\rho \in \mathcal{T} \quad et \quad [\![M']\!]_\rho \in \mathcal{T} \qquad \Rightarrow \qquad \mathsf{El}\big([\![M]\!]_\rho\big) \ =\ \mathsf{El}\big([\![M']\!]_\rho\big).$$

Preuve. Cette proposition est une conséquence du dernier point de la proposition précédente, et des axiomes 1 et 2 des modèles abstraits du calcul implicite. □

Cette proposition montre que les dénotations de deux termes $\beta\eta$-convertibles lorsqu'elles sont toutes les deux des types dans le modèle $\mathcal{M}$ ont exactement le même contenu extensionnel (c'est-à-dire les mêmes éléments par la fonction de décodage El), bien que les dénotations en question puissent être différentes.

Avant d'établir la validité de l'interprétation étendue vis-à-vis des règles de typage du calcul implicite restreint, nous devons d'abord étendre la fonction d'interprétation aux contextes.

Définition 3.3.15 (Interprétation des contextes) — Soit Γ un contexte. On désigne par $[\![\Gamma]\!]$ le sous-ensemble de $\mathbf{Val}_{\mathcal{M}}$ défini par

$$[\![\Gamma]\!] \quad = \quad \big\{\rho \in \mathbf{Val}_{\mathcal{M}};\quad \forall (x:T) \in \Gamma \quad [\![T]\!]_\rho \in \mathcal{T} \ \text{ et } \ \rho(x) \in \mathsf{El}([\![T]\!]_\rho)\big\}.$$

Lemme 3.3.16 (Interprétation de sous-contextes) — *Si* $\Gamma \subset \Gamma'$*, alors* $[\![\Gamma']\!] \subset [\![\Gamma]\!]$.

Preuve. Immédiat d'après la définition de $[\![\Gamma]\!]$. □

Proposition 3.3.17 (Validité restreinte) — *Si* $\Gamma \vdash_r M : T$ *est un jugement dérivable dans le calcul implicite restreint, alors on a pour toute valuation* $\rho \in \mathbf{Val}_{\mathcal{M}}$ *:*

$$\rho \in [\![\Gamma]\!] \qquad \Rightarrow \qquad [\![T]\!]_\rho \in \mathcal{T} \quad \textit{et} \quad [\![M]\!]_\rho \ \in \ \mathsf{El}\big([\![T]\!]_\rho\big).$$

Preuve. Par récurrence sur la dérivation du jugement $\Gamma \vdash_r M : T$ en distinguant les cas suivant la dernière règle utilisée. □

Remarquons que dans le cas général, l'interprétation étendue n'est valide que vis-à-vis des jugements du calcul implicite restreint. La raison en est que dans le modèle, la présence d'un type vide peut suffire à invalider la règle de renforcement. Pour comprendre ce phénomène, nous devons donc raffiner la notion de modèle du calcul implicite en introduisant les notions de *modèle restreint* et de *modèle non restreint*.

3.3.5 Modèles restreints et modèles non-restreints

Définition 3.3.18 (Modèles restreints et non-restreints) — Soit $i \in \omega$. Un modèle du calcul implicite $(\mathcal{M}, \cdot, [\![]\!], \leq, \mathcal{T}, \mathsf{El}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega})$ est dit :

- *i-restreint* s'il existe $t \in \mathsf{El}(\mathsf{u}_i)$ tel que $\mathsf{El}(t) = \varnothing$;
- *non-restreint* si pour tout $t \in \mathcal{T}$, $\mathsf{El}(t)$ a au moins un élément.

Les modèles i-restreints du calcul implicite invalident la règle de renforcement ainsi que le montre la proposition suivante :

Proposition 3.3.19 (Invalidation de la règle de renforcement) — *Pour tout* $i \in \omega$*, notons* $\mathsf{False}_i = \forall T : \mathsf{Type}_i \,.\, T$ *(avec* $\mathsf{Type}_0 \equiv \mathsf{Prop}$*). Si* $\mathcal{M}$ *est un modèle i-restreint du calcul implicite, alors*

1. $\mathsf{El}\big([\![\mathsf{False}_i]\!]\big) \ = \ \varnothing$ *;*
2. $\mathsf{El}\big([\![\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i]\!]\big) \ = \ \mathcal{M}$
3. $\mathsf{El}\big([\![(\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i) \to \mathsf{False}_i]\!]\big) \ = \ \varnothing$

Preuve. Les termes False_i, $\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i$ et $(\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i) \to \mathsf{False}_i$ sont des types bien formés du calcul implicite restreint dans le contexte vide. D'après la proposition 3.3.17, leurs dénotations (qui ne dépendent pas de la valuation) sont donc des types dans le modèle $\mathcal{M}$. D'après l'axiome 4, on a donc

$$\mathsf{El}([\![\mathsf{False}_i]\!]) \ = \ \mathsf{El}\big([\![\forall T : \mathsf{Type}_i \,.\, T]\!]\big) \ = \ \bigcap_{t \in \mathsf{El}(\mathsf{u}_i)} \mathsf{El}(t) \ = \ \varnothing$$

puisqu'il existe $t_0 \in \mathsf{El}(\mathsf{u}_i)$ tel que $\mathsf{El}(t_0) = \varnothing$ par hypothèse. De même, on a en vertu des axiomes 4 et 5

$$\mathsf{El}\big([\![\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i]\!]\big) \;=\; \{b \in \mathcal{M};\quad \forall\, a \in \varnothing \quad b \in \varnothing\} \;=\; \mathcal{M}$$

et

$$\mathsf{El}\big([\![(\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i) \to \mathsf{False}_i]\!]\big) \;=\; \{f \in \mathcal{M};\quad \forall\, a \in \mathcal{M} \quad fa \in \varnothing\} \;=\; \varnothing. \qquad \square$$

L'invalidation de la règle de renforcement vient de ce que la proposition

$$(\forall \_ : \mathsf{False}_i \,.\, \mathsf{False}_i) \to \mathsf{False}_i$$

est prouvable dans le calcul implicite par le terme $\lambda p \,.\, p$ — à l'aide de la règle de renforcement — mais que sa dénotation est un type dont le contenu extensionnel est vide, ce qui interdit ainsi l'extension de la propriété de validité 3.3.17 à l'ensemble des jugements du calcul implicite. En revanche, l'existence d'un modèle 0-restreint permet de montrer de manière directe la cohérence logique du calcul implicite restreint :

Corollaire 3.3.20 (Cohérence du calcul implicite restreint) — *S'il existe un modèle* 0*-restreint du calcul implicite, alors le calcul implicite restreint est cohérent.*

Preuve. Résulte immédiatement de l'égalité $\mathsf{El}\big([\![\mathsf{False}_0]\!]\big) = \varnothing$ établie par la proposition précédente et de la propriété de validité restreinte 3.3.17. $\square$

Dans les chapitres 5 et 6 nous construirons des modèles 0-restreints du calcul implicite, qui nous permettront ainsi de prouver la cohérence logique du Calcul des Constructions implicite restreint. Ce n'est qu'au chapitre 7 que nous construirons un modèle non-restreint non trivial pour montrer la propriété de normalisation forte de tous les termes bien typés du calcul implicite (avec règle de renforcement).

Dans le cas où $\mathcal{M}$ est un modèle non restreint du calcul implicite, le résultat de validité s'étend automatiquement à tous les jugements du calcul implicite, y compris ceux qui font usage de la règle de renforcement :

Proposition 3.3.21 (Validité non-restreinte) — *Si $\mathcal{M}$ est un modèle non-restreint du calcul implicite, alors :*

- *si* $\Gamma \vdash$ *alors* $[\![\Gamma]\!] \neq \varnothing$
- *si* $\Gamma \vdash M : T$*, alors* $[\![\Gamma]\!] \neq \varnothing$ *et pour toute valuation* $\rho \in \mathbf{Val}_{\mathcal{M}}$ *on a*

$$\rho \in [\![\Gamma]\!] \qquad \Rightarrow \qquad [\![T]\!]_\rho \in \mathcal{T} \quad et \quad [\![M]\!]_\rho \;\in\; \mathsf{El}\big([\![T]\!]_\rho\big)$$

(où $\Gamma \vdash$ *et* $\Gamma \vdash M : T$ *désignent ici des jugements dérivables dans le calcul implicite complet).*

Preuve. Par une récurrence mutuelle sur les dérivations des jugements $\Gamma \vdash$ et $\Gamma \vdash M : T$ en distinguant les cas suivant la dernière règle appliquée. Le cas des règles de bonne formation de contexte (WF-E) et (WF-S) est évident. Le cas des règles de typage autres que la règle de renforcement se traitent de la même manière que dans la preuve de la proposition 3.3.17. Il suffit seulement de s'assurer à chaque étape que l'interprétation du contexte Γ est non vide, ce qui est immédiat d'après le lemme 3.3.16 puisque les prémisses portent toujours sur un

contexte dans lequel Γ est inclus. Reste à traiter le cas de la règle (Str), en considérant une dérivation de la forme

$$\dfrac{\begin{array}{c}\vdots\ d\\ \Gamma;[x:T]\vdash M:U \quad x\notin FV(M) \quad x\notin FV(U)\end{array}}{\Gamma\vdash M:U}\ (\textsc{Str})$$

D'après le lemme 3.3.16 et l'hypothèse de récurrence, $[\![\Gamma]\!]$ est clairement non vide. Soit $\rho\in\Gamma$. Pour montrer que $[\![U]\!]_\rho\in\mathcal{T}$, il est nécessaire d'utiliser un argument de récurrence forte en supposant que l'hypothèse de récurrence est valide pour n'importe quel jugement figurant dans un nœud de la sous-dérivation d, et pas seulement pour le jugement $\Gamma;[x:T]\vdash M:U$. On remarque alors que d contient un sous-arbre étiqueté par un jugement de bonne formation de contexte $\Gamma;[x:T]\vdash$ provenant lui-même d'une application de la règle (WF-S) :

$$d\left\{\begin{array}{l}\vdots\\ \dfrac{\Gamma\vdash T:s}{\Gamma;[x:T]\vdash}\ (\text{WF-S})\\ \vdots\\ \Gamma;[x:T]\vdash M:U\end{array}\right.$$

D'après l'hypotèse de récurrence appliquée à la dérivation conduisant au jugement $\Gamma\vdash T:s$, on a $[\![T]\!]_\rho\in\mathsf{El}([\![s]\!])$, d'où $[\![T]\!]_\rho\in\mathcal{T}$ (d'après l'axiome 6). Le modèle $\mathcal{M}$ étant non-restreint, considérons un objet $a\in\mathsf{El}([\![T]\!]_\rho)$ et posons $\rho'=(\rho;x\leftarrow a)$. On vérifie aisément que ρ' appartient à $[\![\Gamma;[x:T]]\!]$, d'où il ressort en vertu de l'hypothèse de récurrence que $[\![U]\!]_{\rho'}\in\mathcal{T}$ et $[\![M]\!]_{\rho'}\in[\![U]\!]_{\rho'}$. Comme $x\notin FV(M)$ et $x\notin FV(U)$, on a

$$[\![M]\!]_{\rho'}=[\![M]\!]_\rho \qquad \text{et} \qquad [\![U]\!]_{\rho'}=[\![U]\!]_\rho$$

(puisque ρ et ρ' ne diffèrent que sur la variable x), d'où $[\![U]\!]_\rho\in\mathcal{T}$ et $[\![M]\!]_\rho\in[\![U]\!]_\rho$. □

Malheureusement, les modèles non-restreints du calcul implicite ne permettent pas de prouver en général la cohérence logique du calcul implicite complet, puisque dans de tels modèles, les dénotations de tous les types sont habitées. (Remarquons par exemple que le modèle trivial du calcul implicite est un modèle non-restreint.) Cependant, nous introduirons au chapitre 7 une classe intéressante de modèles non-restreints — appelés *modèles de normalisation* — qui nous permettra de prouver la normalisation forte de tous les termes bien typés du Calcul des Constructions implicite (avec la règle de renforcement). La cohérence logique du formalisme complet s'en déduira de manière indirecte, à l'aide de la proposition 2.6.3.

3.4 Quelques notions de théorie des ensembles

Au cours de la construction des modèles du calcul implicite (restreint ou non) que nous effectuerons au chapitres 5, 6 et 7, nous utiliserons fréquemment les outils standards de la théorie des ensembles ZFC.[22] Avant de nous lancer dans ces constructions, il est nécessaire d'effectuer quelques rappels de théorie des ensembles, et plus particulièrement de théorie des cardinaux. Dans toute cette section — de même qu'au cours de la construction des modèles

[22] Théorie des ensembles de Zermelo-Fränkel (ZF) avec axiome du choix (C). Rappelons que cette théorie ne contient ni l'axiome de bonne fondation (AF) ni l'hypothèse généralisée du continu (HGC).

concrets du calcul implicite — nous travaillerons dans ZFC, éventuellement étendu à l'aide de l'axiome de ω-inaccessibilité que nous évoquerons à la fin de cette section. Nous nous bornerons ici à quelques rappels élémentaires, et le lecteur désireux d'approfondir cette question trouvera dans [39] (chapitres 1 et 2) un exposé plus complet sur le sujet.

3.4.1 Ordinaux et cardinaux

On désigne par *On* la collection des ordinaux. Rappelons que *On* ne désigne par un ensemble, mais une partie (au sens intuitif) de l'univers de la théorie de ensembles distinguée par une certaine propriété (ici la propriété "x est un ordinal"). Formellement, une collection n'est rien d'autre qu'une formule à une variable libre éventuellement paramétrée par d'autres variables. Les collections ne sont pas des objets de la théorie, mais plutôt un abus de langage dont l'utilisation est commode.[23]

La collection *On* est munie d'une relation d'ordre $x \leq y$. Cette relation d'ordre est un *bon ordre*, en ce sens que tout ensemble non vide d'ordinaux admet toujours un plus petit élément. Rappelons également qu'un ordinal est égal à l'ensemble des ordinaux strictement plus petits que lui, c'est-à-dire :

$$On(x) \quad \Rightarrow \quad x \;=\; \{y; \quad On(y) \;\text{ et }\; y < x\}.$$

En d'autres termes, si x et y sont des ordinaux, on a les équivalences

$$(x < y \quad \Leftrightarrow \quad x \in y) \qquad \text{et} \qquad (x \leq y \quad \Leftrightarrow \quad x \subset y).$$

On désigne par $0 = \varnothing$ le plus petit ordinal. Tout ordinal x admet un successeur $x+1 = x \cup \{x\}$ qui est le plus petit ordinal strictement plus grand que x. Si E est un ensemble (éventuellement vide) d'ordinaux, E admet une borne supérieure

$$\sup_{x \in E} x \quad = \quad \bigcup_{x \in E} x$$

qui est encore un ordinal. Tout ordinal x est soit
- l'ordinal $x = 0$;
- un ordinal *successeur*, *i.e.* de la forme $x = y + 1$;
- un ordinal *limite*, *i.e.* de la forme $x = \sup_{y<x} y$ avec $x \neq 0$.

Un ordinal x est *fini* s'il est strictement plus petit que tout ordinal limite. Les ordinaux finis sont les entiers naturels de la théorie des ensembles.

Deux ensembles E et E' sont *équipotents* s'il existe une bijection de E sur E'. Un *cardinal* est un ordinal x qui n'est équipotent à aucun ordinal $y < x$. Tous les ordinaux finis sont des

[23] Dans une théorie des types simples à la Church, la distinction ensemble/collection peut s'exprimer de la manière suivante. Les deux types simples de base ι et o désignent respectivement le type des ensembles et le type des propositions. La relation d'appartenance est un objet de type $\iota \to \iota \to o$, et le type des collections est naturellement le type $\iota \to o$. À chaque ensemble $x : \iota$ correspond une collection, qui est donnée par $(\lambda y : \iota \,.\, y \in x) : \iota \to o$. Bien entendu, il y a des collections qui ne correspondent à aucun ensemble, telle que la collection de tous les ensembles $\lambda x : \iota \,.\, x = x$. Le point de vue que nous adopterons par la suite sera beaucoup plus restricitif, puisque nous ne sortirons jamais du langage de la théorie des ensembles du premier ordre. Dans ce cadre, une collection n'est plus un objet de la théorie, mais seulement un abus de langage que l'on peut toujours remplacer par une écriture au premier ordre ayant le même sens. En particulier, nous ne nous autoriserons aucune quantification sur les collections.

cardinaux, ainsi que le plus petit ordinal infini ω. En revanche, $\omega + 1$ n'est pas un cardinal (car équipotent à ω), de même que tous les autres ordinaux dénombrables.[24]

La collection des cardinaux est notée Cn, et est bien ordonnée par la relation d'ordre induite par les ordinaux. Si α et β sont des cardinaux, on a les équivalences

$$\begin{array}{rcl} \alpha \leq \beta & \Leftrightarrow & \text{il existe une injection de } \alpha \text{ dans } \beta \\ & \Leftrightarrow & \text{il existe une surjection de } \beta \text{ sur } \alpha \\ \alpha = \beta & \Leftrightarrow & \text{il existe une bijection de } \alpha \text{ sur } \beta \end{array}$$

Tout ensemble E est équipotent avec un cardinal et un seul, que l'on note $\overline{\overline{E}}$. Tout cardinal α admet un *successeur* (en tant que cardinal), que l'on note α^+, et qui est le plus petit cardinal strictement supérieur à α. Pour tout cardinal infini α, on a

$$\alpha \quad < \quad \alpha^+ \quad \leq \quad 2^\alpha \quad = \quad \overline{\overline{\mathfrak{P}(\alpha)}},$$

et l'égalité $\alpha^+ = 2^\alpha$ n'est pas décidable par les axiomes de ZFC.[25] Par ailleurs, la borne supérieure d'un ensemble de cardinaux E

$$\sup_{\alpha \in E} \alpha \quad = \quad \bigcup_{\alpha \in E} \alpha$$

est encore un cardinal.

Les cardinaux infinis sont en même nombre (au sens intuitif) et ordonnés de la même manière (au sens intuitif) que les ordinaux eux mêmes. Pour tout ordinal x (fini ou infini), on note $\aleph_x$ ("aleph x") le x-ième cardinal infini. La suite transfinie $(\aleph_x)$[26] est définie par

$$\begin{array}{lcll} \aleph_0 & = & \omega & \text{(cardinal dénombrable)} \\ \aleph_{x+1} & = & \aleph_x^+ & \\ \aleph_x & = & \sup_{y<x} \aleph_y & \text{(si } x \text{ ordinal limite)} \end{array}$$

La relation "$\alpha = \aleph_x$" réalise un isomorphisme (au sens intuitif) entre la collection des ordinaux et la collection des cardinaux infinis.

[24] Les ordinaux dénombrables jouent un rôle important dans l'étude de l'expressivité logique des théories du premier ordre ou même des théories des types. Rappelons que les ordinaux $\omega+\omega = \omega.2$, ω^2, ω^ω et ω^{ω^ω} sont des ordinaux dénombrables, de même que ε_0 (ordinal de l'arithmétique) et Γ_0 (ordinal imprédicatif). Le plus petit ordinal infini non-dénombrable — c'est-à-dire le cardinal $\aleph_1$ — est donc au-delà de tous ces ordinaux.

[25] L'énoncé "pour tout cardinal infini α, $\alpha^+ = 2^\alpha$ ", indécidable dans ZFC, est appelé *hypothèse généralisée du continu* (HGC). On désigne par *hypothèse du continu* (HC) l'énoncé plus faible "$\aleph_1 = 2^{\aleph_0}$", où $\aleph_0 = \omega$ désigne le cardinal dénombrable, et $\aleph_1$ son successeur (en tant que cardinal) qui est le plus petit cardinal infini non-dénombrable. De manière équivalente, l'hypothèse du continu exprime que tout sous-ensemble infini de la droite réelle est soit dénombrable, soit équipotent à la droite réelle elle-même.

[26] La suite transfinie des $\aleph_x$ (indicée par les ordinaux) n'est pas une suite (ou une fonction) au sens usuel du terme, car elle ne constitue pas un ensemble. En toute rigueur, on devrait parler de la *relation fonctionnelle* "$\alpha = \aleph_x$", qui est une formule de la théorie des ensembles à deux variables libres x et α. Nous ne détaillerons pas davantage les techniques [39] qui permettent d'écrire un tel énoncé dans le langage du premier ordre de la théorie des ensembles, ni la justification de ce qu'on appelle une *définition par récurrence transfinie*.

3.4.2 Cofinalité et cardinaux réguliers

Définition 3.4.1 (Cofinalité) — Soit x un ordinal. On appelle *cofinalité* de x le plus petit ordinal $y \leq x$ tel qu'il existe une suite strictement croissante $(x_i)_{i<y}$ d'ordinaux $x_i < x$ indicée par y qui n'est pas strictement bornée dans x. La cofinalité de x est notée $\mathrm{cof}(x)$.

Intuitivement, la cofinalité d'un ordinal x exprime le nombre minimum d'opérations (donné par $\mathrm{cof}(x)$) qu'il est nécessaire d'effectuer pour "grimper tout en haut" de l'ordinal x à l'aide d'ordinaux strictement plus petits que x.

Par exemple, $\mathrm{cof}(0) = 0$ et $\mathrm{cof}(n) = 1$ pour tout entier naturel $n \neq 0$. La cofinalité du premier ordinal infini ω est l'ordinal ω lui-même. Plus généralement, si x est un ordinal dénombrable, on a soit $\mathrm{cof}(x) = 1$ (dans le cas où x est un ordinal successeur), soit $\mathrm{cof}(x) = \omega$ (dans le cas où x est un ordinal limite), ce qui exprime dans ce dernier cas que tout ordinal limite dénombrable peut être obtenu en calculant la limite *dénombrable* d'une suite strictement croissante d'ordinaux strictement plus petits que lui.

Pour tout ordinal x, $\mathrm{cof}(x)$ est un cardinal. De plus la fonction cof (au sens intuitif) est idempotente puisque $\mathrm{cof}(\mathrm{cof}(x)) = \mathrm{cof}(x)$ quel que soit l'ordinal x.

Définition 3.4.2 (Ordinal régulier) — Un ordinal x est *régulier* si $\mathrm{cof}(x) = x$.

Tout ordinal régulier est un cardinal (par la suite, on ne parlera donc que de *cardinaux réguliers*) et la cofinalité de n'importe quel ordinal est un cardinal régulier.

Les cardinaux finis réguliers sont 0 et 1, et le plus petit cardinal infini régulier est $\omega = \aleph_0$. Tous les cardinaux successeurs infinis (*i.e.* de la form $\aleph_{x+1}$ pour un certain ordinal x) sont des cardinaux réguliers. Le plus petit cardinal infini non-régulier (ou *singulier*) est $\aleph_\omega$, puisque $\mathrm{cof}(\aleph_\omega) = \omega$.

Proposition 3.4.3 (Cardinaux infinis réguliers) — *Un cardinal infini μ est regulier si et seulement si pour toute famille d'ensembles $(E_i)_{i\in I}$ on a*

$$\overline{\overline{I}} < \mu \quad et \quad \left(\forall\, i \in I \quad \overline{\overline{E_i}} < \mu\right) \quad \Rightarrow \quad \overline{\overline{\bigcup_{i\in I} E_i}} \;<\; \mu.$$

Un cardinal infini régulier μ partage avec $\aleph_0$ cette propriété qu'il n'est pas possible d'atteindre la cardinalité de μ en effectuant la réunion d'une famille d'ensembles E_i de cardinalité $\overline{\overline{E_i}} < \mu$, elle-même indicée par un ensemble I de cardinalité $\overline{\overline{I}} < \mu$, de la même façon que la réunion d'une famille finie d'ensembles finis reste toujours un ensemble fini.

À ce titre, $\aleph_\omega$ n'est pas un cardinal régulier (c'est même le plus petit cardinal infini singulier) car il est égal à la réunion dénombrable des cardinaux $\aleph_i$ ($i \in \omega$) eux-mêmes strictement plus petits que $\aleph_\omega$.

3.4.3 Ensembles transitifs et ensembles bien fondés

La théorie des ensembles est une théorie du premier ordre mono-sortée dans laquelle tous les objets de la théorie sont des ensembles, y compris leurs éléments. Pour comprendre la structure intime d'un ensemble, il n'est parfois pas suffisant de connaître ses éléments, mais il faut également connaître la structure de ses éléments en tant qu'ensembles, c'est-à-dire les

éléments de ses éléments, et ainsi de suite. Pour ce faire, on introduit les notions d'*ensemble transitif* et de *clôture transitive*.

Définition 3.4.4 (Ensemble transitif) — Un ensemble X est *transitif* si tout élément de X est une partie de X, c'est-à-dire :

$$\forall\, x, y \qquad y \in x \quad \text{et} \quad x \in X \quad \Rightarrow \quad y \in X.$$

Tous les ordinaux sont des ensembles transitifs, mais il existe des ensembles transitifs qui ne sont pas des ordinaux, tels que l'ensemble $\{\varnothing; \{\varnothing\}; \{\{\varnothing\}\}\}$. L'ensemble $\{\{\{\varnothing\}\}\}$ est un exemple simple d'ensemble non-transitif.

Définition 3.4.5 (Clôture transitive) — Soit E un ensemble. On appelle *clôture transitive* de E et on note $\mathrm{Cl}(E)$ l'ensemble défini par $\mathrm{Cl}(E) = \bigcup_{i\in\omega} E_i$ où $(E_i)_{i\in\omega}$ est la suite d'ensembles définie par récurrence sur $i \in \omega$ par $E_0 = X$ et $E_{i+1} = \bigcup_{x\in X_i} x$.

Par définition, la clôture transitive de E est le plus petit ensemble transitif dans lequel E est inclus. Par exemple, la clôture transitive de l'ensemble (non-transitif) $\{\{\{\varnothing\}\}\}$ est l'ensemble (transitif) $\{\{\{\varnothing\}\}; \{\varnothing\}; \varnothing\}$. Notons qu'un ensemble E est transitif si et seulement si $\mathrm{Cl}(E) = E$. La clôture transitive d'un ensemble est caractérisée de manière récursive par

Lemme 3.4.6 — *Pour tout ensemble E on a*

$$\mathrm{Cl}(E) \quad = \quad E \,\cup \bigcup_{x\in E} \mathrm{Cl}(x).$$

Définition 3.4.7 (Ensemble héréditairement fini) — Un ensemble X est dit *héréditairement fini* si tous les éléments de $\mathrm{Cl}(\{X\})$ sont des ensembles finis.

Notons que $\mathrm{Cl}(\{X\}) = \{X\} \cup \mathrm{Cl}(X)$. Intuitivement, un ensemble héréditairement fini est un ensemble fini dont les éléments sont finis, les éléments de ses éléments sont finis, etc.

Tous les ordinaux finis sont des ensembles héréditairement finis. L'ensemble $\{\{\{\varnothing\}\}; \varnothing\}\}$ est un ensemble héréditairement fini qui n'est pas un ordinal. Enfin le singleton $\{\omega\}$ est fini, mais pas héréditairement fini. La propriété de finitude héréditaire est caractérisée récursivement par

Lemme 3.4.8 — *Un ensemble X est héréditairement fini si et seulement si X est fini et si tous les éléments de X sont des ensembles héréditairement finis.*

Définition 3.4.9 (Ensemble bien fondé) — Un ensemble X est dit *bien fondé* si la restriction de la relation $x \in y$ à l'ensemble $\mathrm{Cl}(X)$ est une relation bien fondée.

Lemme 3.4.10 — *Un ensemble X est bien fondé si et seulement si tous ses éléments sont des ensembles bien fondés.*

Notons que tous les ordinaux sont naturellement des ensembles bien fondés. Dans la théorie des ensembles plus généralement, il n'est pas possible de construire explicitement un ensemble qui ne soit pas bien fondé, bien que les axiomes de ZFC ne permettent pas de prouver que tous les ensembles sont bien fondés. C'est pourquoi on y ajoute fréquement l'*axiome de la*

fondation (AF) qui peut s'énoncer par "tous les ensembles sont bien fondés". Dans ce qui suit, nous ne supposerons pas un tel axiome. L'intérêt majeur de cette notion est que la collection des ensembles bien fondés (dont on vérifie aisément qu'elle est stable par tous les axiomes de ZFC) se prête naturellement aux raisonnement par *récurrence bien fondée*. Pour cela, il est nécessaire d'introduire la *hiérarchie cumulative* (V_x).

3.4.4 Hiérarchie cumulative (V_x)

Définition 3.4.11 (Hiérarchie cumulative) — On appelle *hiérarchie cumulative* la suite transfinie (V_x) indicée par la collection des ordinaux définie par

$$\begin{array}{lcl} V_0 & = & \varnothing \\ V_{x+1} & = & \mathfrak{P}(V_x) \\ V_x & = & \bigcup_{y<x} V_y \quad \text{(si } x \text{ ordinal limite)} \end{array}$$

(Rappelons que la suite transfinie n'est pas une fonction au sens usuel du terme ni même un ensemble, mais une relation fonctionelle "$y = V_x$" à deux variables libres x et y.)

Proposition 3.4.12 — *La suite transfinie (V_x) est une suite croissante d'ensembles transitifs. Par ailleurs :*

1. *$V_x \subset V_y$ si et seulement si $x \leq y$*
2. *$V_x \in V_y$ si et seulement si $x \in y$*
3. *Les ordinaux appartenant à V_x sont les ordinaux $y < x$, c'est-à-dire :*

$$\{y \in V_x; \quad On(y)\} \quad = \quad x.$$

Remarquons que pour tout ordinal fini x, l'ensemble V_x est fini, et même héréditairement fini. L'ensemble V_ω est le premier ensemble infini de cette hiérarchie, et c'est même l'ensemble de tous les ensembles héréditairement finis bien fondés (cet ensemble est dénombrable). La suite des ensembles V_x croît très vite, puisque V_ω est dénombrable, $V_{\omega+1}$ a la puissance du continu (en particulier, $\mathfrak{P}(\omega) \subset V_{\omega+1}$), etc. En fait, la collection V formée par la réunion (au sens intuitif) de tous les ensembles V_x est égale à la collection de tous les ensembles bien fondés :[27]

Proposition 3.4.13 (Collection V) — *Pour tout ensemble X on a*

$$\begin{array}{rcl} X \text{ bien fondé} & \Leftrightarrow & V(X) \\ & \Leftrightarrow & \exists\, x \quad On(x) \quad et \quad X \in V_x \end{array}$$

Définition 3.4.14 (Rang d'un ensemble bien fondé) — Soit X un ensemble bien fondé. On appelle *rang de X* le plus petit ordinal x tel que $X \in V_x$, que l'on note $\mathsf{rk}(X)$.

Remarquons que pour tout ensemble X bien fondé, on a

$$\mathsf{rk}(X) \quad = \quad \sup_{Y \in X} \mathsf{rk}(Y) \; + \; 1$$

et que chaque ensemble V_x n'est autre que l'ensemble de tous les ensembles bien fondés de rang strictement plus petit que x. (En particulier, un ensemble bien fondé est héréditairement fini si et seulement si son rang est fini.) Notons également que pour tout ordinal x, on a $\mathsf{rk}(x) = x + 1$.

[27] Si on suppose l'axiome de la fondation, la collection V est donc égale à la collection de tous les ensembles.

3.4.5 Ensembles inaccessibles et cardinaux inaccessibles

Définition 3.4.15 (Ensemble inaccessible) — Un ensemble E est dit *inaccessible* s'il est transitif et s'il satisfait les axiomes suivants :

1. $\omega \in E$
2. si $X \in E$, alors $\mathfrak{P}(X) \in E$
3. si $X \in E$ et $Y_x \in E$ pour tout $x \in X$, alors $\bigcup_{x \in X} Y_x \in E$.

Intuitivement, un ensemble inaccessible est stable par toutes les opérations de la théorie des ensembles de Zermelo-Fränkel. Tout ensemble inaccessible constitue immédiatement un modèle de ZFC, et permet par conséquent de prouver la cohérence logique de cette théorie. Il n'y a donc que peu d'espoir à pouvoir construire un tel ensemble dans ZFC, à moins que cette théorie ne soit incohérente (en vertu du second théorème d'incomplétude de Gödel).

Une propriété essentielle des ensembles inaccessibles est la suivante :

Proposition 3.4.16 (Stabilité par formation de produit dépendant) — *Si E est un ensemble inacessible, alors E est clos par formation de produit ensembliste arbitraire, c'est-à-dire : si $T \in E$ et $U_x \in E$ pour tout $x \in T$, alors $\prod_{x \in T} U_x \in E$.*

Définition 3.4.17 (Cardinal inaccessible) — Un cardinal μ est dit *inaccessible* s'il satisfait les conditions suivantes

1. $\aleph_0 < \mu$
2. si $\alpha < \mu$, alors $2^\alpha < \mu$
3. si $\alpha < \mu$ et $\beta_i < \mu$ pour tout $i \in \alpha$, alors $\sup_{i \in \alpha} \beta_i < \mu$.

Notons qu'il n'y pas de relation triviale entre les ensembles inaccessibles et les cardinaux du même nom : un cardinal inaccessible ne constitue pas un ensemble inaccessible, et réciproquement, un ensemble inaccessible n'est pas un cardinal inaccessible (ni même un cardinal ou un ordinal). En revanche, le *cardinal* d'un ensemble inaccessible est un cardinal inaccessible :

Proposition 3.4.18 (Cardinal d'un ensemble inaccessible) — *Si E est un ensemble inaccessible, alors son cardinal $\overline{\overline{E}}$ est un cardinal inaccessible.*

Autrement dit, l'existence d'un ensemble inaccessible entraîne l'existence d'un cardinal inaccessible. En fait, l'existence d'un ensemble inaccessible est même équivalente à l'existence d'un cardinal du même nom. La partie réciproque est une conséquence immédiate de la proposition suivante :

Proposition 3.4.19 (Ensemble inaccessible) — *Si μ est un cardinal inaccessible, alors :*

1. *V_μ est un ensemble inaccessible*
2. *$\overline{\overline{V_\mu}} = \mu = \aleph_\mu$*

Dans la suite de cette thèse, on utilisera fréquement l'*axiome de ω-inaccessibilité* (AI^ω), qui s'énonce ainsi :

Axiome 3.4.20 (ω-inaccessibilité) — *Il existe une infinité de cardinaux inaccessibles.*

La théorie ZFC+AI$^\omega$ est beaucoup plus forte que ZFC : elle permet de montrer la cohérence de ZFC, et même de théories déjà bien plus fortes que ZFC (par exemple, toutes les théories de la forme ZFC+ "il existe n cardinaux inaccessibles" où n est un entier).[28]

La théorie ZFC + AI$^\omega$ est très adaptée à la construction de modèles des théories des types imprédicatives avec univers. L'axiome de ω-inaccessibilité entraîne l'existence d'une suite strictement croissante $(\mu_i)_{i\in\omega}$ de cardinaux inaccessibles, laquelle permet de construire la suite d'ensembles inaccessibles emboîtés $(V_{\mu_i})_{i\in\omega}$. Cette suite de modèles de la théorie des ensembles satisfait tous les critères d'une hiérarchie d'univers (dans le cadre de l'interprétation classique) puisque :

1. $V_{\mu_0} \in V_{\mu_1} \in V_{\mu_2} \in \cdots \in V_{\mu_i} \in V_{\mu_{i+1}} \in \cdots$
2. $V_{\mu_0} \subset V_{\mu_1} \subset V_{\mu_2} \subset \cdots \subset V_{\mu_i} \subset V_{\mu_{i+1}} \subset \cdots$
3. les ensembles V_{μ_i} sont clos par formation de produit dépendant

Dans le modèle classique par ailleurs (*cf* paragraphe 3.1.1), la construction du premier univers Type_1 ne nécessite pas l'hypothèse d'inaccessibilité, puisque l'ensemble (accessible) V_ω satisfait tous les critères requis, et en particulier :

Proposition 3.4.21 (Clôture de V_ω par produits ensemblistes) — *Si $T \in V_\omega$ et $U_x \in V_\omega$ pour tout $x \in T$, alors $\prod_{x\in T} U_x \in V_\omega$.*

Une telle interprétation de Type_1 par V_ω n'est possible que dans la mesure où les ensembles $\mathbf{0} = \varnothing$, $\mathbf{1} = \{\bullet\}$ et la paire $\{\mathbf{0};\mathbf{1}\}$ interprétant Prop sont des ensembles héréditairement finis bien fondés, et par conséquent des éléments de V_ω.[29]

Dans le cas du modèle de CCω dans ω-**Set**, cette interprétation de Type_1 par V_ω n'est plus possible, puisque le type des propositions est déjà interprété par un ensemble infini. L'hypothèse d'inaccessibilité est alors requise pour interpréter le premier univers prédicatif.[30]

[28] Bien entendu, il existe des théories des ensembles strictement plus fortes que ZFC + AI$^\omega$, telles que la théorie des ensembles de Tarski-Grothendieck, dans laquelle on dispose d'autant de cardinaux inaccessibles que d'ordinaux. Cette théorie des ensembles est à la base du système Mizar.

[29] Pour que l'ensemble $\mathbf{1} = \{\bullet\}$ et la paire $\{\mathbf{0};\mathbf{1}\}$ soient des éléments de V_ω, il faut et il suffit que l'objet $\bullet$ dénotant tous les termes de preuve soit lui-même un élément de V_ω.

[30] Précisons que l'utilisation des cardinaux inaccessibles n'est pas strictement nécessaire pour interpréter CCω (ou ECC). En particulier, la preuve de normalisation forte de Z. Luo [42] s'effectue dans ZF, et les idées sur lesquelles sont basées cette preuve ont été depuis reprises par P.-A. Melliès et B. Werner [48] pour construire des modèles de normalisation génériques pour les systèmes de types imprédicatifs avec univers. En l'absence de l'hypothèse d'inaccessibilité, la construction du modèle devient beaucoup plus ardue puisqu'elle nécessite d'introduire des outils supplémentaires afin de limiter l'explosion combinatoire de la taille des espaces de fonctions. En pratique, les outils utilisés par [48] permettent de construire des modèles de CCω qui "tiennent" dans ZF (et même dans l'ensemble V_{ω^2}). Dans la suite de cette thèse cependant, nous n'introduirons pas de tels outils afin de limiter les difficultés, et nous n'hésiterons pas à invoquer l'existence de cardinaux inaccessibles chaque fois que cela nous semblera nécessaire, ce qui simplifiera grandement la construction des modèles, déjà délicate par ailleurs.

Chapitre 4

Espaces cohérents

Ce chapitre — dont la structure est très fortement inspirée de [31] — est une présentation détaillée de la théorie des espaces cohérents, qui constitue la charpente théorique des différents modèles du calcul implicite que nous présenterons au cours des chapitres 5, 6 et 7. Initialement, les espaces cohérents ont été introduits par J.-Y. Girard afin d'interpréter le système F dans le cadre de la sémantique dénotationnelle [29]. C'est également la notion d'espace cohérent qui lui a permis de découvrir quelques années plus tard que les connecteurs usuels de la logique pouvaient être décomposés de manière plus fine à l'aide de connecteurs linéaires et de deux exponentielles, donnant ainsi naissance à une nouvelle branche de la logique : la logique linéaire.

L'usage que nous allons faire des espaces cohérents au cours de cette thèse est à cet égard assez atypique, puisqu'il n'est pas directement relié à la logique linéaire ou à l'une de ses problématiques.[1] Ici, nous allons davantage nous servir des espaces cohérents comme d'un outil de construction de modèles du λ-calcul, et qui a en outre l'avantage d'être sans doute une des structures les plus simples à manipuler parmi les différentes structures de domaines qui ont été introduites depuis les travaux de Scott [57, 58] en sémantique dénotationnelle. Cependant, nous verrons au chapitre 5 que le choix d'utiliser les espaces cohérents plutôt qu'une autre structure de domaine n'est pas seulement pragmatique, mais que le paradigme que nous allons utiliser pour interpréter le typage et le sous-typage est en fait très lié à la structure des espaces cohérents et à la stabilité.[2]

Ce chapitre est organisé autour de trois grandes sections. Dans la première section sont introduits tous les concepts directement rattachés à la catégorie des espaces cohérents : stabilité, produit direct et somme directe, espaces des fonctions stables et des fonctions linéaires, etc. La deuxième section est consacrée à la notion de plongement rigide et aux concepts qui lui sont associés, telles que la notion de colimite ou la méthode de construction d'un modèle du λ-calcul pur. Dans la troisième section, nous proposons deux généralisations naturelles de la notion de stabilité, qui sont les notions de quasi-stabilité et de μ-stabilité. Nous y montrons en particulier que la plupart des résultats évoqués dans les deux premières sections se transposent naturellement dans ce cadre plus général, et nous terminons ce chapitre en montrant

[1] Les constructions que nous allons effectuer dans les chapitres suivants — toutes basées sur la notion de μ-stabilité où μ désigne un cardinal considérablement grand — sont même à l'opposé de l'esprit d'"économie" prôné par la logique linéaire.

[2] On pourra s'en convaincre par exemple en examinant la proposition 5.1.6.

que les catégories des μ-espaces cohérents constituent un cadre naturel pour la construction de λ-modèles sous-extensionnels tels qu'ils ont été définis au chapitre 3.

4.1 La catégorie des espaces cohérents

4.1.1 Définitions

Définition 4.1.1 — Un *espace cohérent* est un ensemble d'ensembles $\mathcal{A}$ satisfaisant les critères suivants :

1. (Clôture inférieure) Si $a \in \mathcal{A}$ et $a' \subset a$, alors $a' \in \mathcal{A}$;
2. (Complétude binaire) Pour toute famille $(a_i)_{i\in I}$ à valeurs dans $\mathcal{A}$ on a :

$$(\forall\, i, j \in I \quad a_i \cup a_j \in \mathcal{A}) \qquad \Rightarrow \qquad \bigcup_{i\in I} a_i \;\in\; \mathcal{A}$$

Soit $\mathcal{A}$ un espace cohérent. D'après le critère de complétude binaire appliqué à la famille vide, l'ensemble vide $\varnothing$ est un élément de $\mathcal{A}$, lequel représente l'*objet indéfini*. Les éléments de $\mathcal{A}$ sont appelés ses *points*, et ces points sont eux-mêmes des ensembles dont les éléments sont appelés les *atomes* de $\mathcal{A}$. L'espace cohérent $\mathcal{A}$ est naturellement ordonné par l'ordre de l'inclusion, pour lequel l'objet indéfini $\varnothing$ constitue le plus petit élément de $\mathcal{A}$.

L'ensemble des atomes de $\mathcal{A}$, appelé *trame de* $\mathcal{A}$ et noté $|\mathcal{A}|$, est donné par

$$|\mathcal{A}| \;=\; \bigcup \mathcal{A} \;=\; \{\alpha; \quad \exists a \in \mathcal{A} \quad \alpha \in a\}$$

Pour tout atome $\alpha \in |\mathcal{A}|$, on a $\{\alpha\} \in \mathcal{A}$ en vertu du critère de clôture inférieure, ce qui nous permet de caractériser la trame de $\mathcal{A}$ par

$$|\mathcal{A}| \;=\; \big\{\alpha; \quad \{\alpha\} \in \mathcal{A}\big\}$$

Dans ce qui suit, nous désignerons les espaces cohérents par les lettres capitales en écriture cursive $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, etc. ; les points de ces espaces seront désignés par les lettres minuscules latines a, b, c, etc. et les atomes constituant ces points par les minuscules grecques α, β, γ, etc.

Présentation en termes de graphes L'espace cohérent $\mathcal{A}$ — qui est un ensemble de parties de $|\mathcal{A}|$ — est, en vertu du critère de complétude binaire, caractérisé par les paires d'atomes qui y figurent, puisque pour tout ensemble d'atomes $a \subset |\mathcal{A}|$ on a

$$a \in \mathcal{A} \qquad \Leftrightarrow \qquad \forall \alpha_1, \alpha_2 \in a \;\; \{\alpha_1; \alpha_2\} \in \mathcal{A}.$$

On note alors $\alpha_1 \frown \alpha_2 \pmod{\mathcal{A}}$ la relation binaire définie par

$$\alpha_1 \frown \alpha_2 \pmod{\mathcal{A}} \quad \equiv \quad \{\alpha_1; \alpha_2\} \in \mathcal{A},$$

appelée *relation de cohérence* sur $|\mathcal{A}|$. Cette relation réflexive et symétrique confère à $|\mathcal{A}|$ une structure de *graphe non-orienté* qui caractérise les points de $\mathcal{A}$ puisque pour tout ensemble d'atomes $a \subset |\mathcal{A}|$ on a

$$a \in \mathcal{A} \qquad \Leftrightarrow \qquad \forall \alpha_1, \alpha_2 \in a \quad \alpha_1 \frown \alpha_2 \pmod{\mathcal{A}}.$$

Autrement dit, les points de $\mathcal{A}$ sont précisément les *cliques* du graphe non-orienté $(|\mathcal{A}|, \supset)$, c'est-à-dire les sous-graphes complets de $\mathcal{A}$ (*i.e.* les sous-graphes dont deux sommets quelconques sont joints par une arête).

Réciproquement, si G est un graphe non-orienté, l'ensemble de ses cliques constitue un espace cohérent, dont la trame est l'ensemble des sommets de G et la relation de cohérence est précisément la relation donnée par les arêtes de G. Autrement dit, les correspondances réciproques définies ci-dessus établissent une bijection entre la classe des espaces cohérents et la classe des graphes non-orientés. De manière complètement équivalente, nous aurions donc pu définir la notion d'espace cohérent de la manière suivante :

Un espace cohérent est l'ensemble des cliques d'un graphe non orienté.

Dans la figure 4.1 nous avons représenté un graphe non-orienté à 4 éléments, ainsi que l'espace cohérent associé. Dans le dessin de gauche, les traits reliant les sommets représentent les arêtes (ou si l'on préfère, la relation de cohérence) tandis que dans le dessin de droite, les traits reliant les points (ou si l'on préfère, les cliques) représentent l'ordre de l'inclusion.

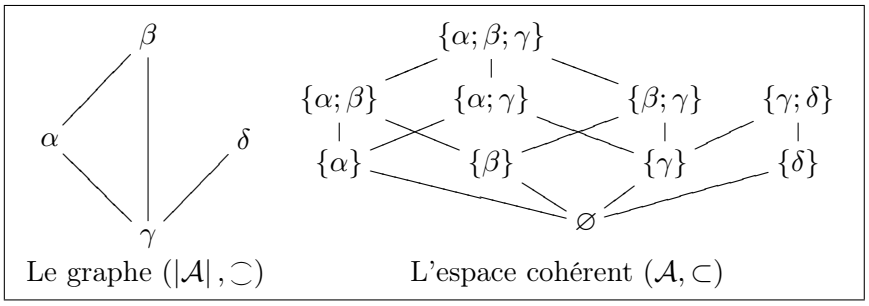


FIG. 4.1 – Un graphe non-orienté et l'espace cohérent associé

Dans ce qui suit nous adopterons indifféremment l'un ou l'autre des deux points de vue, et nous emploierons dorénavant les mots "point" et "clique" comme des synonymes.

4.1.2 Structure de l'ensemble ordonné $(\mathcal{A}, \subset)$

Contrairement aux domaines abstraits, un espace cohérent n'est pas seulement un ensemble de points abstraits muni d'une relation d'ordre abstraite satisfaisant certains critères. La structure d'espace cohérent donne non seulement accès à l'*implémentation des points* — ce sont eux-mêmes des ensembles d'*atomes* — mais aussi à l'*implémentation de la relation qui les ordonne* — c'est la relation d'inclusion, dont les propriétés nous sont familières.

Cette particularité des espaces cohérents qui nous permet de raisonner sur des entités — les atomes — encore plus élémentaires que les points qui composent ces espaces, n'est pas spécifique aux espaces cohérents, mais est caractéristique des domaines concrets [37], dont l'étude constitue à elle seule une branche particulière de la théorie des domaines abstraits — la théorie des domaines concrets — et dont la théorie des espaces cohérents fait partie.

Dans ce paragraphe, nous allons étudier les propriétés "abstraites" de la relation d'inclusion entre cliques, qui confère aux espaces cohérents leur structure d'ensemble ordonné. Rappelons que la première de ces propriétés est l'existence d'un plus petit élément, qui est la clique vide $\varnothing$. Dans ce qui suit, $\mathcal{A}$ désigne un espace cohérent quelconque.

Bornes inférieure et supérieure Considérons une famille $(a_i)_{i\in I}$ d'éléments de $\mathcal{A}$ indicée par un ensemble I quelconque.

- Si l'ensemble d'indices I est non vide, la famille $(a_i)_{i\in I}$ admet une borne inférieure (dans $\mathcal{A}$), qui n'est autre que l'intersection des cliques a_i (laquelle constitue une clique en vertu du critère de *clôture inférieure*) :

$$\inf_{i\in I} a_i \;=\; \bigcap_{i\in I} a_i.$$

- Pour tout ensemble d'indices I (y compris l'ensemble vide), la famille $(a_i)_{i\in I}$ admet une borne supérieure (dans $\mathcal{A}$) si et seulement si la réunion des cliques a_i est également une clique, c'est à dire un point de $\mathcal{A}$, et sa borne supérieure est donnée par

$$\sup_{i\in I} a_i \;=\; \bigcup_{i\in I} a_i.$$

  Lorsque cette borne supérieure existe, nous dirons que $(a_i)_{i\in I}$ est une *famille de cliques compatibles*. Une condition nécessaire et suffisante pour que $(a_i)_{i\in I}$ soit une famille de cliques compatibles est que pour tous $i, j \in I$, les cliques a_i et a_j sont compatibles (c'est-à-dire $a_i \cup a_j \in \mathcal{A}$).

Ensembles dirigés et approximants finis Un sous-ensemble $D \subset \mathcal{A}$ non vide est dit *dirigé* si toute paire d'éléments de D admet un majorant dans D, c'est-à-dire :

$$\forall\, a_1, a_2 \in D \quad \exists\, a_3 \in D \qquad a_1 \subset a_3 \quad \text{et} \quad a_2 \subset a_3.$$

Si $D \subset \mathcal{A}$ est un ensemble dirigé, alors D admet une borne supérieure. Autrement dit, l'ensemble ordonné $(\mathcal{A}, \subset)$ est un CPO.[3]

Les éléments *compacts* (ou *finis*, au sens de Scott) sont exactement les cliques finies de $\mathcal{A}$, c'est-à-dire les points de $\mathcal{A}$ constitués d'un nombre fini d'atomes. L'ensemble des cliques finies de $\mathcal{A}$ est noté $\mathcal{A}_{\mathbf{fin}}$.

Si a est un point de $\mathcal{A}$, on appelle *approximant fini de* a toute clique finie $a_0 \in \mathcal{A}_{\mathbf{fin}}$ telle que $a_0 \subset a$. L'ensemble des approximants finis de a, noté $\Im(a)$, est un sous-ensemble dirigé de $\mathcal{A}$, dont la borne supérieure est le point a lui-même. Autrement dit, l'ensemble ordonné $(\mathcal{A}, \subset)$ est un domaine *algébrique*.

Structure de domaine de Scott Dans les lignes qui précèdent, nous avons montré qu'un espace cohérent, en tant qu'ensemble ordonné, est un CPO algébrique avec un plus petit élément qui est la clique vide. Par ailleurs, il est immédiat que toute partie majorée d'un espace cohérent admet une borne supérieure (en vertu du critère de clôture inférieure), ce qui achève de montrer le fait que tout espace cohérent a une structure sous-jacente de domaine de Scott [58].

Pourtant, contrairement aux domaines de Scott, la notion de morphisme associée aux espaces cohérents n'est pas la notion de fonction continue, mais la notion (plus forte) de *fonction stable*, laquelle se révèle davantage adaptée à structure concrète des espaces cohérents.

[3] *Complete Partial Order* en anglais.

4.1.3 Fonctions stables

Définition 4.1.2 — Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents. Une fonction $F : \mathcal{A} \to \mathcal{B}$ est dite *stable* si elle satisfait les critères suivants :

1. (Monotonie) si $a' \subset a$, alors $F(a') \subset F(a)$;
2. (Continuité) si $(a_i)_{i\in I}$ est une famille dirigée de points de $\mathcal{A}$, alors
$$F\Big(\bigcup_{i\in I} a_i\Big) \;=\; \bigcup_{i\in I} F(a_i).$$
3. (Stabilité binaire) Si $a_1 \cup a_2 \in \mathcal{A}$, alors $F(a_1 \cap a_2) \;=\; F(a_1) \cap F(a_2)$.

La fonction identité est clairement une fonction stable, de même que la composée de deux fonctions stables est également une fonction stable. Par conséquent, la classe des espaces cohérents munie des fonctions stables constitue une catégorie, appelée *catégorie des espaces cohérents*.

Remarques 4.1.3
1. Les critères 1 et 2 correspondent à la définition usuelle de la notion de *fonction continue* au sens de Scott. Une fonction stable est donc une fonction continue satisfaisant de surcroît le critère de stabilité binaire (critère 3).
2. Le critère 2 (continuité) dans la définition ci-dessus peut être remplacé par la condition équivalente :

   2b. (Continuité) Pour tout $a \in \mathcal{A}$, $F(a) \;=\; \bigcup_{a_0 \in \Im(a)} F(a_0)$

   (où $\Im(a)$ désigne l'ensemble des approximants finis de a) qui exprime que l'image d'un point de $\mathcal{A}$ est la borne supérieure des images de ses approximants finis. Remarquons que sous-cette forme, le critère 2b entraîne le critère 1 (monotonie), qui est donc superflu dans la définition. Par ailleurs, le critère 3 (stabilité binaire) entraîne également le critère 1.
3. Le critère 3 (stabilité binaire) s'étend naturellement par récurrence à une intersection finie non vide de cliques compatibles entre elles :

   3b. (Stabilité n-aire) Si $a_1 \cup \cdots \cup a_n \in \mathcal{A}$ (avec $n > 0$), alors
$$F(a_1 \cap \cdots \cap a_n) \;=\; F(a_1) \cap \cdots \cap F(a_n).$$

Fonctions stables à plusieurs arguments

Définition 4.1.4 — Soient $\mathcal{A}_1, \ldots, \mathcal{A}_n$ et $\mathcal{B}$ des espaces cohérents. Une fonction F définie sur $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ et à valeurs dans $\mathcal{B}$ est dite *stable* si elle satisfait les critères suivants :

1. (Monotonie) Pour tous $a_i, a'_i \in \mathcal{A}_i$ $(i \in [1..n])$
$$(\forall\, i \quad a'_i \subset a_i) \qquad \Rightarrow \qquad F(a'_1, \ldots, a'_n) \subset F(a_1, \ldots, a_n).$$
2. (Continuité) Si $D_1, \ldots, D_n$ sont des sous-ensembles dirigés des espaces $\mathcal{A}_1, \ldots, \mathcal{A}_n$ respectivement, alors :
$$F(\sup D_1, \ldots, \sup D_n) \;=\; \sup_{D_1 \times \cdots \times D_n} F.$$

3. (Stabilité binaire) Pour tous $a_i, a'_i \in \mathcal{A}_i$ $(i \in [1..n])$

$$(\forall\, i \quad a_i \cup a'_i \in \mathcal{A}_i) \quad \Rightarrow \quad F(a_1 \cap a'_1, \ldots, a_n \cap a'_n) \;=\; F(a_1, \ldots, a_n) \cap F(a'_1, \ldots, a'_n)$$

Là encore, une fonction à plusieurs arguments est stable si elle est continue au sens de Scott (critères 1 et 2) et si elle satisfait le critère de stabilité binaire (critère 3).

Contrairement à la continuité au sens de Scott, la stabilité d'une fonction à plusieurs arguments n'est pas équivalente à la stabilité de cette fonction sur chacune de ses composantes. Le contre exemple classique permettant d'infirmer une telle propriété est donné par la fonction calculant le *"ou" parallèle*.

Le "ou" parallèle Soit Bool l'espace cohérent défini par

$$\mathsf{Bool} \;=\; \left\{ \begin{array}{ccc} \{\mathsf{true}\} & & \{\mathsf{false}\} \\ & \diagdown\ \diagup & \\ & \varnothing & \end{array} \right\}$$

et dont la trame $|\mathsf{Bool}| = \{\mathsf{true}; \mathsf{false}\}$ est constituée de deux atomes distincts true et false, lesquels sont incohérents entre eux.

La fonction "ou" parallèle est la fonction $\mathsf{p\text{-}or} : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$ définie par la table (symétrique) suivante :

$b_1 \backslash b_2$	$\varnothing$	$\{\mathsf{false}\}$	$\{\mathsf{true}\}$
$\varnothing$	$\varnothing$	$\varnothing$	$\{\mathsf{true}\}$
$\{\mathsf{false}\}$	$\varnothing$	$\{\mathsf{false}\}$	$\{\mathsf{true}\}$
$\{\mathsf{true}\}$	$\{\mathsf{true}\}$	$\{\mathsf{true}\}$	$\{\mathsf{true}\}$

Intuitivement, la fonction "ou" parallèle calcule le "ou" logique de ses deux arguments, et renvoie systématiquement la valeur $\{\mathsf{true}\}$ dès que l'un au moins de ses deux arguments vaut $\{\mathsf{true}\}$, *y compris si l'autre argument est indéfini* (lequel est représenté par la clique vide $\varnothing$).

La fonction $\mathsf{p\text{-}or}$ est continue au sens de Scott, et est même stable sur chacune de ses composantes, puisque :

1. (stabilité sur la première composante) pour tout $b_2 \in \mathsf{Bool}$, la fonction $\mathsf{p\text{-}or}(\cdot, b_2) : \mathsf{Bool} \to \mathsf{Bool}$ est stable ;
2. (stabilité sur la seconde composante) pour tout $b_1 \in \mathsf{Bool}$, la fonction $\mathsf{p\text{-}or}(b_1, \cdot) : \mathsf{Bool} \to \mathsf{Bool}$ est stable.

En revanche, la fonction $\mathsf{p\text{-}or}$ n'est pas stable sur $\mathsf{Bool} \times \mathsf{Bool}$ puisque

$$\mathsf{p\text{-}or}\big(\{\mathsf{true}\}, \varnothing\big) \;\cap\; \mathsf{p\text{-}or}\big(\varnothing, \{\mathsf{true}\}\big) \;=\; \{\mathsf{true}\} \cap \{\mathsf{true}\} \;=\; \{\mathsf{true}\}$$

bien que

$$\mathsf{p\text{-}or}\big(\{\mathsf{true}\} \cap \varnothing, \varnothing \cap \{\mathsf{true}\}\big) \;=\; \mathsf{p\text{-}or}(\varnothing, \varnothing) \;=\; \varnothing \;\neq\; \{\mathsf{true}\}.$$

4.1.4 Espaces cohérents plats

Définition 4.1.5 (Espace cohérent plat) — Soit X un ensemble quelconque. On désigne par $X_\perp$ l'espace cohérent dont le graphe sous-jacent est défini par

1. $|X_\perp| \;=\; X$
2. Pour tous $x, y \in X$, $\quad x \frown y \pmod{X_\perp} \quad$ ssi $\quad x = y$.

Plus généralement, on dira d'un espace cohérent qu'il est *plat* si la relation de cohérence sur sa trame est *triviale*, c'est-à-dire égale à la relation d'identité. Bien entendu, tout espace cohérent plat $\mathcal{A}$ est de la forme $\mathcal{A} = X_\perp$ en prenant $X = |\mathcal{A}|$. L'espace Bool introduit dans le paragraphe précédent est un exemple d'espace cohérent plat, formé sur la paire $X = \{\mathsf{true}; \mathsf{false}\}$.

Les cliques d'un espace cohérent plat $X_\perp$ sont de l'une des deux formes suivantes :
- la clique vide $\varnothing$;
- les singletons $\{x\}$, où $x \in X$.

La terminologie d'espace cohérent *plat* provient du fait que, la clique vide mise à part, toutes les cliques de $X_\perp$ se situent au même niveau, et sont incomparables et incompatibles entre elles :

$$X_\perp \;=\; \left\{ \begin{array}{ccccccc} \{x_1\} & & \{x_2\} & & \{x_3\} & & \cdots \\ & \searrow & & \searrow\;\swarrow & & \swarrow & \\ & & & \varnothing & & & \end{array} \right\}$$

(où $X = \{x_1; x_2; x_3; \ldots\}$). Par la suite, on identifiera chaque singleton $\{x\}$ avec l'atome x lui-même, et on considérera simplement que $X_\perp = X \cup \{\varnothing\}$.

Par ailleurs, si $\mathcal{A}$ désigne un espace cohérent quelconque, toute application $f : X \to \mathcal{A}$ (au sens de la théorie des ensembles) peut être prolongée en une fonction stable $f_\perp : X_\perp \to \mathcal{A}$:

Lemme 4.1.6 (Lifting) — *Soient X un ensemble, $\mathcal{A}$ un espace cohérent et f une application de X dans $\mathcal{A}$. La fonction $f_\perp : X_\perp \to \mathcal{A}$ définie par*

$$f_\perp(c) \;=\; \begin{cases} \varnothing & \text{si } c = \varnothing \\ f(x) & \text{si } c = \{x\} \end{cases}$$

est une fonction stable.

D'après le lemme ci-dessus, il est clair que toute fonction continue de $X_\perp$ dans $\mathcal{A}$ est stable.

4.1.5 Produit direct de deux espaces cohérents

Définition 4.1.7 (Produit direct) — Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ deux espaces cohérents. On appelle *produit direct des espaces $\mathcal{A}_1$ et $\mathcal{A}_2$* et on désigne par $\mathcal{A}_1 \,\&\, \mathcal{A}_2$ l'espace cohérent dont le graphe sous-jacent est donné par

1. $|\mathcal{A}_1 \,\&\, \mathcal{A}_2| \;=\; \{1\} \times |\mathcal{A}_1| \;\cup\; \{2\} \times |\mathcal{A}_2|$;
2. $\gamma \frown \gamma' \pmod{\mathcal{A}_1 \,\&\, \mathcal{A}_2}$ si l'une des conditions suivantes est satisfaite
   - $\gamma = (1, \alpha_1)$, $\gamma' = (1, \alpha'_1)$ et $\alpha_1 \frown \alpha'_1 \pmod{\mathcal{A}_1}$;
   - $\gamma = (2, \alpha_2)$, $\gamma' = (2, \alpha'_2)$ et $\alpha_2 \frown \alpha'_2 \pmod{\mathcal{A}_2}$;
   - $\gamma = (1, \alpha_1)$, $\gamma' = (2, \alpha'_2)$;
   - $\gamma = (2, \alpha_2)$, $\gamma' = (1, \alpha'_1)$.

Intuitivement, la trame de l'espace cohérent $\mathcal{A}_1 \,\&\, \mathcal{A}_2$ est construite en recollant des copies des trames $|\mathcal{A}_1|$ et $|\mathcal{A}_2|$ (données par les ensembles $\{1\} \times |\mathcal{A}_1|$ et $\{2\} \times |\mathcal{A}_2|$ respectivement). La relation de cohérence correspondante est obtenue en transportant sur $|\mathcal{A}_1 \,\&\, \mathcal{A}_2|$ les arêtes provenant des relations de cohérence des espaces $\mathcal{A}_1$ et $\mathcal{A}_2$, auxquelles on a ajouté de nouvelles arêtes de manière à relier chaque atome provenant de $|\mathcal{A}_1|$ à chaque atome provenant de $|\mathcal{A}_2|$.

L'espace $\mathcal{A}_1 \times \mathcal{A}_2$ est un *produit cartésien* au sens de la théorie des catégories. En particulier, la fonction

$$(a_1, a_2) \quad \mapsto \quad \{1\} \times a_1 \;\cup\; \{2\} \times a_2$$

est une bijection de $\mathcal{A}_1 \times \mathcal{A}_2$ dans $\mathcal{A}_1 \,\&\, \mathcal{A}_2$, et même un isomorphisme au sens des structures de domaine de Scott sous-jacentes. L'intérêt de travailler avec le produit direct $\mathcal{A}_1 \,\&\, \mathcal{A}_2$ plutôt qu'avec le produit cartésien ensembliste $\mathcal{A}_1 \times \mathcal{A}_2$ est que le premier ensemble a naturellement une structure d'espace cohérent, ce qui n'est pas le cas du second.[4]

Dans la définition de l'espace produit, l'introduction des ensembles $\{1\} \times |\mathcal{A}_1|$ et $\{2\} \times |\mathcal{A}_2|$ pour définir la trame de $\mathcal{A}_1 \,\&\, \mathcal{A}_2$ ne sert qu'à s'assurer du fait que les trames des images de $|\mathcal{A}_1|$ et de $|\mathcal{A}_2|$ sont disjointes dans la trame $|\mathcal{A}_1 \,\&\, \mathcal{A}_2|$. Dans le cas où $|\mathcal{A}_1|$ et $|\mathcal{A}_2|$ sont déjà des ensembles disjoints, on évitera de recourir à un tel artifice en posant simplement

$$|\mathcal{A}_1 \,\&\, \mathcal{A}_2| \;=\; |\mathcal{A}_1| \cup |\mathcal{A}_2|\,.$$

L'existence du produit cartésien catégorique $\mathcal{A}_1 \,\&\, \mathcal{A}_2$ pour chaque paire d'espaces $\mathcal{A}_1$ et $\mathcal{A}_2$ confère à la catégorie des espaces cohérents une structure de *catégorie cartésienne*. Par ailleurs, la définition de la notion de fonction stable à plusieurs arguments est cohérente avec la définition du produit direct $\mathcal{A}_1 \,\&\, \mathcal{A}_2$:

Proposition 4.1.8 — *Soient $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{B}$ des espaces cohérents. Une fonction (à un argument) $F : \mathcal{A}_1 \,\&\, \mathcal{A}_2 \to \mathcal{B}$ est stable si et seulement si la fonction (à deux arguments) $F' : \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{B}$ définie pour tout $(a_1, a_2) \in \mathcal{A}_1 \times \mathcal{A}_2$ par*

$$F'(a_1, a_2) \;=\; F\big(\{1\} \times a_1 \cup \{2\} \times a_2\big)$$

est stable sur $\mathcal{A}_1 \times \mathcal{A}_2$.

4.1.6 Somme directe de deux espaces cohérents

Définition 4.1.9 (Somme directe) — Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ deux espaces cohérents. On appelle *somme directe des espaces $\mathcal{A}_1$ et $\mathcal{A}_2$* et on désigne par $\mathcal{A}_1 \oplus \mathcal{A}_2$ l'espace cohérent dont le graphe sous-jacent est donné par

1. $|\mathcal{A}_1 \oplus \mathcal{A}_2| \;=\; \{1\} \times |\mathcal{A}_1| \;\cup\; \{2\} \times |\mathcal{A}_2|$;
2. $\gamma \subset \gamma' \pmod{\mathcal{A}_1 \oplus \mathcal{A}_2}$ si l'une des conditions suivantes est satisfaite
   - $\gamma = (1, \alpha_1)$, $\gamma' = (1, \alpha_1')$ et $\alpha_1 \subset \alpha_1' \pmod{\mathcal{A}_1}$;
   - $\gamma = (2, \alpha_2)$, $\gamma' = (2, \alpha_2')$ et $\alpha_2 \subset \alpha_2' \pmod{\mathcal{A}_2}$.

[4] La notation $A_1 \,\&\, \mathcal{A}_2$ provient de la logique linéaire, du fait que le produit direct sert à interpréter habituellement le connecteur & ("avec").

Comme pour le produit direct, la trame de la somme directe est constituée par la réunion de deux copies disjointes des trames des espaces $\mathcal{A}_1$ et $\mathcal{A}_2$. En revanche, la relation de cohérence sur $|\mathcal{A}_1 \oplus \mathcal{A}_2|$ est constituée uniquement des arêtes provenant des relations de cohérence issues des espaces $\mathcal{A}_1$ et $\mathcal{A}_2$, les atomes provenant de l'un étant incohérents avec les atomes issus de l'autre dans la somme directe $\mathcal{A}_1 \oplus \mathcal{A}_2$. Par conséquent, une clique c appartenant à l'espace cohérent $\mathcal{A}_1 \oplus \mathcal{A}_2$ est :

- soit de la forme $c = \{1\} \times a_1$ avec $a_1 \in \mathcal{A}_1$,
- soit de la forme $c = \{2\} \times a_2$ avec $a_2 \in \mathcal{A}_2$,

avec le cas particulier de la clique vide qui est simultanément des deux formes puisque

$$\varnothing \;=\; \{1\} \times \varnothing \;=\; \{2\} \times \varnothing.$$

Plus formellement on a le résultat suivant :

Lemme 4.1.10 — *Les applications $i_1 : \mathcal{A}_1 \to \mathcal{A}_1 \oplus \mathcal{A}_2$ et $i_2 : \mathcal{A}_2 \to \mathcal{A}_1 \oplus \mathcal{A}_2$ définies pour tous $a_1 \in \mathcal{A}_1$ et $a_2 \in \mathcal{A}_2$ par*

$$\begin{array}{rcl} i_1(a_1) & = & \{1\} \times a_1, \\ i_2(a_2) & = & \{2\} \times a_2, \end{array}$$

sont des fonctions stables injectives. Toute clique $c \in \mathcal{A}_1 \oplus \mathcal{A}_2$ est

- *ou bien de la forme $c = i_1(a_1)$ (pour une clique $a_1 \in \mathcal{A}_1$),*
- *ou bien de la forme $c = i_2(a_2)$ (pour une clique $a_2 \in \mathcal{A}_2$).*

De plus, on a pour tous $a_1 \in \mathcal{A}_1$ et $a_2 \in \mathcal{A}_2$:

$$i_1(a_1) = i_2(a_2) \qquad \Leftrightarrow \qquad a_1 = a_2 = \varnothing.$$

Intuitivement, la somme directe $\mathcal{A}_1 \oplus \mathcal{A}_2$ correspond donc à l'union disjointe des espaces cohérents $\mathcal{A}_1$ et $\mathcal{A}_2$ dans laquelle on a identifié la clique vide de $\mathcal{A}_1$ à la clique vide de $\mathcal{A}_2$. Par ailleurs, dans le cas où les trames $|\mathcal{A}_1|$ et $|\mathcal{A}_2|$ sont déjà des ensembles disjoints, on définira plus simplement la trame de la somme directe $\mathcal{A}_1 \oplus \mathcal{A}_2$ en posant directement

$$|\mathcal{A}_1 \oplus \mathcal{A}_2| \;=\; |\mathcal{A}_1| \cup |\mathcal{A}_2|$$

et en définissant la relation de cohérence sur $|\mathcal{A}_1 \oplus \mathcal{A}_2|$ de manière analogue.

4.1.7 Espace des fonctions stables

Trace d'une fonction stable Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents. Au paragraphe 4.1.5, nous avons vu que le produit cartésien ensembliste $\mathcal{A} \times \mathcal{B}$ n'a pas une structure d'espace cohérent, d'où la nécessité d'introduire un espace cohérent $\mathcal{A}\&\mathcal{B}$ qui lui soit isomorphe en tant que domaine de Scott. Le même problème survient à nouveau pour l'ensemble des fonctions stables, lequel constitue un domaine de Scott (pour l'ordre de Berry que nous introduirons plus loin), mais pas un espace cohérent. Afin de remédier à ce problème, nous devons donc changer la représentation des fonctions stables, de la même façon que nous avons dû changer la représentation des couples pour construire $\mathcal{A} \,\&\, \mathcal{B}$. Ce changement de représentation s'effectue au moyen de la notion de trace :

Définition 4.1.11 (Trace) — Soient $\mathcal{A}$, $\mathcal{B}$ des espaces cohérents, et $F : \mathcal{A} \to \mathcal{B}$ une fonction stable. On appelle *trace* de F et on note $\mathsf{Tr}(F)$ l'ensemble des couples $(a_0, \beta) \in \mathcal{A}_{\mathbf{fin}} \times |\mathcal{B}|$ tels que

1. $\beta \in F(a_0)$;
2. Pour tout $a_1 \subsetneq a_0$, on a $\beta \notin F(a_1)$.

Afin de comprendre l'intuition qui se cache derrière cette définition, considérons un élément (a_0, β) de la trace d'une fonction stable F, où $a_0 = \{\alpha_1; \ldots; \alpha_n\}$ est une clique finie. Comme F est monotone, la première condition ne signifie pas seulement que $\beta \in F(a_0)$, mais plus généralement que $\beta \in F(a)$ pour n'importe quelle clique $a \in \mathcal{A}$ telle que $\alpha_1, \ldots, \alpha_n \in a$. De plus, la seconde condition exprime la *minimalité* de a_0 vis-à-vis de la première condition. Intuitivement, la condition $(a_0, \beta) \in \mathsf{Tr}(F)$ exprime donc que

> *Si on donne à F au moins tous les atomes $\alpha_1, \ldots, \alpha_n \in a_0$, alors F produit au moins l'atome β, et il n'existe aucun sous-ensemble strict de la clique finie $a_0 = \{\alpha_1; \ldots; \alpha_n\}$ permettant d'assurer cette condition.*

Remarquons par ailleurs que pour n'importe quelle clique $a \in \mathcal{A}$ et pour n'importe quel atome $\beta \in F(a)$, il existe une clique finie $a_0 \subset a$ tel que $(a_0, \beta) \in \mathsf{Tr}(F)$. En effet, le critère de continuité entraîne que

$$\beta \in F(a) = \bigcup_{a_0 \in \Im(a)} F(a_0),$$

d'où découle immédiatement l'existence d'une clique finie $a_0 \subset a$ telle que $\beta \in F(a_0)$. Quitte à choisir une telle clique finie a_0 avec le cardinal minimal, on a bien $(a_0, \beta) \in \mathsf{Tr}(F)$.

Cette remarque a deux conséquences. La première est que la trace d'une fonction stable F suffit à caractériser l'image par F de n'importe quelle clique de $\mathcal{A}$.

Proposition 4.1.12 (Application) — *Soit $F : \mathcal{A} \to \mathcal{B}$ une fonction stable. Pour toute clique $a \in \mathcal{A}$ on a*

$$F(a) = \{\beta \in |\mathcal{B}| \,; \quad \exists a_0 \in \Im(a) \quad (a_0, \beta) \in \mathsf{Tr}(F)\} \tag{4.1}$$

La seconde conséquence de cette remarque, qui découle immédiatement de la première, est que les fonctions stables sont caractérisées par leur trace.

Corollaire 4.1.13 — *Soient F et G deux fonctions stables de $\mathcal{A}$ dans $\mathcal{B}$. On a :*

$$F = G \qquad \Leftrightarrow \qquad \mathsf{Tr}(F) = \mathsf{Tr}(G).$$

Ce résultat nous autorisera à identifier par la suite chaque fonction stable $F : \mathcal{A} \to \mathcal{B}$ avec sa trace $\mathsf{Tr}(F) \subset \mathcal{A}_{\mathbf{fin}} \times |\mathcal{B}|$.

Espace des fonctions stables La caractérisation des fonctions stables par leur trace nous permet de définir l'*espace des fonctions stables de $\mathcal{A}$ dans $\mathcal{B}$*, non pas comme l'ensemble des fonctions stables de $\mathcal{A}$ dans $\mathcal{B}$, mais comme l'ensemble de leurs traces :

Définition 4.1.14 (Espace $\mathcal{A} \to \mathcal{B}$) — Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents. On désigne par $\mathcal{A} \to \mathcal{B}$ l'ensemble des traces des fonctions stables de $\mathcal{A}$ dans $\mathcal{B}$:

$$\mathcal{A} \to \mathcal{B} \quad = \quad \{\mathsf{Tr}(F); \quad F : \mathcal{A} \to \mathcal{B} \text{ stable}\}.$$

Cette définition basée sur la représentation des fonctions stables par leurs traces confère à l'espace $\mathcal{A} \to \mathcal{B}$ une structure d'espace cohérent :

Proposition 4.1.15 — *L'espace des fonctions stable $\mathcal{A} \to \mathcal{B}$ est un espace cohérent, dont la trame est donnée par*

$$|\mathcal{A} \to \mathcal{B}| \quad = \quad \mathcal{A}_{\mathbf{fin}} \times |\mathcal{B}|$$

et dont la relation de cohérence

$$(a_1, \beta_1) \frown (a_2, \beta_2) \pmod{\mathcal{A} \to \mathcal{B}}$$

est caractérisée par la conjonction des deux conditions suivantes

1. *Si $a_1 \cup a_2 \in \mathcal{A}$, alors $\beta_1 \frown \beta_2$ (mod $\mathcal{B}$).*
2. *Si $a_1 \cup a_2 \in \mathcal{A}$ et $a_1 \neq a_2$, alors $\beta_1 \neq \beta_2$.*

Les atomes de l'espace cohérent $\mathcal{A} \to \mathcal{B}$ sont les couples (a_0, β) constitués d'une clique finie de $\mathcal{A}$ et d'un atome de $\mathcal{B}$. Chacun de ces atomes $(a_0, \beta) \in \mathcal{A}_{\mathbf{fin}} \times |\mathcal{B}|$ représente une fonction stable élémentaire $\mathsf{step}_{a_0,\beta}$ définie par

$$\mathsf{step}_{a_0,\beta}(a) \quad = \quad \begin{cases} \{\beta\} & \text{si } a_0 \subset a \\ \varnothing & \text{sinon} \end{cases}$$

et dont la trace $\mathsf{Tr}(\mathsf{step}_{a_0,\beta})$ est réduite au singleton $\{(a_0, \beta)\}$.

Les deux conditions caractérisant la cohérence de deux atomes $(a_1, \beta_1), (a_2, \beta_2) \in |\mathcal{A} \times \mathcal{B}|$ (c'est-à-dire la cohérence de deux fonctions stables élémentaires $\mathsf{step}_{a_1,\beta_1}$ et $\mathsf{step}_{a_2,\beta_2}$) sont un peu plus délicates à comprendre.

La première de ces conditions exprime le fait que l'image d'une clique de $\mathcal{A}$ par une fonction stable doit être une clique de $\mathcal{B}$. En effet, si $(a_1, \beta_1) \frown (a_2, \beta_2)$ (mod $\mathcal{A} \to \mathcal{B}$), il existe une fonction stable F dont la trace est précisément la paire constituée par ces deux atomes :

$$\mathsf{Tr}(F) \;=\; \{(a_1, \beta_1);\ (a_2, \beta_2)\}.$$

Dans le cas où $a_1 \cup a_2 \in \mathcal{A}$ on a d'après (4.1)

$$F(a_1 \cup a_2) \;=\; \{\beta_1; \beta_2\}$$

d'où il ressort que les atomes β_1 et β_2 doivent être cohérents dans $\mathcal{B}$ de manière à ce que l'écriture ci-dessus ait un sens.

La seconde condition est une conséquence de la minimalité de la première composante a_0 du couple $(a_0, \beta) \in \mathsf{Tr}(F)$ vis-à-vis de la condition $\beta \in F(a_0)$ (*cf* la définition de la trace), elle-même intimement liée à la notion de stabilité.

En effet, si F désigne la fonction stable dont la trace est la paire $\{(a_1, \beta_1); (a_2, \beta_2)\}$ (ce qui suppose les atomes (a_1, β_1) et (a_2, β_2) cohérents dans $\mathcal{A} \to \mathcal{B}$), on a

$$F(a_1 \cap a_2) \;=\; F(a_1) \cap F(a_2)$$

dès que $a_1 \cup a_2 \in \mathcal{A}$ (d'après le critère de stabilité binaire). Si l'on suppose en outre que $\beta_1 = \beta_2 = \beta$, il vient

$$F(a_1 \cap a_2) \;=\; F(a_1) \cap F(a_2) \;=\; \{\beta\} \cap \{\beta\} \;=\; \{\beta\}$$

d'où il ressort que $\beta \in F(a_1 \cap a_2)$. Mais comme $\beta = \beta_1 \in F(a_1)$ et $\beta = \beta_2 \in F(a_2)$, il est clair que par définition de la trace on a $a_1 \cap a_2 = a_1$ et $a_1 \cap a_2 = a_2$, d'où $a_1 = a_2$ (en vertu de la condition de minimalité imposée aux cliques a_1 et a_2 vis-à-vis des conditions $\beta \in F(a_1)$ et $\beta \in F(a_2)$).

Nous avons donc démontré que dans le cas où les cliques a_1 et a_2 sont compatibles et où les atomes β_1 et β_2 sont égaux, la cohérence des atomes fonctionnels (a_1, β_1) et (a_2, β_2) entraîne l'égalité des cliques a_1 et a_2, ce qu'exprime exactement la seconde condition de cohérence donnée par la proposition précédente, modulo une légère reformulation par contraposition.

Proposition 4.1.16 (Stabilité de l'application) — *La fonction à deux arguments*

$$\begin{array}{rccc} \mathsf{App} \;:\; & (\mathcal{A} \to \mathcal{B}) \times \mathcal{A} & \longrightarrow & \mathcal{B} \\ & (F, a) & \longmapsto & F(a) \end{array}$$

est une fonction stable (globalement) sur le produit cartésien $(\mathcal{A} \to \mathcal{B}) \times \mathcal{A}$.

Plus généralement, l'espace des fonctions stables $\mathcal{A} \to B$ est une *exponentielle* au sens de la théorie des catégories, d'où il ressort que :

Proposition 4.1.17 (Catégorie cartésienne fermée) — *La catégorie des espaces cohérents est une catégorie cartésienne fermée.*

Ordre extensionnel et ordre stable Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérent. L'espace des fonctions stables $\mathcal{A} \to \mathcal{B}$ est naturellement muni de deux relations d'ordre différentes, qui sont

1. L'ordre *extensionnel* — encore appelé *ordre par points* — défini par :

$$F \leq_{\mathbf{ext}} G \qquad \equiv \qquad \forall\, a \in \mathcal{A} \quad F(a) \subset G(a).$$

2. L'ordre *stable*, induit par l'inclusion de traces :

$$F \leq_{\mathbf{st}} G \qquad \equiv \qquad \mathsf{Tr}(F) \subset \mathsf{Tr}(G)$$

Ces deux relations d'ordre ne sont pas équivalentes : l'ordre stable est plus fort que l'ordre extensionnel

$$\mathsf{Tr}(F) \subset \mathsf{Tr}(G) \qquad \Rightarrow \qquad \forall\, a \in \mathcal{A} \quad F(a) \subset G(a)$$

mais n'est en général pas équivalent (nous en verrons un contre-exemple simple dans le sous-paragraphe qui suit). En revanche, l'ordre stable est équivalent à l'*ordre de Berry*, qui est la relation d'ordre notée $F \leq_{\mathbf{B}} G$ et définie par

$$F \leq_{\mathbf{B}} G \qquad \equiv \qquad \big(\forall a, a' \in \mathcal{A} \qquad a' \subset a \quad \Rightarrow \quad F(a') = F(a) \cap G(a')\big),$$

laquelle constitue clairement une relation d'ordre plus forte que la relation d'ordre par points.

Proposition 4.1.18 (Caractérisation de l'ordre stable) — *Si F et G sont deux fonctions stables de $\mathcal{A}$ dans $\mathcal{B}$, alors*

$$\begin{array}{lll} \mathsf{Tr}(F) \subset \mathsf{Tr}(G) & \Leftrightarrow & F \leq_{\mathbf{B}} G \\ & \Leftrightarrow & \big(\forall\, a, a' \in \mathcal{A} \qquad a' \subset a \quad \Rightarrow \quad F(a') = F(a) \cap G(a')\big). \end{array}$$

Fonctions constantes par vocation et fonctions constantes calculées Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents, et b une clique arbitraire de $\mathcal{B}$. Il existe deux façons de construire une fonction constante retournant la clique b.

1. La *fonction constante par vocation*, notée F_v et définie pour toute clique $a \in \mathcal{A}$ par

$$F_v(a) \quad = \quad b$$

2. La fonction *constante calculée*, ou encore *constante par hasard*, notée F_c et définie par

$$F_c(a) \quad = \quad \begin{cases} b & \text{si } a \neq \varnothing \\ \varnothing & \text{si } a = \varnothing \end{cases}$$

La fonction constante par vocation correspond à un programme qui retourne toujours le même résultat sans même consulter son argument, comme par exemple dans le programme

```
fun (x : int) -> 0
```

Au contraire, la fonction constante calculée utilise son argument, mais les calculs qu'elle effectue avec cet argument aboutissent toujours au même résultat, comme par exemple dans le programme[5]

```
fun (x : int) -> (x - x)
```

Contrairement à la fonction constante par vocation, le résultat de la fonction constante calculée est indéfini si son argument est indéfini. Ces deux fonctions sont des fonctions stables et, du point de vue de l'ordre extensionnel, la fonction constante calculée est plus petite que la fonction constante par vocation correspondante :

$$F_c \leq_{\mathbf{ext}} F_v.$$

En revanche, ces deux fonctions ne sont pas comparables du point de vue de l'ordre stable dans le cas où $b \neq \varnothing$, puisqu'il n'y a aucune inclusion entre leurs traces respectives :

$$\begin{array}{lll} \mathsf{Tr}(F_v) & = & \big\{(\varnothing, \beta); \quad \beta \in b\big\} \\ \mathsf{Tr}(F_c) & = & \big\{(\{\alpha\}, \beta); \quad \alpha \in |\mathcal{A}| \text{ et } \beta \in b\big\}. \end{array}$$

En outre, ces deux fonctions sont non seulement incomparables vis-à-vis de l'ordre stable, mais également incompatibles dans l'espace cohérent des fonctions stables, puisque

$$\mathsf{Tr}(F_v) \cup \mathsf{Tr}(F_c) \quad \notin \quad \mathcal{A} \to \mathcal{B}.$$

[5]Pour que cette comparaison ait un sens, il faut évidemment se placer dans un langage soumis à la discipline de l'appel par nom, tel que Haskell par exemple. Dans un langage basé sur l'appel par valeur (tel que SML ou Caml), toutes les fonctions sont strictes (*i.e.* $F(\varnothing) = \varnothing$) et les seules fonctions constantes qu'il est possible d'exprimer sont des constantes calculées.

4.1.8 Fonctions linéaires

Définition 4.1.19 — Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents. Un fonction $F : \mathcal{A} \to \mathcal{B}$ est dite *linéaire* si elle est stable et si sa trace n'est composée que de couples (a_0, β) dans lesquels la clique finie a_0 est réduite à un atome.

Les fonctions constantes calculées introduites au paragraphe précédent constituent un exemple simple de fonctions linéaires. Par ailleurs, la fonction identité $\mathsf{id}_{\mathcal{A}} : \mathcal{A} \to \mathcal{A}$ est également une fonction linéaire, dont la trace est donnée par

$$\mathsf{Tr}(\mathsf{id}_{\mathcal{A}}) \;=\; \big\{(\{\alpha\}, \alpha);\quad \alpha \in |\mathcal{A}|\big\}.$$

Puisque les atomes de la trace d'une fonction linéaire sont toujours de la forme $(\{\alpha\}, \beta)$, il est naturel de simplifier l'écriture de la trace d'une telle fonction, en introduisant la notion de *trace linéaire*.

Définition 4.1.20 — Soit $F : \mathcal{A} \to \mathcal{B}$ une fonction linéaire. On appelle *trace linéaire* de F et on note $\mathsf{TrLin}(F)$ l'ensemble défini par

$$\begin{array}{rcl} \mathsf{TrLin}(F) & = & \big\{(\alpha, \beta) \in |\mathcal{A}| \times |\mathcal{B}|\,;\quad (\{\alpha\}, \beta) \in \mathsf{Tr}(F)\big\} \\ & = & \big\{(\alpha, \beta) \in |\mathcal{A}| \times |\mathcal{B}|\,;\quad \beta \in F(\{\alpha\})\big\} \end{array}$$

Proposition 4.1.21 — *L'ensemble des traces linéaires des fonctions linéaires $F : \mathcal{A} \to \mathcal{B}$ constitue un espace cohérent noté $\mathcal{A} \multimap \mathcal{B}$, dont la trame est*

$$|\mathcal{A} \multimap \mathcal{B}| \quad = \quad |\mathcal{A}| \times |\mathcal{B}|$$

et dont la relation de cohérence

$$(\alpha_1, \beta_1) \frown (\alpha_2, \beta_2) \pmod{\mathcal{A} \multimap \mathcal{B}}$$

est caractérisée par la conjonction des deux conditions suivantes

1. *Si* $\alpha_1 \frown \alpha_2 \pmod{\mathcal{A}}$, *alors* $\beta_1 \frown \beta_2 \pmod{\mathcal{B}}$.
2. *Si* $\alpha_1 \frown \alpha_2 \pmod{\mathcal{A}}$ *et* $\alpha_1 \neq \alpha_2$, *alors* $\beta_1 \neq \beta_2$.

Nous ne poursuivrons pas davantage cette présentation de l'espace des fonctions linéaires, et nous n'aborderons pas la question de la décomposition de l'espace des fonctions stables $\mathcal{A} \to \mathcal{B}$ sous la forme

$$\mathcal{A} \to \mathcal{B} \quad = \quad \mathcal{A}! \multimap \mathcal{B} \quad = \quad (\mathcal{A}!)^{\perp} \parr \mathcal{B} \quad = \quad (\mathcal{A}^{\perp})? \parr \mathcal{B}$$

qui nous conduirait vers une étude de la sémantique cohérente de la logique linéaire, laquelle dépasse le cadre de cette thèse.

4.2 La catégorie des plongements rigides

4.2.1 Plongements rigides

Définition 4.2.1 (Plongements rigides) — Soient $\mathcal{A}$ et $\mathcal{A}'$ deux espaces cohérents. On appelle *plongement rigide de $\mathcal{A}$ dans $\mathcal{A}'$* toute injection $\phi : |\mathcal{A}| \to |\mathcal{A}'|$ telle que pour tous $\alpha_1, \alpha_2 \in |\mathcal{A}|$ on ait

$$\alpha_1 \frown \alpha_2 \pmod{\mathcal{A}} \quad \Leftrightarrow \quad \phi(\alpha_1) \frown \phi(\alpha_2) \pmod{\mathcal{A}'}.$$

Lorsque ϕ est un plongement rigide de $\mathcal{A}$ dans $\mathcal{A}'$, on le note $\phi : \mathcal{A} \rightarrowtail \mathcal{A}'$.

Un plongement rigide $\phi : |\mathcal{A}| \to |\mathcal{A}'|$ induit naturellement deux fonctions $\phi^+ : \mathcal{A} \to \mathcal{A}'$ et $\phi^- : \mathcal{A}' \to \mathcal{A}$ appelées respectivement *plongement* et *rétraction* associées à ϕ. Ces deux fonctions sont définies par

$$\begin{array}{llcll} \forall\, a \in \mathcal{A} & \phi^+(a) & = & \{\phi(\alpha);\quad \alpha \in a\} & \text{(image directe)} \\ \forall\, a' \in \mathcal{A}' & \phi^-(a') & = & \{\alpha;\quad \phi(\alpha) \in a'\} & \text{(image réciproque)} \end{array}$$

Ces deux fonctions sont des fonctions linéaires dont les traces linéaires sont données par

$$\begin{array}{lcl} \mathsf{TrLin}(\phi^+) & = & \{(\alpha, \phi(\alpha));\quad \alpha \in |\mathcal{A}|\}; \\ \mathsf{TrLin}(\phi^-) & = & \{(\phi(\alpha), \alpha);\quad \alpha \in |\mathcal{A}|\}. \end{array}$$

Les fonctions $\phi^+ : \mathcal{A} \multimap \mathcal{A}'$ et $\phi^- : \mathcal{A}' \multimap \mathcal{A}$ sont en outre reliées par les relations suivantes :

$$\phi^- \circ \phi^+ \;=\; \mathsf{id}_{\mathcal{A}} \qquad \text{et} \qquad \phi^+ \circ \phi^- \;\leq_{\mathbf{st}}\; \mathsf{id}_{\mathcal{A}'}.$$

(Remarquons que ϕ^+ est injective, et que ϕ^- est surjective.)

Définition 4.2.2 (Inclusion rigide) — Soient $\mathcal{A}$ et $\mathcal{A}'$ deux espaces cohérents. On dit que $\mathcal{A}$ *est rigidement inclus dans $\mathcal{A}'$*, ou encore que $\mathcal{A}$ est un *sous-espace cohérent* de $\mathcal{A}'$, si $|\mathcal{A}| \subset |\mathcal{A}'|$ et si les relations de cohérence issues de $\mathcal{A}$ et de $\mathcal{A}'$ coïncident sur $\mathcal{A}$, c'est-à-dire

$$\alpha_1 \frown \alpha_2 \pmod{\mathcal{A}} \quad \Leftrightarrow \quad \alpha_1 \frown \alpha_2 \pmod{\mathcal{A}'}$$

pour tous $\alpha_1, \alpha_2 \in |\mathcal{A}|$. Lorsque $\mathcal{A}$ est rigidement inclus dans $\mathcal{A}'$, on le note $\mathcal{A} \Subset \mathcal{A}'$.

De manière équivalente, un espace cohérent $\mathcal{A}$ est rigidement inclus dans un espace cohérent $\mathcal{A}'$ si et seulement si la trame de $\mathcal{A}$ est incluse dans la trame de $\mathcal{A}'$ et si l'injection canonique $\phi : |\mathcal{A}| \to |\mathcal{A}'|$ envoyant chaque atome de $\mathcal{A}$ sur lui-même constitue un plongement rigide. Dans ce cas particulier, le plongement $\phi^+ : \mathcal{A} \multimap \mathcal{A}'$ et la rétraction $\phi^- : \mathcal{A}' \multimap \mathcal{A}$ associés à l'inclusion rigide $\mathcal{A} \Subset \mathcal{A}'$ sont donnés par

$$\begin{array}{llcll} \forall\, a \in \mathcal{A} & \phi^+(a) & = & a & \text{(inclusion de } \mathcal{A} \text{ dans } \mathcal{A}') \\ \forall\, a' \in \mathcal{A}' & \phi^-(a') & = & a' \cap |\mathcal{A}| & \text{(restriction de } \mathcal{A}' \text{ vers } \mathcal{A}) \end{array}$$

Par ailleurs, tout plongement rigide $\phi : |\mathcal{A}| \rightarrowtail |\mathcal{A}'|$ peut être vu comme une inclusion rigide, simplement en identifiant chaque atome $\alpha \in |\mathcal{A}|$ avec son image $\phi(\alpha) \in |\mathcal{A}'|$. Nous verrons plus loin qu'un tel point de vue est justifié en pratique par le fait que les inclusions rigides sur des espaces élémentaires induisent naturellement des inclusions rigides sur les structures composées telles que le produit direct $\mathcal{A} \,\&\, \mathcal{B}$, la somme directe $\mathcal{A} \oplus \mathcal{B}$, l'espace des fonctions stables $\mathcal{A} \to \mathcal{B}$ ou encore l'espace des fonctions linéaires $\mathcal{A} \multimap \mathcal{B}$.

Lien avec les couples plongement-rétraction Nous avons vu que tout plongement rigide $\phi : |\mathcal{A}| \mapsto |\mathcal{A}'|$ induit naturellement un *couple plongement-rétraction* de $\mathcal{A}$ vers $\mathcal{A}'$, c'est-à-dire un couple (ϕ^+, ϕ^-) formé de deux fonctions stables $\phi^+ : \mathcal{A} \to \mathcal{A}'$ et $\phi^- : \mathcal{A}' \to \mathcal{A}$ telles que

$$\phi^- \circ \phi^+ \;=\; \mathsf{id}_{\mathcal{A}} \qquad \text{et} \qquad \phi^+ \circ \phi^- \;\leq_{\mathbf{st}}\; \mathsf{id}_{\mathcal{A}'}.$$

Cette notion de couple plongement-rétraction dans les espaces cohérents est l'analogue de la notion de couple plongement-rétraction dans les domaines de Scott, à ceci près que le plongement ϕ^+ et la rétraction ϕ^- sont des fonctions stables, et que la majoration

$$\phi^+ \circ \phi^- \;\leq_{\mathbf{st}}\; \mathsf{id}_{\mathcal{A}'}$$

concerne l'ordre stable et non l'ordre extensionnel.

Dans les domaines de Scott, la notion de couple plongement-rétraction est fondamentale pour résoudre les équations récursives de domaines par construction de point fixe, dont les solutions fournissent la base des modèles du λ-calcul par la sémantique dénotationnelle. Cependant, cette notion est généralement d'un emploi délicat, et la nécessité de travailler simultanément avec deux applications a une fâcheuse tendance à obscurcir les détails de la construction, et rend généralement illisibles les domaines obtenus par passage à la colimite des itérés du foncteur à partir duquel ces solutions sont généralement construites.

L'intérêt de travailler avec des espaces cohérents (et plus généralement avec des domaines concrets) plutôt qu'avec des domaines de Scott est que dans les premiers, la notion de couple plongement-rétraction est beaucoup plus simple, car elle se ramène systématiquement à la notion de plongement rigide, qui peut toujours être considérée comme une simple inclusion rigide à un renommage d'atomes près.

Proposition 4.2.3 (Caractérisation des couples plongement-rétraction) —*Soient $\mathcal{A}$ et $\mathcal{A}'$ des espaces cohérents. Si $F^+ : \mathcal{A} \to \mathcal{A}'$ et $F^- : \mathcal{A}' \to \mathcal{A}$ sont des fonctions stables telles que*

$$F^- \circ F^+ \;=\; \mathsf{id}_{\mathcal{A}} \qquad \textit{et} \qquad F^+ \circ F^- \;\leq_{\mathbf{st}}\; \mathsf{id}_{\mathcal{A}'},$$

alors il existe un plongement rigide $\phi : |\mathcal{A}| \to |\mathcal{A}'|$ telle que $F^+ = \phi^+$ et $F^- = \phi^-$.

Précisons que la terminologie de *plongement rigide* indique qu'une telle application ne se contente pas seulement de transporter les cliques du sous-espace vers le sur-espace (comme c'est le cas dans le cadre général des domaines abstraits), mais qu'elle envoie également les atomes du premier vers les atomes du second (de manière injective), et que la relation de cohérence est fidèlement transportée par cette application.

Plongements rigides et structures composées Nous allons montrer que les plongements rigides s'étendent naturellement aux structures composées telles que le produit direct $\mathcal{A} \,\&\, \mathcal{B}$, la somme directe $\mathcal{A} \oplus \mathcal{B}$, l'espace des fonctions stables $\mathcal{A} \to \mathcal{B}$ et l'espace des fonctions linéaires $\mathcal{A} \multimap \mathcal{B}$. Pour cela, considérons quatre espaces cohérents $\mathcal{A}$, $\mathcal{A}'$, $\mathcal{B}$ et $\mathcal{B}'$.

Proposition 4.2.4 (Composition des plongements rigides) — *Si $\phi_{\mathcal{A}}$ et $\phi_{\mathcal{B}}$ sont des plongements rigides de $\mathcal{A}$ dans $\mathcal{A}'$ et de $\mathcal{B}$ dans $\mathcal{B}'$ respectivement, alors*

- *L'application* $\phi_{\mathcal{A}\&\mathcal{B}} : |\mathcal{A} \,\&\, \mathcal{B}| \to |\mathcal{A}' \,\&\, \mathcal{B}'|$ *définie par*

$$\phi_{\mathcal{A}\&\mathcal{B}}((1,\alpha)) \;=\; (1, \phi_{\mathcal{A}}(\alpha)) \qquad \textit{et} \qquad \phi_{\mathcal{A}\&\mathcal{B}}((2,\beta)) \;=\; (2, \phi_{\mathcal{B}}(\beta))$$

constitue un plongement rigide de $\mathcal{A} \,\&\, \mathcal{B}$ *dans* $\mathcal{A}' \,\&\, \mathcal{B}'$*.*

- *L'application* $\phi_{\mathcal{A}\oplus\mathcal{B}} : |\mathcal{A} \oplus \mathcal{B}| \to |\mathcal{A}' \oplus \mathcal{B}'|$ *définie par*

$$\phi_{\mathcal{A}\oplus\mathcal{B}}((1,\alpha)) \;=\; (1, \phi_{\mathcal{A}}(\alpha)) \qquad \textit{et} \qquad \phi_{\mathcal{A}\oplus\mathcal{B}}((2,\beta)) \;=\; (2, \phi_{\mathcal{B}}(\beta))$$

constitue un plongement rigide de $\mathcal{A} \oplus \mathcal{B}$ *dans* $\mathcal{A}' \oplus \mathcal{B}'$*.*

- *L'application* $\phi_{\mathcal{A}\to\mathcal{B}} : |\mathcal{A} \to \mathcal{B}| \to |\mathcal{A}' \to \mathcal{B}'|$ *définie par*

$$\phi_{\mathcal{A}\to\mathcal{B}}((a_0,\beta)) \;=\; \big(\phi^+_{\mathcal{A}}(a_0), \phi_{\mathcal{B}}(\beta)\big)$$

constitue un plongement rigide de $\mathcal{A} \to \mathcal{B}$ *dans* $\mathcal{A}' \to \mathcal{B}'$*.*

- *L'application* $\phi_{\mathcal{A}\multimap\mathcal{B}} : |\mathcal{A} \multimap \mathcal{B}| \to |\mathcal{A}' \multimap \mathcal{B}'|$ *définie par*

$$\phi_{\mathcal{A}\multimap\mathcal{B}}((\alpha,\beta)) \;=\; \big(\phi_{\mathcal{A}}(\alpha), \phi_{\mathcal{B}}(\beta)\big)$$

constitue un plongement rigide de $\mathcal{A} \multimap \mathcal{B}$ *dans* $\mathcal{A}' \multimap \mathcal{B}'$*.*

Les quatre plongements rigides ci-dessus se ramènent tous à la fonction identité lorsque $\phi_{\mathcal{A}}$ et $\phi_{\mathcal{B}}$ sont les injections canoniques de $|\mathcal{A}|$ dans $|\mathcal{A}'|$ et de $|\mathcal{B}|$ dans $|\mathcal{B}'|$ dans le cas particulier où on a les inclusions rigides $\mathcal{A} \Subset \mathcal{A}'$ et $\mathcal{B} \Subset \mathcal{B}'$. Ceci nous permet d'énoncer le corollaire suivant :

Corollaire 4.2.5 (Composition des inclusions rigides) — *Si* $\mathcal{A} \Subset \mathcal{A}'$ *et* $\mathcal{B} \Subset \mathcal{B}'$*, alors :*

$$\mathcal{A} \,\&\, \mathcal{B} \;\Subset\; \mathcal{A}' \,\&\, \mathcal{B}'; \quad \mathcal{A} \oplus \mathcal{B} \;\Subset\; \mathcal{A}' \oplus \mathcal{B}'; \quad \mathcal{A} \to \mathcal{B} \;\Subset\; \mathcal{A}' \to \mathcal{B}' \quad \textit{et} \quad \mathcal{A} \multimap \mathcal{B} \;\Subset\; \mathcal{A}' \multimap \mathcal{B}'.$$

Dans ce qui suit, nous appellerons *catégorie des plongements rigides* la catégorie dont les objets sont les espaces cohérents et dont les morphismes sont les plongements rigides. La proposition et le corollaire précédents indiquent une des propriétés centrales de cette catégorie, qui est la suivante :

> *Dans la catégorie des plongements rigides, les bi-foncteurs produit-direct, somme directe, mais aussi les bi-foncteurs exponentielle stable et exponentielle linéaire sont covariants par rapport à leurs deux composantes.*

Ce résultat, qui est évident pour le produit direct et pour la somme directe, semble beaucoup moins intuitif pour l'espace des fonctions stables (et celui des fonctions linéaires), puisque le bi-foncteur

$$(\mathcal{A}, \mathcal{B}) \quad \mapsto \quad (\mathcal{A} \to \mathcal{B})$$

agit généralement de manière *covariante* vis-à-vis de sa seconde composante (la composante "codomaine") mais de manière *contravariante* vis-à-vis de sa première composante (la composante "domaine").

Ceci n'est en fait vrai que dans la mesure où l'on considère ce bi-foncteur *dans la catégorie des espaces cohérents*, c'est-à-dire dans la catégorie où les morphismes sont donnés par les fonctions stables, là où le bi-foncteur $(\mathcal{A}, \mathcal{B}) \mapsto (\mathcal{A} \to \mathcal{B})$ est une *exponentielle*.

Dans la catégorie des *plongements rigides*, ceci n'est plus vrai. En effet, l'espace des fonctions stables $\mathcal{A} \to \mathcal{B}$ n'est plus une exponentielle, car il est construit à partir d'une notion de morphisme différente de la notion de plongement rigide. Il n'y a donc aucune objection à ce que ce bi-foncteur se comporte à présent de manière covariante vis-à-vis de ses deux composantes, puisque la notion de morphisme vis-à-vis de laquelle est exprimée cette propriété de covariance (la notion de plongement rigide) diffère de la notion de morphisme présidant à la construction de ce bi-foncteur (la notion de fonction stable).

Proposition 4.2.6 (Isomorphismes) — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents, et $F : \mathcal{A} \to \mathcal{B}$ une fonction stable. Si F est bijective, alors*

1. *la fonction F et sa réciproque F^{-1} sont des fonctions linéaires ;*
2. *le couple (F, F^{-1}) est un couple plongement-rétraction de $\mathcal{A}$ vers $\mathcal{B}$ induit par un plongement rigide $\phi : \mathcal{A} \rightarrowtail \mathcal{B}$ établissant une correspondance bijective entre les atomes de $\mathcal{A}$ et les atomes de $\mathcal{B}$.*

En d'autres termes, cette proposition exprime que la notion d'isomorphisme au sens des fonctions stables coïncide avec la notion d'isomorphisme au sens des plongements rigides ou, si l'on préfère, qu'un isomorphisme au sens des fonctions stables (au niveau des cliques) induit immédiatement un isomorphisme au sens des trames sous-jacentes (au niveau des atomes).

Lorsque nous parlerons d'espaces cohérents isomorphes, nous n'aurons donc pas besoin de préciser s'il s'agit d'un isomorphime au sens des fonctions stables ou au sens des plongements rigides, puisque ces deux notions sont équivalentes.

4.2.2 Colimites

Considérons un ensemble ordonné $(I, \leq)$ dirigé (*i.e.* tel que pour tous $i, j \in I$ il existe $k \in I$ tel que $i \leq k$ et $j \leq k$) ainsi qu'une famille croissante d'espaces cohérents $(\mathcal{A}_i)_{i \in I}$ indicée par cet ensemble ordonné I, c'est-à-dire une famille d'espaces $(\mathcal{A}_i)_{i \in I}$ telle que

$$\forall\, i, j \in I \qquad i \leq j \quad \Rightarrow \quad \mathcal{A}_i \Subset \mathcal{A}_j.$$

Définition 4.2.7 (Colimite) — Si $(I, \leq)$ est un ensemble dirigé, et si $(\mathcal{A}_i)_{i \in I}$ est une famille croissante d'espaces cohérents indicée par l'ensemble I, on appelle *colimite* de cette famille l'espace cohérent noté $\operatorname*{colim}_{i \in I} \mathcal{A}_i$ dont la trame est donnée par

$$\left|\operatorname*{colim}_{i \in I} \mathcal{A}_i\right| \quad = \quad \bigcup_{i \in I} |\mathcal{A}_i| \quad \text{(union dirigée)}$$

et dont la relation de cohérence $\alpha_1 \frown \alpha_2 \pmod{\operatorname*{colim}_{i \in I} \mathcal{A}_i}$ est définie par

$$\alpha_1 \frown \alpha_2 \pmod{\operatorname*{colim}_{i \in I} \mathcal{A}_i} \qquad \equiv \qquad \exists\, i \in I \quad \alpha_1, \alpha_2 \in |\mathcal{A}_i| \quad \text{et} \quad \alpha_1 \frown \alpha_2 \pmod{\mathcal{A}_i}.$$

Notons que la trame de la colimite est égale à la réunion croissante des trames des espaces $\mathcal{A}_i$, mais également que la relation de cohérence sur la colimite est la réunion croissante des relations de cohérence sur les espaces $\mathcal{A}_i$.

Proposition 4.2.8 (Caractérisation de la colimite) — *Soit $(\mathcal{A}_i)_{i\in I}$ une famille croissante d'espaces cohérents indicée par un ensemble dirigé I. La colimite de cette famille est le plus petit espace cohérent dans lequel sont inclus rigidement tous les espaces $\mathcal{A}_i$, c'est-à-dire :*

1. *Pour tout $i \in I$, $\mathcal{A}_i \Subset \operatorname*{colim}_{i\in I} \mathcal{A}_i$.*
2. *Pout tout espace cohérent $\mathcal{A}$ tel que $\mathcal{A}_i \Subset \mathcal{A}$ quel que soit $i \in I$, on a $\operatorname*{colim}_{i\in I} \mathcal{A}_i \Subset \mathcal{A}$.*

Enfin, les bi-foncteurs produit direct, somme directe, exponentielle stable et exponentielle linéaire commutent avec les colimites :

Proposition 4.2.9 (Commutation avec les colimites) — *Soient $(\mathcal{A}_i)_{i\in I}$ et $(\mathcal{B}_i)_{i\in I}$ des familles croissantes d'espaces cohérents indicées par un même ensemble dirigé I. On a :*

$$\begin{aligned}
\operatorname*{colim}_{i\in I}(\mathcal{A}_i \,\&\, \mathcal{B}_i) &= \left(\operatorname*{colim}_{i\in I} \mathcal{A}_i\right) \,\&\, \left(\operatorname*{colim}_{i\in I} \mathcal{B}_i\right) \\
\operatorname*{colim}_{i\in I}(\mathcal{A}_i \oplus \mathcal{B}_i) &= \left(\operatorname*{colim}_{i\in I} \mathcal{A}_i\right) \oplus \left(\operatorname*{colim}_{i\in I} \mathcal{B}_i\right) \\
\operatorname*{colim}_{i\in I}(\mathcal{A}_i \to \mathcal{B}_i) &= \left(\operatorname*{colim}_{i\in I} \mathcal{A}_i\right) \to \left(\operatorname*{colim}_{i\in I} \mathcal{B}_i\right) \\
\operatorname*{colim}_{i\in I}(\mathcal{A}_i \multimap \mathcal{B}_i) &= \left(\operatorname*{colim}_{i\in I} \mathcal{A}_i\right) \multimap \left(\operatorname*{colim}_{i\in I} \mathcal{B}_i\right)
\end{aligned}$$

Remarquons que dans la proposition précédente, les égalités énoncées ne sont pas des égalités "à isomorphisme près", mais bien des égalités au sens de la théorie des ensembles, puisque nous travaillons ici avec des inclusions rigides, et non avec des plongements rigides.

4.2.3 Construction d'un modèle du λ-calcul

La catégorie cartésienne fermée des espaces cohérents est réflexive, en ce sens qu'elle admet un objet $\mathcal{A}$ non-trivial (*i.e.* différent de l'espace cohérent nul) tel que $\mathcal{A}$ est isomorphe à l'espace de ses fonctions $\mathcal{A} \to \mathcal{A}$. Un espace $\mathcal{A}$ isomorphe à $\mathcal{A} \to \mathcal{A}$ constitue naturellement un λ-modèle extensionnel :

Proposition 4.2.10 (λ-modèle extensionnel cohérent) — *Si $\mathcal{A}$ est un espace cohérent isomorphe à l'espace de ses fonctions stables $\mathcal{A} \to \mathcal{A}$, alors il existe un opérateur binaire $(\cdot) : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ et une fonction d'interprétation $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{A}} \mapsto \mathcal{A}$ tels que le triplet $(\mathcal{A}, \cdot, [\![]\!])$ constitue un λ-modèle extensionnel.*

(Cette proposition sera démontrée dans un cadre plus général au paragraphe 4.3.3.)

Sans entrer dans les détails, la construction d'un tel espace suit le schéma général suivant : On considère un espace $\mathcal{A}_0$ non trivial muni d'un plongement rigide $\phi_0 : \mathcal{A}_0 \rightarrowtail [\mathcal{A}_0 \to \mathcal{A}_0]$. Un tel espace est facile à construire, par exemple en posant $\mathcal{A}_0 = X_\perp$ (où X est un ensemble non vide arbitraire) et en munissant cet espace du plongement rigide $\phi_0 : \mathcal{A}_0 \rightarrowtail [\mathcal{A}_0 \to \mathcal{A}_0]$ défini pour tout $x \in X = |\mathcal{A}_0|$ par

$$\phi_0(x) \;=\; (\varnothing, x) \;\in\; |\mathcal{A}_0 \to \mathcal{A}_0|\,.$$

On définit ensuite par récurrence la suite $(\mathcal{A}_i)_{i\in\omega}$ en possant successivement $\mathcal{A}_{i+1} = (\mathcal{A}_i \to \mathcal{A}_i)$ pour chaque $i \in \omega$, et on munit ensuite cette suite d'espaces cohérents $(\mathcal{A}_i)_{i\in\omega}$ d'une suite de plongements rigides $\phi_i : \mathcal{A}_i \rightarrowtail \mathcal{A}_{i+1}$ construite par récurrence à partir de $\phi_0 : \mathcal{A}_0 \rightarrowtail \mathcal{A}_1$ à l'aide de la proposition 4.2.4. La suite d'espaces cohérents ainsi construite est croissante :

$$\mathcal{A}_0 \overset{\phi_0}{\rightarrowtail} \mathcal{A}_1 \overset{\phi_1}{\rightarrowtail} \mathcal{A}_2 \rightarrowtail \cdots \rightarrowtail \mathcal{A}_i \overset{\phi_i}{\rightarrowtail} \mathcal{A}_{i+1} \rightarrowtail \cdots$$

et, quitte à renommer les atomes des espaces $\mathcal{A}_i$, il est même possible de faire en sorte que les plongements rigides $\phi_i : \mathcal{A}_i \rightarrowtail \mathcal{A}_{i+1}$ deviennent de simples inclusions rigides $\mathcal{A}_i \Subset \mathcal{A}_{i+1}$. On considère alors la colimite

$$\mathcal{A} \quad = \quad \operatorname*{colim}_{i\in\omega} \mathcal{A}_i$$

et on vérifie simplement que

$$\mathcal{A} \to \mathcal{A} \quad = \quad \Big(\operatorname*{colim}_{i\in\omega} \mathcal{A}_i\Big) \to \Big(\operatorname*{colim}_{i\in\omega} \mathcal{A}_i\Big) \quad = \quad \operatorname*{colim}_{i\in\omega} (\mathcal{A}_i \to \mathcal{A}_i) \quad \approx \quad \operatorname*{colim}_{i\in\omega} \mathcal{A}_{i+1} \quad = \quad \mathcal{A}$$

d'après la proposition 4.2.9. Il est par ailleurs clair que l'espace cohérent $\mathcal{A}$ est non-trivial si et seulement si l'espace de départ $\mathcal{A}_0$ est lui-même non-trivial (ce qui est automatiquement le cas si l'on pose $\mathcal{A}_0 = X_\perp$ avec $X \neq \varnothing$).

λ-modèles cohérents Dans la construction ci-dessus, nous avons volontairement omis tous les détails liés à la description des isomorphismes et au problème du renommage des atomes des espaces $\mathcal{A}_i$. Notons que pour calculer la colimite telle que nous l'avons définie au paragraphe 4.2.2, il est nécessaire de renommer en bloc tous les atomes des espaces $\mathcal{A}_i$ de manière à transformer les plongements rigides $\phi_i : \mathcal{A}_i \rightarrowtail \mathcal{A}_{i+1}$ en des inclusions rigides $\mathcal{A}_i \Subset \mathcal{A}_{i+1}$. Cependant, un tel renommage fait disparaître l'égalité définitionnelle $\mathcal{A}_{i+1} = (\mathcal{A}_i \to \mathcal{A}_i)$ en la remplaçant par un isomorphisme $\mathcal{A}_{i+1} \approx (\mathcal{A}_i \to \mathcal{A}_i)$, ce qui explique qu'au final on ait seulement un isomorphisme $\mathcal{A} \approx (\mathcal{A} \to \mathcal{A})$.

Dans le cadre de la théorie des ensembles, il est généralement assez difficile de construire un espace $\mathcal{A}$ tel qu'on ait l'*égalité* $\mathcal{A} = (\mathcal{A} \to \mathcal{A})$ et non un simple isomorphisme.[6] Pour cette raison, il est souvent plus commode de relâcher la contrainte d'isomorphisme $\mathcal{A} \approx (\mathcal{A} \to \mathcal{A})$ en ne demandant que l'existence d'un plongement rigide $\phi : (\mathcal{A} \to \mathcal{A}) \rightarrowtail \mathcal{A}$. Dans ce cas, il est facile de construire un tel espace $\mathcal{A}$ pour lequel le plongement rigide $\phi : (\mathcal{A} \to \mathcal{A}) \rightarrowtail \mathcal{A}$ est en réalité une simple inclusion rigide $(\mathcal{A} \to \mathcal{A}) \Subset \mathcal{A}$.

De plus, les espaces cohérents $\mathcal{A}$ tels qu'il existe un plongement rigide $\phi : (\mathcal{A} \to \mathcal{A}) \rightarrowtail \mathcal{A}$ capturent bien la notion de λ-modèle sous-extensionnel introduite au paragraphe 3.3.1 :

[6] Il est facile de voir qu'une égalité de la forme $\mathcal{A} = (\mathcal{A} \to \mathcal{A})$ contredit l'axiome de la fondation lorsque $\mathcal{A}$ n'est pas l'espace nul. En effet, si l'on considère un atome $\alpha \in |\mathcal{A}|$, il est possible de construire par récurrence une famille d'atomes $(\alpha_i)_{i\in\omega}$ et une famille de cliques $(a_i)_{i\in\omega}$ en posant

$$\alpha_0 \;=\; \alpha \qquad \text{et} \qquad (a_i, \alpha_{i+1}) = \alpha_i \qquad \text{pour tout } i \in \omega$$

la seconde égalité ne faisant du sens que dans la mesure où les trames $|\mathcal{A}|$ et $|\mathcal{A} \to \mathcal{A}|$ sont égales. Sachant que dans ZF, les couples $\alpha_i = (a_i, \alpha_{i+1})$ sont codés par $\alpha_i = \{\{a_i\}; \{a_i; \alpha_{i+1}\}\}$, on a immédiatement la chaîne infinie d'appartenances descendantes

$$\alpha_0 \quad \ni \quad \{a_0; \alpha_1\} \;\ni\; \alpha_1 \;\ni\; \{a_1; \alpha_2\} \;\ni\; \alpha_2 \;\ni\; \cdots \;\ni\; \alpha_i \;\ni\; \{a_i; \alpha_{i+1}\} \;\ni\; \alpha_{i+1} \;\ni\; \cdots$$

ce qui contredit l'axiome de la fondation.

Proposition 4.2.11 (λ-modèle cohérent) — *Si $\mathcal{A}$ est un espace cohérent muni d'un plongement rigide $\phi : (\mathcal{A} \to \mathcal{A}) \rightarrowtail \mathcal{A}$, alors il existe un opérateur binaire $(\cdot) : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ et une fonction d'interprétation $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{A}} \mapsto \mathcal{A}$ tels que le quadruplet $(\mathcal{A}, \cdot, [\![]\!], \subset)$ constitue un λ-modèle sous-extensionnel.*

(Cette proposition sera montrée dans un cadre plus général au paragraphe 4.3.3.)

4.3 La stabilité revisitée

Nous avons vu au paragraphe 4.1.3 que le critère de stabilité binaire des fonctions stables s'étendait par une récurrence immédiate en un critère de *stabilité n-aire* :

$$a_1 \cup \cdots \cup a_n \in \mathcal{A} \qquad \Rightarrow \qquad F(a_1 \cap \cdots \cap a_n) \;=\; F(a_1) \cap \cdots \cap F(a_n) \qquad (n > 0).$$

En fait, le critère de continuité imposé aux fonctions stables permet par passage à la limite d'étendre cette propriété à toute intersection non-vide d'une famille de cliques compatibles, même dans le cas où cette famille est infinie :

Proposition 4.3.1 (Quasi-stabilité) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents. Si $F : \mathcal{A} \to \mathcal{B}$ est une fonction stable, alors pour toute famille non vide $(a_i)_{i\in I}$ d'éléments de $\mathcal{A}$ on a :*

$$\bigcup_{i\in I} a_i \in \mathcal{A} \qquad \Rightarrow \qquad F\Big(\bigcap_{i\in I} a_i\Big) \;=\; \bigcap_{i\in I} F(a_i).$$

Preuve. Soient $F : \mathcal{A} \to \mathcal{B}$ une fonction stable, et $(a_i)_{i\in I}$ une famille non-vide de cliques compatibles de $\mathcal{A}$. Notons $\bar{a} \in \mathcal{A}$ la réunion de la famille $(a_i)_{i\in I}$, et considérons un approximant fini $a' \in \Im(\bar{a})$ quelconque. La famille $(a' \cap a_i)_{i\in I}$ ne comporte qu'un nombre fini d'éléments, car elle n'est constituée que d'approximants finis de a', lesquels sont en nombre fini. On peut alors appliquer le critère de stabilité n-aire qui nous donne ainsi

$$\begin{aligned}
F\Big(a' \;\cap\; \bigcap_{i\in I} a_i\Big) \;&=\; F\Big(\bigcap_{i\in I} (a' \cap a_i)\Big) \\
&=\; \bigcap_{i\in I} F(a' \cap a_i) && \text{(stabilité } n\text{-aire)} \\
&=\; \bigcap_{i\in I} \big(F(a') \cap F(a_i)\big) && \text{(stabilité binaire)} \\
&=\; F(a') \;\cap\; \bigcap_{i\in I} F(a_i)
\end{aligned}$$

(Toutes les cliques impliquées dans les égalités ci-dessus sont compatibles car incluses dans $\bar{a}$.) Par passage à la limite, il vient :

$$\begin{aligned}
F\Big(\bigcap_{i\in I} a_i\Big) \;&=\; F\Big(\bigcup_{a'\in\Im(\bar{a})} \Big(a' \;\cap\; \bigcap_{i\in I} a_i\Big)\Big) && \text{(car } a_i \subset \bar{a} \text{ pour tout } i \in I) \\
&=\; \bigcup_{a'\in\Im(\bar{a})} F\Big(a' \;\cap\; \bigcap_{i\in I} a_i\Big) && \text{(passage à la limite dirigée)} \\
&=\; \bigcup_{a'\in\Im(\bar{a})} \Big(F(a') \;\cap\; \bigcap_{i\in I} F(a_i)\Big) && \text{(d'après l'égalité établie plus haut)} \\
&=\; F(\bar{a}) \;\cap\; \bigcap_{i\in I} F(a_i) \;=\; \bigcap_{i\in I} F(a_i),
\end{aligned}$$

la dernière égalité résultant de la monotonie et de la continuité de F. □

Cette dernière proposition est importante, car elle suggère une définition plus générale de la notion de stabilité, qui est complètement indépendante du critère de continuité : la *quasi-stabilité*. Ainsi que nous allons le constater dans le paragraphe qui suit, cette notion de quasi-stabilité possède de bonnes propriétés algébriques, dont la plus importante est sans doute la possibilité de représenter les fonctions quasi-stables à l'aide d'une trace (dont la définition diffère légèrement de la définition des traces dans le cas stable). Nous verrons en particulier que cette notion induit l'existence d'une catégorie cartésienne fermée — la catégorie des *quasi-espaces cohérents* — qui, à la différence de la catégorie des espaces cohérents, n'est pas une catégorie réflexive, et possède de ce fait une structure qui la rapproche davantage de la catégorie des ensembles.[7]

4.3.1 La catégorie des quasi-espaces cohérents

Définition 4.3.2 (Quasi-stabilité) — Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents. Une fonction $F : \mathcal{A} \to \mathcal{B}$ est dite *quasi-stable* si pour toute famille non vide de cliques $(a_i)_{i\in I} \in \mathcal{A}^I$ on a

$$\bigcup_{i\in I} a_i \ \in \ \mathcal{A} \quad \Rightarrow \quad F\Big(\bigcap_{i\in I} a_i\Big) \quad = \quad \bigcap_{i\in I} F(a_i).$$

D'après la proposition 4.3.1, toute fonction stable est quasi-stable, mais la réciproque est fausse en général : nous verrons plus loin des exemples de fonctions quasi-stables non continues. Intuitivement, le critère de quasi-stabilité est ce qu'il reste de la stabilité lorsqu'on a tout oublié de la continuité. En particulier, la notion de quasi-stabilité a de bonnes propriétés algébriques : nous verrons plus bas qu'elle permet de conserver la représentation par les traces, sous réserve d'autoriser dans sa définition des couples (a, β) où a n'est pas nécessairement une clique finie.[8]

Remarquons également que toute fonction quasi-stable est monotone : il suffit pour cela de considérer le cas particulier d'une famille à deux éléments a_1 et a_2 où $a_1 \subset a_2$. Par ailleurs, il est clair que la fonction identité est clairement une fonction quasi-stable, et que la quasi-stabilité est préservée par la composition de fonctions (la vérification est élémentaire). Autrement dit :

Définition 4.3.3 (Catégorie des quasi-espaces cohérents) — La classe des espaces cohérents munis des fonctions quasi-stables forme une catégorie, appelée *catégorie des quasi-espaces cohérents*.

Malgré l'absence de critère de continuité, il est possible de conserver la représentation par les traces, à cette différence près qu'on doit maintenant autoriser dans la trace des fonctions quasi-stables la présence de couples (a, β) où a est une clique (éventuellement) infinie.

[7] Rappelons que la catégorie des ensembles est une catégorie cartésienne fermée *non-réflexive*. La non-réflexivité résulte de l'argument classique de diagonalisation de Cantor selon lequel un ensemble X n'est jamais équipotent à l'ensemble de ses parties $\mathfrak{P}(X)$. Nous verrons que la catégorie des quasi-espaces cohérents n'est pas réflexive pour des raisons très similaires.

[8] À ce titre, le critère de stabilité binaire (n-aire) n'est pas un bon critère pour définir une notion de stabilité indépendante du cadre de la continuité. Il est facile de voir que ce critère n'est pas suffisant pour faire en sorte qu'une fonction soit déterminée par sa trace. En l'absence de tout critère de continuité, le critère de stabilité ne fait de sens que s'il est énoncé pour des intersections compatibles (non-vides) indicées par des ensembles de cardinal arbitraire.

Définition 4.3.4 (Trace d'une fonction quasi-stable) — Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents, et $F : \mathcal{A} \to \mathcal{B}$ une fonction quasi-stable. La *trace* de F, notée $\mathsf{Tr}(F)$, est l'ensemble des couples $(a_0, \beta) \in \mathcal{A} \times |\mathcal{B}|$ tels que

1. $\beta \in F(a_0)$;
2. pour tout $a_1 \subsetneq a_0$ on a $\beta \notin F(a_1)$.

Insistons sur le fait que cette définition n'effectue aucune hypothèse sur le cardinal des cliques a_0 intervenant dans les couples $(a_0, \beta) \in \mathsf{Tr}(F)$, alors que dans le cas stable on impose en outre que la clique a_0 soit finie.

Proposition 4.3.5 (Application) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents, et $F : \mathcal{A} \to \mathcal{B}$ une fonction quasi-stable. Pour toute clique $a \in \mathcal{A}$ on a :*

$$F(a) \quad = \quad \{\beta \in |\mathcal{B}| \,; \quad \exists a_0 \subset a \quad (a_0, \beta) \in \mathsf{Tr}(F)\}.$$

Preuve. Soit F une fonction quasi-stable et $a \in \mathcal{A}$ une clique arbitraire. L'inclusion

$$\{\beta \in |\mathcal{B}| \,; \quad \exists a_0 \subset a \quad (a_0, \beta) \in \mathsf{Tr}(F)\} \quad \subset \quad F(a)$$

est immédiate d'après la définition de la trace, et du fait que F est une fonction monotone. Pour montrer l'inclusion réciproque, considérons un atome $\beta \in F(a)$ quelconque. Soit D l'ensemble des cliques $a' \subset a$ telles que $\beta \in F(a')$. Cet ensemble est un ensemble non-vide (car $a \in D$), dont les éléments sont des cliques compatibles. D'après la définition de la quasi-stabilité on a donc

$$\beta \quad \in \quad \bigcap_{a' \in D} F(a') \quad = \quad F\Big(\bigcap_{a' \in D} a'\Big),$$

ce qui prouve que D admet un plus petit élément (i.e. l'intersection des éléments de D), que nous noterons a_0. D'après ce qui précède, il est clair que $\beta \in F(a_0)$, et que pour tout $a_1 \subsetneq a_0$, on a $\beta \notin F(a_1)$ puisque a_0 est le plus petit élément de D. Par conséquent, on a $(a_0, \beta) \in \mathsf{Tr}(F)$, avec $a_0 \subset a$. □

Corollaire 4.3.6 (Caractérisation par la trace) — *Les fonctions quasi-stables sont caractérisées par leurs traces : si $F, G : \mathcal{A} \to \mathcal{B}$ sont des fonctions quasi-stables, alors*

$$\mathsf{Tr}(F) = \mathsf{Tr}(G) \quad \Leftrightarrow \quad F = G$$

Preuve. L'implication directe résulte de la proposition précédente et de l'extensionnalité des fonctions en théorie des ensembles (la réciproque étant triviale). □

Proposition 4.3.7 (Trace stable) — *Pour toute fonction stable $F : \mathcal{A} \to \mathcal{B}$, la trace de F au sens des fonctions quasi-stables coïncide avec la trace de F au sens des fonctions stables.*

Preuve. Il s'agit essentiellement de vérifier que la trace de F au sens des fonctions quasi-stables ne contient que des couples $(a, \beta) \in \mathcal{A} \times |\mathcal{B}|$ où a est fini, ce qui est immédiat en vertu du critère de continuité. □

Proposition 4.3.8 (Espace cohérent des fonctions quasi-stables) — *Soient $\mathcal{A}$, $\mathcal{B}$ des espaces cohérents. L'ensemble des traces de toutes les fonctions quasi-stables $F : \mathcal{A} \to \mathcal{B}$ forme un espace cohérent noté $\mathcal{A} \xrightarrow{q} \mathcal{B}$, dont la trame est*

$$\left|\mathcal{A} \xrightarrow{q} \mathcal{B}\right| \quad = \quad \mathcal{A} \times |\mathcal{B}|$$

et dont la relation de cohérence est caractérisée par :

$$(a_1,\beta_1) \frown (a_2,\beta_2) \pmod{\mathcal{A} \xrightarrow{q} \mathcal{B}} \quad \Leftrightarrow \quad \begin{array}{l} \big(a_1 \cup a_2 \in \mathcal{A} \;\Rightarrow\; \beta_1 \frown \beta_2 \pmod{\mathcal{B}}\big) \;\; \textit{et} \\ \big(a_1 \cup a_2 \in \mathcal{A} \;\; \textit{and} \;\; a_1 \neq a_2 \;\Rightarrow\; \beta_1 \neq \beta_2\big) \end{array}$$

Preuve. Il s'agit essentiellement de vérifier que pour tout $a_0 \in \mathcal{A}$ et pour tout $\beta \in |\mathcal{B}|$, la fonction $\mathsf{step}_{a_0,\beta}$ définie par

$$\mathsf{step}_{a_0,\beta}(a) \quad = \quad \begin{cases} \{\beta\} & \text{si } a_0 \subset a \\ \varnothing & \text{sinon} \end{cases}$$

constitue effectivement une fonction quasi-stable dont la trace est $\{(a_0,\beta)\}$. Le reste de la preuve se conduit de la même manière que dans le cas de l'espace des fonctions stables. □

Proposition 4.3.9 (Covariance) — *La correspondance $(\mathcal{A},\mathcal{B}) \mapsto (\mathcal{A} \xrightarrow{q} \mathcal{B})$ constitue un foncteur covariant (non-continu) dans la catégorie des plongements rigides, c'est-à-dire :*

$$\mathcal{A} \Subset \mathcal{A}' \;\; \textit{and} \;\; \mathcal{B} \Subset \mathcal{B}' \quad \Rightarrow \quad \mathcal{A} \xrightarrow{q} \mathcal{B} \;\Subset\; \mathcal{A}' \xrightarrow{q} \mathcal{B}'.$$

Preuve. Résulte immédiatement de la structure de l'espace cohérent des (traces des) fonctions quasi-stables. □

Notons que le bi-foncteur covariant $(\mathcal{A},\mathcal{B}) \mapsto [\mathcal{A} \xrightarrow{q} \mathcal{B}]$ n'est pas continu, car il ne commute pas en général avec les colimites de suites croissantes d'espaces cohérents indicés par un ensemble dirigé. (Nous verrons que dans le cadre des fonctions μ-stables, l'exponentielle μ-stable ne commute pas avec les colimites dirigées, mais avec les colimites μ-dirigées.)

Proposition 4.3.10 (Fermeture cartésienne) — *La catégorie des quasi-espaces cohérents est une catégorie cartésienne fermée non-réflexive.*

Preuve. Il est facile de vérifier que le produit direct $\mathcal{A} \,\&\, \mathcal{B}$ constitue également un produit cartésien dans la catégorie des quasi-espaces cohérents. Pour montrer que cette catégorie est cartésienne fermée, il suffit de vérifier que la fonction d'application

$$\mathbf{App} \quad : \quad [\mathcal{A} \,\&\, (\mathcal{A} \xrightarrow{q} \mathcal{B})] \;\to\; \mathcal{B}$$

et l'isomorphisme de Curry

$$\mathbf{Curry} \quad : \quad [\mathcal{A} \,\&\, \mathcal{B} \xrightarrow{q} \mathcal{C}] \;\to\; [\mathcal{A} \xrightarrow{q} (\mathcal{B} \xrightarrow{q} \mathcal{C})]$$

sont des fonctions quasi-stables, dont les traces sont immédiatement données par

$$\mathsf{Tr}(\mathbf{App}) \quad = \quad \Big\{\big(\{1\} \times a \;\cup\; \{(2,(a,\beta))\},\; \beta\big); \quad a \in \mathcal{A},\; \beta \in |\mathcal{B}|\Big\}$$

$$\mathsf{Tr}(\mathbf{Curry}) \quad = \quad \Big\{\big(\{(\{1\} \times a \;\cup\; \{2\} \times b,\gamma)\},\; (a,(b,\gamma))\big); \quad a \in \mathcal{A},\; b \in \mathcal{B},\; \gamma \in |\mathcal{C}|\Big\}.$$

La catégorie des quasi-espaces cohérents n'est pas réflexive parce qu'il n'existe pas d'espace cohérent non-trivial $\mathcal{A}$ tel qu'on ait un plongement rigide de $\mathcal{A} \xrightarrow{q} \mathcal{A}$ dans $\mathcal{A}$, ce qui résulte d'un argument de cardinalité immédiat. □

4.3.2 La catégorie des μ-espaces cohérents

Dans ce paragraphe, μ désigne un cardinal régulier infini.

Définition 4.3.11 (Sous-ensemble μ-dirigé) — Soit $(D, \leq)$ un ensemble ordonné quelconque. Un sous-ensemble $X \subset D$ est dit *μ-dirigé* si pour tout $Y \subset X$ tel que $\overline{\overline{Y}} < \mu$, il existe $x \in X$ tel que $y \leq x$ pour tout $y \in Y$.

La notion de sous-ensemble μ-dirigé est la généralisation naturelle de la notion d'ensemble dirigé, dans laquelle on ne considère plus seulement les sous-ensembles finis, mais les sous-ensembles de cardinal strictement inférieur à μ. En particulier, un sous-ensemble $X \subset D$ est $\aleph_0$-dirigé si et seulement si il est dirigé au sens habituel du terme. Par ailleurs, tout ensemble μ-dirigé est ($\aleph_0$-)dirigé, mais la réciproque est bien évidemment fausse.

De manière analogue, on dira qu'une famille $(x_i)_{i\in I}$ d'éléments de D est μ-dirigée si son image est un sous-ensemble μ-dirigé de D.

Définition 4.3.12 (μ-approximants) — Soit $\mathcal{A}$ un espace cohérent et $a \in \mathcal{A}$ une clique. On appelle *μ-approximant* de a toute clique $a_0 \subset a$ telle que $\overline{\overline{a_0}} < \mu$. L'ensemble des μ-approximants de $\mathcal{A}$ est noté $\Im_\mu(a)$.

L'hypothèse selon laquelle μ est un cardinal régulier est ici très importante, car elle permet de montrer que $\Im_\mu(a)$ est un sous-ensemble μ-dirigé de $\mathcal{A}$. (Il est facile de vérifier que cette propriété est invalidée si l'on suppose que μ est singulier.)

Définition 4.3.13 (μ-continuité et μ-stabilité) — Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents. Une fonction $F : \mathcal{A} \to \mathcal{B}$ est dite

- *μ-continue* si F est monotone et si pour tout $a \in \mathcal{A}$ on a
$$F(a) \;=\; \bigcup_{a_0 \in \Im_\mu(a)} F(a_0).$$
- *μ-stable* si F est μ-continue et quasi-stable.

Là encore, la notion de μ-continuité (resp. μ-stabilité) est une généralisation naturelle de la notion de continuité (resp. stabilité). En particulier, une fonction est $\aleph_0$-continue si et seulement si elle est continue au sens de Scott. En vertu de la proposition 4.3.1, il est également clair qu'une fonction est $\aleph_0$-stable si et seulement si elle est stable au sens usuel. Notons également que toute fonction μ-continue $F : \mathcal{A} \to \mathcal{B}$ commute avec les limites μ-dirigées :
$$F\Big(\bigcup_{i\in I} a_i\Big) \quad = \quad \bigcup_{i\in I} F(a_i) \qquad \big((a_i)_{i\in I} \text{ } \mu\text{-dirigée}\big)$$

En outre, l'hypothèse de régularité effectuée sur le cardinal μ permet de montrer le lemme suivant :

Lemme 4.3.14 (Composition) — *La composée de deux fonctions μ-continues (resp. μ-stables) est une fonction μ-continue (resp. μ-stable).*

Insistons sur le fait que l'hypothèse de régularité est ici centrale : si μ est singulier, la notion de μ-continuité (resp. μ-stabilité) n'a pas de sens, et il est facile de construire des contre-exemples montrant que dans ce cas, la propriété de μ-continuité (resp. μ-stabilité) n'est pas préservée par la composition de fonctions.

Par la suite, on ne s'intéressera plus qu'aux fonctions μ-stables, qui ont l'avantage de pouvoir être représentées par leurs traces.

Définition 4.3.15 (Catégorie des μ-espaces cohérents) — La classe des espaces cohérents munis des fonctions μ-stables forme une catégorie, appelée *catégorie des μ-espaces cohérents.*

Proposition 4.3.16 (Caractérisation des fonctions μ-stables) — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents. Une fonction quasi-stable $F : \mathcal{A} \xrightarrow{q} \mathcal{B}$ est μ-stable si et seulement si sa trace n'est constituée que d'atomes $(a, \beta) \in \mathcal{A} \times |\mathcal{B}|$ tels que $\overline{\overline{a}} < \mu$.*

Preuve. Immédiat d'après la définition de la trace. □

Corollaire 4.3.17 (Espace cohérent des fonctions μ-stables) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents. L'ensemble des traces de toutes les fonctions μ-stables forme un espace cohérent noté $\mathcal{A} \xrightarrow{\mu} \mathcal{B}$, dont la trame est donée par*

$$\left|\mathcal{A} \xrightarrow{\mu} \mathcal{B}\right| \quad = \quad \mathcal{A}_{(\mu)} \times |\mathcal{B}|$$

(où $\mathcal{A}_{(\mu)}$ désigne l'ensemble des cliques de $\mathcal{A}$ de cardinal $< \mu$) et dont la relation de cohérence est caractérisée par la même formule que pour les fonctions quasi-stables.

Preuve. Conséquence immédiate de la proposition précédente. □

Remarquons que l'espace des fonctions μ-stables $\mathcal{A} \xrightarrow{\mu} \mathcal{B}$ est rigidement inclus dans l'espace des fonctions quasi-stables $\mathcal{A} \xrightarrow{q} \mathcal{B}$. Plus généralement, si (μ_x) désigne la suite transfinie des cardinaux réguliers infinis, on a les inclusions rigides

$$\mathcal{A} \to \mathcal{B} \quad = \quad \mathcal{A} \xrightarrow{\aleph_0} \mathcal{B} \quad \Subset \quad \mathcal{A} \xrightarrow{\mu_1} \mathcal{B} \quad \Subset \quad \mathcal{A} \xrightarrow{\mu_2} \mathcal{B} \quad \Subset \quad \ldots \quad \Subset \quad \mathcal{A} \xrightarrow{q} \mathcal{B}.$$

Bien entendu, cette suite transfinie est stationaire à partir d'un certain rang. Lorsque μ est strictement plus grand que le cardinal de la trame de $\mathcal{A}$, on a $\mathcal{A} \xrightarrow{\mu} \mathcal{B} = \mathcal{A} \xrightarrow{q} \mathcal{B}$.

Proposition 4.3.18 (Colimites μ-dirigées) — *La correspondance $(\mathcal{A}, \mathcal{B}) \mapsto (\mathcal{A} \xrightarrow{\mu} \mathcal{B})$ définit un bi-foncteur covariant μ-continu dans la catégorie des plongements rigides, c'est-à-dire :*

1. *Si $\mathcal{A} \Subset \mathcal{A}'$ et $\mathcal{B} \Subset \mathcal{B}'$, alors $(\mathcal{A} \xrightarrow{\mu} \mathcal{B}) \Subset (\mathcal{A}' \xrightarrow{\mu} \mathcal{B}')$.*
2. *Si $(\mathcal{A}_i)_{i\in I}$ et $(\mathcal{B}_i)_{i\in I}$ sont deux familles croissantes d'espaces cohérents indicées par un même ensemble μ-dirigé I, on a :*

$$\operatorname*{colim}_{i\in I}\left(\mathcal{A}_i \xrightarrow{\mu} \mathcal{B}_i\right) \quad = \quad \left(\operatorname*{colim}_{i\in I} \mathcal{A}_i\right) \xrightarrow{\mu} \left(\operatorname*{colim}_{i\in I} \mathcal{B}_i\right).$$

Preuve. Le premier point est immédiat d'après la définition de l'espace $\mathcal{A} \xrightarrow{\mu} \mathcal{B}$. Considérons deux familles croissantes d'espaces cohérents $(\mathcal{A}_i)_{i\in I}$ et $(\mathcal{B}_i)_{i\in I}$ indicées par un même ensemble ordonné $(I, \leq)$ μ-dirigé. Notons

$$\mathcal{A} \;=\; \operatorname*{colim}_{i\in I} \mathcal{A}_i \qquad \text{et} \qquad \mathcal{B} \;=\; \operatorname*{colim}_{i\in I} \mathcal{B}_i.$$

Pour tout $i \in I$, on a $\mathcal{A}_i \Subset \mathcal{A}$ et $\mathcal{B}_i \Subset \mathcal{B}$, d'où $(\mathcal{A}_i \xrightarrow{\mu} \mathcal{B}_i) \Subset (\mathcal{A} \xrightarrow{\mu} \mathcal{B})$. Par passage à la colimite, il vient naturellement

$$\operatorname*{colim}_{i\in I}(\mathcal{A}_i \xrightarrow{\mu} \mathcal{B}_i) \quad \Subset \quad (\mathcal{A} \xrightarrow{\mu} \mathcal{B}).$$

Pour montrer que ces deux espaces sont égaux, il suffit donc de montrer que tout atome de la trame du second est également un atome de la trame du premier. Pour cela, considérons un atome $(a, \beta) \in \left|\mathcal{A} \xrightarrow{\mu} \mathcal{B}\right|$. Comme $|\mathcal{A}| = \bigcup_{i\in I} |\mathcal{A}_i|$ et $|\mathcal{B}| = \bigcup_{i\in I} |\mathcal{B}_i|$, il existe des indices $i_\alpha \in I$ ($\alpha \in a$) et un indice $i_\beta \in I$ tels que $\alpha \in |\mathcal{A}_{i_\alpha}|$ pour tout $\alpha \in a$ et $\beta \in \left|\mathcal{B}_{i_\beta}\right|$. Comme $\overline{\overline{a}} < \mu$, l'ensemble d'indices $\{i_\alpha;\ \alpha \in a\} \cup \{i_\beta\}$ est de cardinal $< \mu$, d'où il existe $i_0 \in I$ tel que $i_\alpha \leq i_0$ pour tout $\alpha \in a$ et $i_\beta \leq i_0$ (puisque I est μ-dirigé). On a alors $\alpha \in |\mathcal{A}_{i_\alpha}| \subset |\mathcal{A}_{i_0}|$ pour tout $\alpha \in a$ et $\beta \in \left|\mathcal{B}_{i_\beta}\right| \subset |\mathcal{B}_{i_0}|$, d'où $(a, \beta) \in \left|\mathcal{A}_{i_0} \xrightarrow{\mu} \mathcal{B}_{i_0}\right|$, ce qui prouve que (a, β) est un atome de la colimite $\operatorname*{colim}_{i\in I}(\mathcal{A}_i \xrightarrow{\mu} \mathcal{B}_i)$. □

Proposition 4.3.19 (Fermeture cartésienne) — *La catégorie des μ-espaces cohérents est une catégorie cartésienne fermée réflexive.*

Preuve. La preuve de fermeture cartésienne s'effectue comme dans le cas de la stabilité usuelle. Pour montrer que cette catégorie est réflexive, on construit un espace cohérent non-trivial $\mathcal{A}$ tel que $\mathcal{A} \approx (\mathcal{A} \xrightarrow{\mu} \mathcal{A})$ suivant la même méthode qu'au paragraphe 4.2.3, à cette différence près que $\mathcal{A}$ n'est pas la colimite dénombrable d'une suite d'espace cohérents, mais une colimite transfinie d'une suite transfinie d'espaces cohérents indicée par l'ensemble des ordinaux $x < \mu$. On part d'un espace $\mathcal{A}_0$ muni d'un plongement rigide $\phi_0 : \mathcal{A}_0 \rightarrowtail (\mathcal{A}_0 \xrightarrow{\mu} \mathcal{A}_0)$ (construit par exemple avec la méthode décrite au paragraphe 4.2.3). On définit ensuite par récurrence transfinie une suite d'espaces cohérents $(\mathcal{A}_x)_{x<\mu}$ en posant

$$\begin{aligned}
\mathcal{A}_0 &= \mathcal{A}_0 \\
\mathcal{A}_{x+1} &= \mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x \\
\mathcal{A}_x &= \operatorname*{colim}_{y<x} \mathcal{A}_y \qquad \text{(si } x \text{ ordinal limite)}
\end{aligned}$$

Pour que cette définition ait un sens (en particulier dans le cas limite), il faut également montrer à chaque étape (correspondant à un ordinal $x < \mu$) qu'il existe une famille de plongements rigides $\phi_{y,x} : \mathcal{A}_y \rightarrowtail \mathcal{A}_x$ (pour $y < x$) telle qu'on ait $\phi_{z,x} = \phi_{y,x} \circ \phi_{z,y}$ pour tous $z < y < x$ (ce qui permet essentiellement de justifier l'existence de la colimite dans le cas où x est un ordinal limite). Cette suite transfinie étant construite, on renomme en bloc les atomes des espaces $\mathcal{A}_x$ de manière à transformer les plongements rigides $\phi_{y,x} : \mathcal{A}_y \rightarrowtail \mathcal{A}_x$ ($y < x < \mu$) en inclusions rigides $\mathcal{A}_y \Subset \mathcal{A}_x$, ce qui est possible grâce à la relation $\phi_{z,x} = \phi_{y,x} \circ \phi_{z,y}$ valable pour tous $z < y < x < \mu$. On considère alors la colimite

$$\mathcal{A} \quad = \quad \operatorname*{colim}_{x<\mu} \mathcal{A}_x,$$

calculée sur l'ensemble d'indices $\mu = \{x;\ On(x) \text{ et } x < \mu\}$ qui est un ensemble μ-dirigé puisque μ est un cardinal régulier. D'après la proposition 4.3.18 on a alors

$$\mathcal{A} \xrightarrow{\mu} \mathcal{A} \quad = \quad \Big(\operatorname*{colim}_{x<\mu} \mathcal{A}_x\Big) \xrightarrow{\mu} \Big(\operatorname*{colim}_{x<\mu} \mathcal{A}_x\Big) \quad = \quad \operatorname*{colim}_{x<\mu}(\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x) \quad \approx \quad \operatorname*{colim}_{x<\mu} \mathcal{A}_{x+1} \quad = \quad \mathcal{A}. \ \square$$

Ainsi que nous l'avons déjà évoqué dans le cadre de la ($\aleph_0$-)stabilité (au paragraphe 4.2.3), il assez difficile de construire un μ-espace cohérent $\mathcal{A}$ satisfaisant l'égalité $\mathcal{A} = (\mathcal{A} \xrightarrow{\mu} \mathcal{A})$ au sens de la théorie des ensembles. Par la suite, nous ne chercherons qu'à contruire des espaces $\mathcal{A}$ satisfaisant l'inclusion rigide $(\mathcal{A} \xrightarrow{\mu} \mathcal{A}) \Subset \mathcal{A}$, lesquels correspondent aux λ-modèles sous-extensionnels introduits au paragraphe 3.3.2.

4.3.3 λ-modèles μ-cohérents

Pour terminer cette introduction à la théorie des espaces cohérents, nous nous proposons de montrer que tout espace cohérent $\mathcal{A}$ muni d'un plongement rigide $\phi : (\mathcal{A} \xrightarrow{\mu} \mathcal{A}) \rightarrowtail \mathcal{A}$ constitue un λ-modèle sous-extensionnel. Comme dans le paragraphe précédent, nous supposerons ici que μ désigne un cardinal régulier infini arbitraire. La suite de ce paragraphe est consacrée à la preuve de la proposition suivante :

Proposition 4.3.20 (λ-modèle cohérent) — *Soit $\mathcal{A}$ un espace cohérent. La donnée d'un plongement rigide $\phi : (\mathcal{A} \xrightarrow{\mu} \mathcal{A}) \rightarrowtail \mathcal{A}$ confère à $\mathcal{A}$ une structure de λ-modèle sous-extensionnel $(\mathcal{A}, \cdot, [\![]\!], \subset)$, ordonné par l'inclusion de cliques. Par ailleurs, le λ-modèle sous-jacent $(\mathcal{A}, \cdot, [\![]\!])$ est extensionnel si et seulement si le plongement rigide ϕ est un isomorphisme.*

Dans tout ce qui suit, on suppose que $\mathcal{A}$ est un espace cohérent muni d'un plongement rigide $\phi : (\mathcal{A} \xrightarrow{\mu} \mathcal{A}) \rightarrowtail \mathcal{A}$. Rappelons que le plongement rigide ϕ induit un plongement $\phi^+ : (\mathcal{A} \xrightarrow{\mu} \mathcal{A}) \multimap \mathcal{A}$ et une rétraction $\phi^- : \mathcal{A} \multimap (\mathcal{A} \xrightarrow{\mu} \mathcal{A})$ reliés par

$$\phi^- \circ \phi^+ \ = \ \mathsf{id}_{\mathcal{A} \xrightarrow{\mu} \mathcal{A}} \qquad \text{et} \qquad \phi^+ \circ \phi^- \ \leq_{\mathbf{st}} \ \mathsf{id}_{\mathcal{A}}.$$

(L'égalité de gauche implique que ϕ^+ est injective tandis que ϕ^- est surjective).

Intuitivement, ϕ^+ transforme toute (trace de) fonction μ-stable $F : \mathcal{A} \xrightarrow{\mu} \mathcal{A}$ en une clique de $\mathcal{A}$, et la fonction ϕ^- extrait de toute clique de $\mathcal{A}$ sa "partie fonctionnelle", qui est une trace de fonction μ-stable. Pour tous $f, a \in \mathcal{A}$ on note

$$f \cdot a \quad = \quad \phi^-(f)(b) \quad = \quad \big\{\beta \in |\mathcal{A}|\,; \quad \exists a_0 \subset a \quad (a_0, \beta) \in \phi^-(f)\big\}.$$

L'opérateur $(\cdot) : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ ainsi défini est clairement une fonction μ-stable, donc monotone.

Avant de définir la fonction d'interprétation $[\![]\!] : \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$, il est d'abord nécessaire de munir l'ensemble des valuations $\mathbf{Val}_{\mathcal{A}}$ d'une structure d'espace cohérent, laquelle résulte naturellement du lemme suivant :

Lemme 4.3.21 (Espace des valuations) — *L'application $\psi : \mathbf{Val}_{\mathcal{A}} \to (\mathcal{V}_\perp \multimap \mathcal{A})$ définie pour tout $\rho \in \mathbf{Val}_{\mathcal{A}}$ par*

$$\psi(\rho) \quad = \quad \big\{(x, \alpha) \in |\mathcal{V}_\perp \multimap \mathcal{A}|\,; \quad \alpha \in \rho(x)\big\}$$

est une bijection de $\mathbf{Val}_{\mathcal{A}}$ *sur* $\mathcal{V}_\perp \multimap \mathcal{A}$ *(où* $\mathcal{V}_\perp$ *désigne l'espace cohérent plat de trame* $\mathcal{V}$*). De plus, pour tous* $\rho, \rho' \in \mathbf{Val}_{\mathcal{A}}$ *on a*

$$\psi(\rho) \;\subset\; \psi(\rho') \qquad \Leftrightarrow \qquad \big(\forall\, x \in \mathcal{V} \quad \rho(x) \subset \rho'(x)\big).$$

Preuve. La bijectivité de ψ est immédiate une fois qu'on a vérifié que la définition ci-dessus envoit toute valuation sur une clique bien formée de $\mathcal{V} \multimap \mathcal{A}$. L'équivalence des relations d'ordre sur $\mathbf{Val}_{\mathcal{A}}$ et $\mathcal{V}_\perp \multimap \mathcal{A}$ par ψ est alors immédiate. □

Dans tout ce qui suit, nous identifierons l'ensemble des valuations $\mathbf{Val}_{\mathcal{A}}$ avec l'espace cohérent $\mathcal{V}_\perp \multimap \mathcal{A}$. En particulier, nous identifierons la valuation constante $(x_{\in\mathcal{V}} \mapsto \varnothing)$ à son image par ψ qui est la clique vide. Les fonctions de projection et d'extension sont linéaires :

Lemme 4.3.22 — *Soient* $x \in \mathcal{V}$ *une variable fixée. Les fonctions* $\mathsf{proj}_x : \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$ *et* $\mathsf{extend}_x : \mathbf{Val}_{\mathcal{A}} \,\&\, \mathcal{A} \to \mathbf{Val}_{\mathcal{A}}$ *définies par*

$$\mathsf{proj}_x(\rho) \;=\; \rho(x) \qquad \textit{et} \qquad \mathsf{extend}_x(\rho, a) \;=\; \rho; x \leftarrow a.$$

sont linéaires, et leurs traces linéaires sont données par :

$$\begin{array}{rcl}
\mathsf{TrLin}(\mathsf{proj_x}) & = & \big\{((x,\alpha),\alpha);\quad \alpha \in |\mathcal{A}|\big\} \\
\mathsf{TrLin}(\mathsf{extend_x}) & = & \big\{\big((1,(y,\alpha)),(y,\alpha)\big);\quad y \neq x,\ \alpha \in |\mathcal{A}|\big\} \cup \\
& & \big\{\big((2,\alpha),(x,\alpha)\big);\quad \alpha \in |\mathcal{A}|\big\}
\end{array}$$

La structure cohérente de l'ensemble des valuations étant dégagée, nous sommes à présent en mesure de construire la fonction d'interprétation $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$. Par récurrence sur le terme M, on définit une fonction μ-stable $[\![M]\!] : \mathbf{Val}_{\mathcal{A}} \xrightarrow{\mu} \mathcal{A}$ en posant :

$$\begin{array}{rcl}
[\![x]\!](\rho) & = & \rho(x) \\
[\![M\ N]\!](\rho) & = & [\![M]\!](\rho) \cdot [\![N]\!](\rho) \\
[\![\lambda x\,.\,M]\!](\rho) & = & \phi^+\Big(\mathsf{Tr}\big(a_{\in\mathcal{A}} \mapsto [\![M]\!](\rho; x \leftarrow a)\big)\Big)
\end{array}$$

Notons qu'il est important de considérer $\mathbf{Val}_{\mathcal{A}}$ avec sa structure d'espace cohérent afin de montrer à chaque étape que la fonction $[\![M]\!] : \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$ est μ-stable. Cette remarque permet essentiellement de justifier que dans la formule définissant l'interprétation de l'abstraction, la fonction $a_{\in\mathcal{A}} \mapsto [\![M]\!](\rho; x \leftarrow a)$ est bien une fonction μ-stable. Dans ce qui suit, on notera $[\![M]\!]_\rho = [\![M]\!](\rho)$.

D'après la définition de la fonction d'interprétation, le triplet $(\mathcal{A}, \cdot, [\![]\!])$ vérifie les axiomes 1 et 2 des λ-modèles. L'axiome 3 découle de l'égalité $\phi^- \circ \phi^+ = \mathsf{id}_{\mathcal{A} \xrightarrow{\mu} \mathcal{A}}$ par

$$\begin{array}{rcl}
[\![\lambda x\,.\,M]\!]_\rho \cdot a & = & \phi^-\Big(\phi^+\big(\mathsf{Tr}(a_{0\in\mathcal{A}} \mapsto [\![M]\!]_{\rho;x\leftarrow a_0})\big)\Big)(a) \\
& = & \big(a_{0\in\mathcal{A}} \mapsto [\![M]\!]_{\rho;x\leftarrow a_0}\big)(a) \;=\; [\![M]\!]_{\rho;x\leftarrow a}
\end{array}$$

et l'axiome 4 résulte immédiatement de la définition de l'interprétation de l'abstraction.

L'opérateur $(a,b) \mapsto a \cdot b$ est clairement monotone, et les interprétations des termes $\lambda x\,.\,x$ et $\lambda f x\,.\,f\ x$ sont données par

$$\begin{aligned}
[\![\lambda x\,.\,x]\!] &= \phi^+\big(\mathsf{Tr}(a_{\in\mathcal{A}} \mapsto a)\big)\\
&= \phi^+\big(\mathsf{Tr}(\mathsf{id}_{\mathcal{A}})\big)\\
[\![\lambda f x\,.\,f\ x]\!] &= \phi^+\Big(\mathsf{Tr}\Big(f_{\in\mathcal{A}} \mapsto \phi^+\big(\mathsf{Tr}\big(a \mapsto \phi^-(f)(a)\big)\big)\Big)\Big)\\
&= \phi^+(\mathsf{Tr}(f_{\in\mathcal{A}} \mapsto \phi^+(\phi^-(f))))\\
&= \phi^+(\mathsf{Tr}(\phi^+ \circ \phi^-))
\end{aligned}$$

d'où il résulte que $[\![\lambda f x\,.\,f\ x]\!] \subset [\![\lambda x\,.\,x]\!]$ en vertu de l'inégalité $\phi^+ \circ \phi^- \leq_{\mathbf{st}} \mathsf{id}_{\mathcal{A}}$, ce qui achève de montrer que le quadruplet $(\mathcal{A}, \cdot, [\![]\!], \subset)$ est un λ-modèle sous-extensionnel. Par ailleurs on a

$$\begin{aligned}
(\mathcal{A}, \cdot, [\![]\!]) \text{ extensionnel} \quad &\Leftrightarrow \quad [\![\lambda f x\,.\,f\ x]\!] \ = \ [\![\lambda x\,.\,x]\!]\\
&\Leftrightarrow \quad \phi^+ \circ \phi^- \ = \ \mathsf{id}_{\mathcal{A}}\\
&\Leftrightarrow \quad \phi \text{ isomorphisme.}
\end{aligned}$$

Chapitre 5

Un modèle restreint classique du calcul implicite

Ce chapitre est consacré à la construction d'un premier modèle non-trivial du calcul implicite, au sens de la définition 3.3.11. Ce modèle basé sur les espaces cohérents et sur la notion de μ-stabilité introduits dans le chapitre précédent a les caractéristiques suivantes.

- Il s'agit d'un modèle 0-restreint du calcul implicite (Def. 3.3.18), puisqu'il comporte un type propositionnel vide. Par conséquent, ce modèle ne valide que les jugements du calcul implicite restreint (Prop. 3.3.17) et invalide la règle de renforcement (Prop. 3.3.19). En revanche, son existence nous permettra de conclure que le calcul implicite restreint est cohérent (corollaire 3.3.20).
- C'est un modèle de la logique classique, puisqu'il valide le tiers-exclu. Les termes de preuves y sont par ailleurs indiscernables (*proof-irrelevance*), puisque les prédicats basés sur des types propositionnels arbitraires sont dénotés par des fonctions constantes. En revanche, contrairement au modèle ensembliste booléen présenté au chapitre 3, indiscernabilité ne signifie pas égalité des dénotations. Dans le modèle que nous allons construire, des termes de preuves différents auront en général des dénotations différentes, et leur indiscernabilité ne proviendra que du fait qu'il n'existe aucune dénotation représentant un prédicat susceptible de les distinguer.

Dans le contexte vide, un terme du calcul implicite en forme normale est soit une fonction (une λ-abstraction) soit un type (une sorte, un produit explicite ou un produit implicite). Dans le modèle que nous nous proposons de construire, les fonctions seront naturellement interprétées par des fonctions μ-stables. Reste donc à définir la notion par laquelle nous interpréterons les types.

Interprétation des types Curieusement, nous ne définirons pas une notion pour interpréter les types, mais deux notions : la notion de *base de type* d'une part, et la notion de *type sémantique* d'autre part. Une telle opposition est nécessaire pour interpréter le sous-typage et ses propriétés à travers les produits dépendants.

Intuitivement, une base de type est une spécification, et représente le contenu intensionnel d'un type. Techniquement, une base de type est un ensemble de cliques deux-à-deux incompatibles (c'est-à-dire des cliques *discernables*), dont les éléments représentent les objets *minimaux*

satisfaisant une certaine spécification. Par exemple, si X est une base de type, on pourra définir une base de type $X \to X$ dont un des éléments est la trace de l'identité définie sur X, et sur X uniquement (ceci afin de satisfaire le critère de minimalité).

Au contraire, un type sémantique représente le contenu extensionnel d'un type, c'est-à-dire l'ensemble des cliques d'un espace cohérent satisfaisant une certaine spécification. À chaque base de type est associé un type sémantique, qui n'est autre que sa clôture supérieure dans un espace cohérent donné. Ainsi, le type sémantique associé à la base de type $X \to X$ contiendra non seulement la fonction identité définie sur X, mais aussi toutes les (traces des) fonctions qui la prolongent. Parmi ces fonctions qui prolongent la fonction identité restreinte à X (et que l'on ne pourra pas distinguer les unes des autres au sein du type sémantique associé à $X \to X$), on trouvera naturellement la fonction identité "générique" qui est définie sur tout le modèle. Par conséquent, la fonction identité générique sera considérée comme un objet de type $X \to X$, car elle appartient au type sémantique associé à la base de type $X \to X$.

Une définition ouverte Un des aspects les plus notables de l'opposition entre bases de types et types sémantiques[1] est qu'à travers cette opposition, les types acquièrent naturellement une définition ouverte, dans un sens que nous allons préciser.

Dans les modèles ensemblistes de CCω, les types sont interprétés par des ensembles formés par tous les objets qui habitent ce type. Autrement dit, un type est interprété par son contenu extensionnel. Une telle définition est fermée en ce sens que, une fois qu'un ensemble représentant un certain type a été défini, il n'est plus possible d'en changer le contenu et d'y incorporer de nouveaux objets, sauf à redéfinir cet ensemble. C'est précisément cette conception des types comme des objets dont le contenu est figé une fois pour toutes qui explique l'absence d'interprétation satisfaisante du sous-typage en théorie des ensembles.

Avec l'opposition entre bases de types et types sémantiques, la situation devient beaucoup plus intéressante. Dans le modèle que nous allons construire, les types seront représentés intensionnellement par des bases de types.[2] La notion de base de type — liée à l'idée de spécification — est invariante par plongement rigide, ce qui signifie qu'une base de type donnée dans un espace cohérent donné sera toujours une base de type dans n'importe quelle extension de l'espace cohérent de départ. En revanche, le contenu extensionnel lié à une base de type (c'est-à-dire le type sémantique associé, qui est sa clôture supérieure) variera avec l'espace cohérent dans lequel on la considère. Une même base de type aura donc une clôture supérieure d'autant plus grande que l'espace cohérent dans lequel on la calcule est grand. Intuitivement, ceci illustre l'idée — très intuitive — selon laquelle une même spécification est réalisée par davantage d'objets sitôt qu'on étend l'univers du discours.[3]

Ainsi que nous le verrons tout au long de ce chapitre, cette opposition nous permettra d'interpréter de manière satisfaisante de nombreuses subtilités liées au sous-typage dans le calcul implicite, telles que :

[1] On retrouve également une telle opposition dans les travaux de J. Y. Girard autour de la Ludique [30] avec les notions de comportement et d'incarnation, qui sont un peu les analogues des types sémantiques et des bases de types que nous allons présenter ici, et dont les propriétés formelles sont très proches.

[2] En particulier, les types comme *citoyens de première classe* seront définis à l'aide des bases de types, et non à l'aide des types sémantiques correspondants.

[3] Ou, s'il l'on préfère cette image un peu triviale, qu'il y a beaucoup plus de voitures rouges dans le monde qu'il n'y a de voitures rouges en France.

- Les variances du sous-typage dans les produits dépendants, et en particulier la propriété de contravariance vis-à-vis du domaine.
- Le comportement du sous-typage vis-à-vis des univers, et en particulier le fait qu'un type défini dans un univers élevé puisse être un sous-type d'un type défini dans un univers plus bas (par exemple, le fait que $\forall A : \mathsf{Type}_1 \,.\, A \to A$, défini dans Type_2, est un sous-type de $\mathsf{Prop} \to \mathsf{Prop}$, défini dans Type_1).

Par ailleurs, la structure des bases de types et des types sémantiques permettra également d'expliquer le fait que deux mêmes objets puissent être à la fois égaux dans un certain type (par exemple les termes $\lambda xy \,.\, x$ et $\lambda xy \,.\, y$ dans le type des booléens de Prop définis par codage imprédicatif) et différents dans un autre type (les mêmes termes, dans le type des booléens de Type_2 défini par codage prédicatif), ainsi que nous le verrons au paragraphe 5.2.5.

5.1 Types dans les espaces cohérents

5.1.1 Bases de types et types sémantiques

Définition 5.1.1 (Bases de types et types sémantiques) — Soit $\mathcal{A}$ un espace cohérent. On dit d'un ensemble de cliques $X \subset \mathcal{A}$ qu'il est

- une *base de type de* $\mathcal{A}$ si pour toutes cliques $a_1, a_2 \in X$ on a

$$a_1 \cup a_2 \in \mathcal{A} \quad \Rightarrow \quad a_1 = a_2;$$

- un *type sémantique* de $\mathcal{A}$ si X est clos supérieurement dans $\mathcal{A}$ et si pour toute famille non-vide $(a_i)_{i \in I}$ d'éléments de X on a

$$\bigcup_{i \in I} a_i \;\in\; \mathcal{A} \quad \Rightarrow \quad \bigcap_{i \in I} a_i \;\in\; X.$$

Informellement, une base de type est un ensemble de cliques deux à deux incompatibles, tandis qu'un type sémantique est un sous-ensemble clos supérieurement dans $\mathcal{A}$ et stable par intersection compatible arbitraire. Ces deux notions — qui à première vue ne semblent avoir que peu de rapports entre elles — sont en fait profondément reliées ainsi que le montre la proposition suivante :

Proposition 5.1.2 — *Soit $\mathcal{A}$ un espace cohérent.*

1. *Pour toute base de type $X \subset \mathcal{A}$, la clôture supérieure de X dans $\mathcal{A}$, notée $\uparrow X$, est un type sémantique de $\mathcal{A}$.*
2. *Pour tout type sémantique $Y \subset \mathcal{A}$, l'ensemble des éléments minimaux de Y, noté $\lfloor Y \rfloor$, constitue une base de type de $\mathcal{A}$.*
3. *Les correspondances $X \mapsto \uparrow X$ et $Y \mapsto \lfloor Y \rfloor$ sont réciproques l'une de l'autre, et constituent une bijection entre l'ensemble des bases de types de $\mathcal{A}$ et l'ensemble des types sémantiques de $\mathcal{A}$.*

Preuve. 1. Soit $X \subset \mathcal{A}$ une base de type, et $\uparrow X$ sa clôture supérieure dans $\mathcal{A}$. Pour tout $a \in \uparrow X$, il existe par définition de la clôture supérieure une clique $a_0 \in X$ telle que $a_0 \subset a$. Par ailleurs, cette clique a_0 est unique, car si l'on suppose l'existence d'une autre clique $a_0' \in X$

telle que $a'_0 \subset a$, les cliques $a_0, a'_0 \in X$ sont manifestement compatibles, d'où il ressort que $a_0 = a'_0$ d'après la définition des bases de types. Dans ce qui suit, on notera $\lfloor a \rfloor_X$ l'unique clique $a_0 \in X$ telle que $a_0 \subset a$. Considérons à présent une famille non-vide $(a_i)_{i\in I}$ de cliques de $\uparrow X$ compatibles dans $\mathcal{A}$. Pour tous $i, j \in I$, on a $\lfloor a_i \rfloor_X \subset a_i$ et $\lfloor a_j \rfloor_X \subset a_j$ d'où il ressort que les cliques $\lfloor a_i \rfloor_X$ et $\lfloor a_j \rfloor_X$ sont compatibles et donc égales. Par conséquent, toutes les cliques $\lfloor a_i \rfloor_X$ $(i \in I)$ sont égales à une même clique $a_0 \in X$, telle que $a_0 \subset a_i$ pour tout $i \in I$. On a donc $a_0 \subset \bigcap_{i\in I} a_i$, ce qui prouve que l'intersection de la famille $(a_i)_{i\in I}$ appartient à $\uparrow X$.

2. Soit Y un type sémantique de $\mathcal{A}$, et $\lfloor Y \rfloor$ l'ensemble de ses éléments minimaux. Soient a_1 et a_2 deux cliques compatibles de $\lfloor Y \rfloor \subset Y$. D'après la définition des types sémantiques, on a $a_1 \frown a_2 \in Y$, d'où $a_1 = a_1 \frown a_2 = a_2$ puisque a_1 et a_2 sont des cliques minimales dans Y. Donc $\lfloor Y \rfloor$ est une base de type.

3. Si $X \subset \mathcal{A}$ est une base de type, tous les éléments de X sont incomparables entre eux et donc minimaux dans X. Par conséquent, on a $\lfloor \uparrow X \rfloor = X$. Soit $Y \subset \mathcal{A}$ un type sémantique. L'inclusion $\uparrow \lfloor Y \rfloor \subset Y$ étant immédiate, il suffit pour établir l'égalité de montrer que pour tout $a \in Y$, il existe un élément minimal $a_0 \in \lfloor Y \rfloor$ tel que $a_0 \subset a$. Pour cela, considérons une clique $a \in Y$ ainsi que l'ensemble $Z = \{a' \in Y; \quad a' \subset a\}$. Z est un sous-ensemble non-vide de Y dont les éléments sont des cliques compatibles entre elles, et dont l'intersection a_0 est par conséquent un élément de Y d'après la définition des types sémantiques. Il est clair que a_0 est minimal dans Y, d'où $a_0 \in \lfloor Y \rfloor$ et $a_0 \subset a$. □

Lemme 5.1.3 (Coercition) — *Soit X une base de type d'un espace cohérent $\mathcal{A}$. Pour toute clique $a \in \uparrow X$, il existe une unique clique $a_0 \in X$ telle que $a_0 \subset a$, que l'on note $a_0 = \lfloor a \rfloor_X$.*

Preuve. L'existence de a_0 est immédiate, et son unicité a déjà été établie au cours de la preuve de la proposition précédente. □

Définition 5.1.4 (Égalité modulo X) — Soit $X \subset \mathcal{A}$ une base de type. On note $a_1 =_X a_2$ la relation binaire sur $\mathcal{A}$ définie par

$$a_1 =_X a_2 \qquad \equiv \qquad \exists\, a_0 \in X \qquad a_0 \subset a_1 \quad \text{et} \quad a_0 \subset a_2.$$

Remarquons que de manière équivalente, on a pour tous $a_1, a_2 \in \mathcal{A}$

$$a_1 =_X a_2 \qquad \Leftrightarrow \qquad a_1, a_2 \in \uparrow X \quad \text{et} \quad \lfloor a_1 \rfloor_X = \lfloor a_2 \rfloor_X,$$

d'où il ressort clairement que pour toute base de type $X \subset \mathcal{A}$, la relation $(=_X)$ est une relation d'équivalence partielle sur $\mathcal{A}$ dont le domaine est le type sémantique $\uparrow X$ associé à X.

Pour toute base de type $X \subset \mathcal{A}$, le type sémantique $\uparrow X$ se décompose en l'union disjointe des classes d'équivalence de la relation $a_1 =_X a_2$

$$\uparrow X \quad = \quad \bigcup_{a\in X} \uparrow\{a\} \qquad \text{(union disjointe)}$$

où pour chaque clique $a \in X$, $\uparrow\{a\}$ désigne le *type sémantique principal* de la clique a, dont la base de type est le singleton $\{a\}$, que nous qualifierons par la suite de *base de type principale de a*. Au paragraphe 5.2.5, nous verrons qu'au sein du type sémantique $\uparrow X$ les éléments d'une même classe $\uparrow\{a\}$ sont indiscernables.

Proposition 5.1.5 (Invariance des bases de types par inclusion rigide) — *Soit $\mathcal{A}$ et $\mathcal{A}'$ des espaces cohérents tels que $\mathcal{A} \Subset \mathcal{A}'$. Pour tout ensemble de cliques $X \subset \mathcal{A}$, on a l'équivalence*

$$X \text{ base de type de } \mathcal{A} \quad \Leftrightarrow \quad X \text{ base de type de } \mathcal{A}'.$$

Preuve. Cette propriété est immédiate du fait que les relations de cohérence issues de $\mathcal{A}$ et de $\mathcal{A}'$ coïncident sur $|\mathcal{A}|$. □

En revanche, un type sémantique $Y \subset \mathcal{A}$ n'est généralement ni une base de type ni un type sémantique dans un espace cohérent $\mathcal{A}'$ tel que $\mathcal{A} \Subset \mathcal{A}'$ (la notion de clôture supérieure n'est pas préservée par plongement rigide). La propriété d'invariance énoncée ci-dessus ne vaut donc que pour les bases de types. C'est pour cette raison que dans ce qui suit, nous travaillerons de préférence avec des bases de types plutôt qu'avec les types sémantiques qui leur sont associés. En particulier, lorsque viendra le moment de considérer dans le modèle les types comme des objets de première classe, il sera naturel de coder cette notion à l'aide des bases de types correspondantes. Cependant, la relation de typage s'interprétera par l'appartenance au type sémantique associé à une base de type donnée.

La notion de type sémantique est intimement liée à la quasi-stabilité. La proposition suivante montre qu'en fait, chacune de ces deux notions peut être définie à partir de l'autre :

Proposition 5.1.6 (Types sémantiques et quasi-stabilité) — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents.*

1. *Une fonction $F : \mathcal{A} \to \mathcal{B}$ est quasi-stable si et seulement si l'image réciproque de tout type sémantique de $\mathcal{B}$ par F est un type sémantique de $\mathcal{A}$.*
2. *Un ensemble de cliques $Y \subset \mathcal{A}$ est un type sémantique de $\mathcal{A}$ si et seulement si sa fonction caractéristique $F : \mathcal{A} \to \mathbf{1}$ définie par*

$$F(a) \quad = \quad \begin{cases} \{\bullet\} & \text{si } a \in Y \\ \varnothing & \text{sinon} \end{cases}$$

est une fonction quasi-stable. Lorsque c'est le cas, la trace de cette fonction caractéristique est isomorphe à la base de type associée à Y :

$$\mathsf{Tr}(F) \quad = \quad \big\{(a, \bullet); \quad a \in \lfloor Y \rfloor\big\}.$$

Nous ne donnerons pas ici la preuve — élémentaire mais fastidieuse — de ce résultat dont nous n'aurons pas à nous servir par la suite.

5.1.2 Sous-typage

Définition 5.1.7 (Sous-typage) — Soient $\mathcal{A}$ un espace cohérent, X et X' des bases de types de $\mathcal{A}$. On dit que X est un *sous-type* de Y si $\uparrow X \subset \uparrow Y$, ce que l'on note $X \leq Y$.

La relation de sous-typage est une relation d'ordre sur les bases de types, qui est dérivée de l'ordre de l'inclusion sur les types sémantiques correspondants. Remarquons que si X et Y sont des bases de types, $X \subset Y$ entraîne $X \leq Y$, mais la réciproque est fausse en général.[4]

[4]Le cas particulier de l'inclusion de bases de types correspondra à l'interprétation de la relation de cumulativité dans les modèles que nous construirons dans ce chapitre ainsi que dans les suivants.

Lemme 5.1.8 (Sous-typage par coercition) — *Pour toutes bases de types $X, Y \subset \mathcal{A}$ on a l'équivalence :*

$$X \leq Y \qquad \Leftrightarrow \qquad \forall\, a \in X \quad \exists a' \in Y \quad a' \subset a.$$

De plus, lorsque $X \leq Y$, on a $\lfloor a \rfloor_Y \subset \lfloor a \rfloor_X$ pour tout $a \in \uparrow X$.

Preuve. La preuve de ce résultat est élémentaire, et résulte immédiatement de la définition du sous-typage. □

Remarquons que la relation de sous-typage $X \leq Y$ est définie dans un espace cohérent $\mathcal{A}$ donné, par l'inclusion $\uparrow X \subset \uparrow Y$ des clôtures supérieures de X et de Y calculées dans $\mathcal{A}$. Cette relation dépend donc *a priori* de l'espace $\mathcal{A}$, et devrait être notée $X \leq_{\mathcal{A}} Y$ en toute rigueur. Cependant, le lemme précédent caractérise le sous-typage indépendamment de l'espace $\mathcal{A}$, d'où la proposition suivante :

Proposition 5.1.9 (Indépendance de l'univers du discours) — *Soient $\mathcal{A}$ et $\mathcal{A}'$ des espaces cohérents tels que $\mathcal{A} \Subset \mathcal{A}'$. Pour toutes bases de types $X, Y \subset \mathcal{A}$ on a*

$$X \leq_{\mathcal{A}} Y \qquad \Leftrightarrow \qquad X \leq_{\mathcal{A}'} Y.$$

5.1.3 Intersection et produit dépendant

Proposition 5.1.10 (Intersection et produit dépendant) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a \in X}$ une famille de bases de types de $\mathcal{B}$ indicée par la base de type X.*

- *L'intersection des types sémantiques $\uparrow Y_a$ lorsque a décrit X constitue un type sémantique de $\mathcal{B}$, dont la base de type est notée $\forall(a{\in}X; Y_a)$.*
- *L'ensemble des traces des fonctions quasi-stables $F : \mathcal{A} \xrightarrow{q} \mathcal{B}$ telles que $F(a) \in \uparrow Y_a$ pour tout $a \in X$ constitue un type sémantique de $\mathcal{A} \xrightarrow{q} \mathcal{B}$, dont la base de type est notée $\Pi(a{\in}X; Y_a)$.*

Preuve. Le premier point est immédiat, car la notion de type sémantique est clairement stable par intersection (y compris vide, puisque l'espace $\mathcal{A}$ entier est un type sémantique). Considérons une famille de fonctions quasi-stables $F_i : \mathcal{A} \xrightarrow{q} \mathcal{B}$ indicées par un ensemble I non vide, telles que $F_i(a) \in \uparrow Y_a$ pour tout $a \in X$ et pour tout $i \in I$, et supposons en outre que $(F_i)_{i \in I}$ est une famille de traces compatibles (en identifiant chaque F_i avec sa trace). Soit $a \in X$ fixé. La famille des $F_i(a)$ $(i \in I)$ est compatible, car $F_i(a) \subset \big(\bigcup_{j \in I} F_j\big)(a)$ pour tout $i \in I$, et par conséquent $\bigcap_{i \in I} F_i(a) \in \uparrow Y_a$ puisque $\uparrow Y_a$ est un type sémantique. Comme en outre l'application de fonction est une fonction binaire quasi-stable, on a

$$\Big(\bigcap_{i \in I} F_i\Big)(a) \quad = \quad \bigcap_{i \in I} F_i(a) \quad \in \quad \uparrow Y_a,$$

et cela pour tout $a \in X$. □

Les constructions $\Pi(a{\in}X; Y_a)$ et $\forall(a{\in}X; Y_a)$ nous permettront par la suite d'interpréter le produit explicite et le produit implicite respectivement. Remarquons que pour tout $b \in \mathcal{B}$

et pour tout $f \in \mathcal{A} \xrightarrow{q} \mathcal{B}$ on a les équivalences :

$$\begin{array}{lcll} b \ \in \ \uparrow\forall(a{\in}X; Y_a) & \Leftrightarrow & \forall\, a \in X & b \ \in \ \uparrow Y_a \\ & \Leftrightarrow & \forall\, a \in \uparrow X & b \ \in \ \uparrow Y_{\lfloor a \rfloor_X} \\ f \ \in \ \uparrow\Pi(a{\in}X; Y_a) & \Leftrightarrow & \forall\, a \in X & f(a) \ \in \ \uparrow Y_a \\ & \Leftrightarrow & \forall\, a \in \uparrow X & f(a) \ \in \ \uparrow Y_{\lfloor a \rfloor_X} \end{array}$$

(la dernière équivalence résultant de la monotonie de f et du fait que les ensembles $\uparrow Y_a$ sont clos supérieurement dans $\mathcal{A}$).

Proposition 5.1.11 (Covariance/contravariance) — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents, X et X' des bases de types de $\mathcal{A}$, $(Y_a)_{a\in X}$ et $(Y'_a)_{a\in X'}$ deux familles de bases de types de $\mathcal{B}$ indicées par X et X' respectivement. Si $X' \leq X$ et si pour tout $a' \in X'$ on a $Y_{\lfloor a' \rfloor_X} \leq Y'_{a'}$, alors*

$$\Pi(a{\in}X; Y_a) \ \leq \ \Pi(a{\in}X'; Y'_a) \qquad et \qquad \forall(a{\in}X; Y_a) \ \leq \ \forall(a{\in}X'; Y'_a).$$

Preuve. Cette proposition résulte immédiatement de la définition du produit dépendant, de l'intersection et du sous-typage. □

Remarquons que les constructions $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$ opèrent sur des bases de types et retournent des bases de types. Cependant, la définition des bases de types $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$ fait intervenir explicitement les types sémantiques $\uparrow Y_a$, dont le contenu dépend de l'espace $\mathcal{B}$ dans lequel on se place. Pour cette raison, il serait plus prudent d'utiliser les notations $\forall^{\mathcal{B}}(a{\in}X; Y_a)$ et $\Pi^{\mathcal{A}\xrightarrow{q}\mathcal{B}}(a{\in}X; Y_a)$ indiquant en exposant l'espace de calcul. Heureusement, les bases de types qui résultent des définitions du produit dépendant et de l'intersection ne dépendent pas de cet espace de calcul.

Proposition 5.1.12 (Indépendance de l'univers du discours) — *Soient $\mathcal{A}$, $\mathcal{A}'$, $\mathcal{B}$ et $\mathcal{B}'$ des espaces cohérents tels que $\mathcal{A} \Subset \mathcal{A}'$ et $\mathcal{B} \Subset \mathcal{B}'$. Si X est une base de type de $\mathcal{A}$ et si $(Y_a)_{a\in X}$ est une famille de bases de types de $\mathcal{B}$ indicée par X, on a alors*

$$\forall^{\mathcal{B}}(a{\in}X; Y_a) \ = \ \forall^{\mathcal{B}'}(a{\in}X; Y_a) \qquad et \qquad \Pi^{\mathcal{A}\xrightarrow{q}\mathcal{B}}(a{\in}X; Y_a) \ = \ \Pi^{\mathcal{A}'\xrightarrow{q}\mathcal{B}'}(a{\in}X; Y_a).$$

Preuve. (*Intersection*) Soient Z et Z' les ensembles définis par

$$\begin{array}{rcl} Z & = & \big\{b \in \mathcal{B};\quad \forall\, a \in X \quad b \in \uparrow^{\mathcal{B}} Y_a\big\} \\ Z' & = & \big\{b' \in \mathcal{B}';\quad \forall\, a \in X \quad b' \in \uparrow^{\mathcal{B}'} Y_a\big\} \end{array}$$

(où $\uparrow^{\mathcal{B}} Y_a$ et $\uparrow^{\mathcal{B}'} Y_a$ désignent les clôtures supérieures de Y_a prises dans $\mathcal{B}$ et $\mathcal{B}'$ respectivement). D'après la proposition 5.1.10, le sous-ensemble $Z \subset \mathcal{B}$ (resp. $Z' \subset \mathcal{B}'$) est un type sémantique de $\mathcal{B}$ (resp. de $\mathcal{B}'$) dont la base de type est $\forall^{\mathcal{B}}(a{\in}X; Y_a)$ (resp. $\forall^{\mathcal{B}'}(a{\in}X; Y_a)$). Par ailleurs, on a l'inclusion $Z \subset Z'$. Notons $\phi^- : \mathcal{B}' \multimap \mathcal{B}$ la rétraction de $\mathcal{B}'$ sur $\mathcal{B}$ associée à l'inclusion rigide $\mathcal{B} \Subset \mathcal{B}'$, laquelle est définie pour tout $b' \in \mathcal{B}'$ par $\phi^-(b') = b' \cap |\mathcal{B}|$. On remarque alors que pour tout $b' \in \mathcal{B}'$, on a

$$\begin{array}{lcl} b' \in Z' & \Rightarrow & \forall\, a \in X \quad b' \in \uparrow^{\mathcal{B}'} Y_a \\ & \Rightarrow & \forall\, a \in X \quad b' \cap |\mathcal{B}| \ \in \ (\uparrow^{\mathcal{B}'} Y_a) \cap |\mathcal{B}| \\ & \Rightarrow & \forall\, a \in X \quad \phi^-(b') \in \uparrow^{\mathcal{B}} Y_a \\ & \Rightarrow & \phi^-(b') \in Z, \end{array}$$

ce qui montre que ϕ^- envoie Z' dans Z.

Soit b un élément de $\forall^{\mathcal{B}}(a{\in}X; Y_a)$, c'est-à-dire un élément minimal de Z. Considérons une clique $b' \in Z'$ telle que $b' \subset b$. On a $\phi^-(b') \in Z$ et $\phi^-(b') \subset b' \subset b$, d'où $\phi^-(b') = b' = b$ d'après l'hypothèse de minimalité (dans Z) effectuée sur b, ce qui prouve que b est également minimal dans Z'. D'où l'inclusion $\forall^{\mathcal{B}}(a{\in}X; Y_a) \subset \forall^{\mathcal{B}'}(a{\in}X; Y_a)$.

Réciproquement, soit b' un élément de $\forall^{\mathcal{B}'}(a{\in}X; Y_a)$, c'est-à-dire un élément minimal de Z'. On a $\phi^-(b') \in Z \subset Z'$ et $\phi^-(b') \subset b$, d'où $\phi^-(b') = b'$ d'après l'hypothèse de minimalité (dans Z') effectuée sur b'. On a par conséquent $b' \in Z$, et il est clair que b' est également minimal dans Z, puisque $Z \subset Z'$. D'où l'inclusion $\forall^{\mathcal{B}'}(a{\in}X; Y_a) \subset \forall^{\mathcal{B}}(a{\in}X; Y_a)$.

(*Produit dépendant*) On considère à présent les ensembles Z et Z' définis par

$$\begin{aligned} Z &= \big\{ f \in \mathcal{A} \xrightarrow{q} \mathcal{B};\quad \forall\, a \in X \quad f(a) \in {\uparrow^{\mathcal{B}}} Y_a \big\} \\ Z' &= \big\{ f' \in \mathcal{A}' \xrightarrow{q} \mathcal{B}';\quad \forall\, a \in X \quad f'(a) \in {\uparrow^{\mathcal{B}'}} Y_a \big\} \end{aligned}$$

D'après la proposition 5.1.10, le sous-ensemble $Z \subset \mathcal{A} \xrightarrow{q} \mathcal{B}$ (resp. $Z' \subset \mathcal{A}' \xrightarrow{q} \mathcal{B}'$) est un type sémantique de $\mathcal{A} \xrightarrow{q} \mathcal{B}$ (resp. $\mathcal{A}' \xrightarrow{q} \mathcal{B}'$) dont la base de type est $\Pi^{\mathcal{A} \xrightarrow{q} \mathcal{B}}(a{\in}X; Y_a)$ (resp. $\Pi^{\mathcal{A}' \xrightarrow{q} \mathcal{B}'}(a{\in}X; Y_a)$), et on a clairement l'inclusion $Z \subset Z'$. Notons à présent

$$\phi^- \;:\; (\mathcal{A}' \xrightarrow{q} \mathcal{B}') \;\multimap\; (\mathcal{A} \xrightarrow{q} \mathcal{B})$$

la rétraction de $\mathcal{A}' \xrightarrow{q} \mathcal{B}'$ sur $\mathcal{A} \xrightarrow{q} \mathcal{B}$ associée à l'inclusion rigide $(\mathcal{A} \xrightarrow{q} \mathcal{B}) \Subset (\mathcal{A}' \xrightarrow{q} \mathcal{B}')$. Cette rétraction ϕ^- est définie pour tout $f' \in \mathcal{A}' \xrightarrow{q} \mathcal{B}'$ par $\phi^-(f') = f' \cap \big|\mathcal{A} \xrightarrow{q} \mathcal{B}\big|$ et caractérisée par $\phi^-(f')(a) = f'(a) \cap |\mathcal{B}|$ pour tout $a \in \mathcal{A}$. Pour tout $f' \in \mathcal{A}' \xrightarrow{q} \mathcal{B}'$, on a alors

$$\begin{array}{lll} f' \in Z' & \Rightarrow & \forall\, a \in X \quad f'(a) \in {\uparrow^{\mathcal{B}'}} Y_a \\ & \Rightarrow & \forall\, a \in X \quad f'(a) \cap |\mathcal{B}| \;\in\; ({\uparrow^{\mathcal{B}'}} Y_a) \cap |\mathcal{B}| \\ & \Rightarrow & \forall\, a \in X \quad \phi^-(f')(a) \;\in\; {\uparrow^{\mathcal{B}}} Y_a \\ & \Rightarrow & \phi^-(f') \in Z, \end{array}$$

ce qui montre que ϕ^- envoie Z' dans Z. Le reste de la démonstration se conduit alors de la même façon que dans le cas de l'intersection. □

Si $\mathcal{A}$ est un espace cohérent et $X \subset \mathcal{A}$ un sous-ensemble arbitraire de $\mathcal{A}$, on notera par la suite $|X|$ (par analogie avec la notation $|\mathcal{A}|$) l'union des éléments de X, c'est-à-dire l'ensemble des atomes de $\mathcal{A}$ figurant dans au moins une clique appartenant à X.

La proposition précédente nous permet de préciser le contenu atomique des bases de types $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$ de la manière suivante :

Lemme 5.1.13 (Atomes du produit dépendant et de l'intersection) — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents. Si X est une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X, on a :*

$$|\forall(a{\in}X; Y_a)| \;\subset\; \bigcup_{a\in X} |Y_a| \qquad \textit{et} \qquad |\Pi(a{\in}X; Y_a)| \;\subset\; \mathfrak{P}(|X|) \times \bigcup_{a\in X} |Y_a|\,.$$

Preuve. Soient $\mathcal{A}'$ le sous-espace de $\mathcal{A}$ engendré par $|X|$ et $\mathcal{B}'$ le sous-espace de $\mathcal{B}$ engendré par l'union des $|Y_a|$ lorsque $a \in X$. On a par définition $X \subset \mathcal{A}' \Subset \mathcal{A}$ et $Y_a \subset \mathcal{B}' \Subset \mathcal{B}$ pour tout $a \in X$. D'après la proposition 5.1.12 on a alors

$$\forall(a{\in}X;Y_a) \;=\; \forall^{\mathcal{B}'}(a{\in}X;Y_a) \qquad \text{et} \qquad \Pi(a{\in}X;Y_a) \;=\; \Pi^{\mathcal{A}'\xrightarrow{q}\mathcal{B}'}(a{\in}X;Y_a),$$

d'où l'on tire le résultat attendu, en remarquant simplement que

$$\big|\mathcal{A}'\big| \;=\; |X|\,; \qquad \big|\mathcal{B}'\big| \;=\; \bigcup_{a\in X}|Y_a| \qquad \text{et} \qquad \Big|\mathcal{A}' \xrightarrow{q} \mathcal{B}'\Big| \;\subset\; \mathfrak{P}(|X|)\times\bigcup_{a\in X}|Y_a|\,. \qquad \square$$

5.1.4 Une caractérisation interne de $\forall(a{\in}X;Y_a)$ et $\Pi(a{\in}X;Y_a)$

La définition des bases de types $\forall(a{\in}X;Y_a)$ et $\Pi(a{\in}X;Y_a)$ est une définition *externe*, en ce sens qu'elle recourt à la famille de types sémantiques $\uparrow Y_a$ dont le contenu atomique dépend de l'espace cohérent $\mathcal{B}$ dans lequel on calcule les clôtures supérieures $\uparrow Y_a$. Cependant, la proposition 5.1.12 montre que le contenu des bases de types $\forall(a{\in}X;Y_a)$ et $\Pi(a{\in}X;Y_a)$ ne dépend pas des espaces cohérents $\mathcal{A}$ et $\mathcal{B}$, et le lemme 5.1.13 exploite cette idée en montrant que le contenu atomique des bases de types $\forall(a{\in}X;Y_a)$ et $\Pi(a{\in}X;Y_a)$ n'est construit qu'à partir des contenus atomiques des bases de types X et Y_a ($a \in X$).

Ceci suggère naturellement la possibilité de donner une caractérisation *interne* des bases de types $\forall(a{\in}X;Y_a)$ et $\Pi(a{\in}X;Y_a)$, en décrivant le contenu atomique de leurs cliques de manière directe à partir des cliques des bases de types X et Y_a, sans jamais recourir à des atomes figurant en dehors.

Dans la suite de ce paragraphe, $\mathcal{A}$ et $\mathcal{B}$ désignent des espaces cohérents.

Caractérisation interne de l'intersection Soient X une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$. Par définition, $\forall(a{\in}X;Y_a)$ est le plus grand sous-type commun aux bases de types Y_a ($a \in X$). La coercition d'une clique $b \in \uparrow\forall(a{\in}X;Y_a)$ s'exprime de la manière suivante :

Lemme 5.1.14 (Coercition dans $\forall(a \in X;Y_a)$) — *Pour tout $b \in \uparrow\forall(a{\in}X;Y_a)$, on a*

$$\lfloor b\rfloor_{\forall(a\in X;Y_a)} \;=\; \bigcup_{a\in X}\lfloor b\rfloor_{Y_a}.$$

Preuve. Soit $b \in \uparrow\forall(a{\in}X;Y_a)$, et posons $b_0 = \lfloor b\rfloor_{\forall(a\in X;Y_a)}$. Comme $\forall(a{\in}X;Y_a)$ est un sous-type de chacune des bases de types Y_a, on a $\lfloor b\rfloor_{Y_a} \subset b_0$ pour tout $a \in X$, d'où $\bigcup\lfloor b\rfloor_{Y_a} \subset b_0$. Par ailleurs, la clique $\bigcup\lfloor b\rfloor_{Y_a}$ appartient à tous les types sémantiques $\uparrow Y_a$ lorsque a décrit X, donc au type sémantique $\uparrow\forall(a{\in}X;Y_a)$. Comme par définition, b_0 est minimal dans le type sémantique $\uparrow\forall(a{\in}X;Y_a)$, on a donc l'égalité $\bigcup\lfloor b\rfloor_{Y_a} = b_0 = \lfloor b\rfloor_{\forall(a\in X;Y_a)}$. $\square$

Proposition 5.1.15 (Caractérisation de $\forall(a \in X;Y_a)$) — *Pour toute clique $b \in \mathcal{B}$, on a*

$$b \;\in\; \forall(a{\in}X;Y_a) \qquad \Leftrightarrow \qquad \exists\,(b_a)_{a\in X} \;\in\; \prod_{a\in X} Y_a \qquad b = \bigcup_{a\in X} b_a.$$

Preuve. Cette proposition est une conséquence immédiate du lemme précédent. $\square$

Caractérisation interne du produit dépendant La caractérisation interne du produit dépendant $\Pi(a{\in}X; Y_a)$ est plus complexe que celle de l'intersection $\forall(a{\in}X; Y_a)$, et fait intervenir plusieurs étapes successives de décomposition. La première d'entre-elles est la décomposition de la base de type $\Pi(a{\in}X; Y_a)$ à partir des opérations $\forall$ et $\to$, où $\to$ désigne l'opérateur de produit *non-dépendant*, défini par $X \to Y = \Pi(a{\in}X; Y)$ et caractérisé par

$$\begin{array}{rcl} f \in \uparrow(X \to Y) & \Leftrightarrow & \forall\, a \in X \quad f(a) \ \in \ \uparrow Y \\ & \Leftrightarrow & \forall\, a \in \uparrow X \quad f(a) \ \in \ \uparrow Y \end{array}$$

Lemme 5.1.16 (Décomposition de $\Pi(a{\in}X; Y_a)$) — *Si X est une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X, alors*

$$\Pi(a{\in}X; Y_a) \quad = \quad \forall(a{\in}X; \{a\} \to Y_a),$$

où le singleton $\{a\}$ désigne la base de type principale de la clique $a \in X$.

Preuve. Immédiat d'après la définition du produit dépendant et de l'intersection. □

Notons que la décomposition ci-dessus s'applique également au produit non-dépendant

$$X \to Y \quad = \quad \forall(a{\in}X; \{a\} \to Y),$$

ce qui permet dans tous les cas de décomposer le produit (dépendant ou non-dépendant) à partir d'une construction plus primitive $\{a\} \to Y$.

Lemme 5.1.17 (Cliques de la base de type $\{a\} \to Y$) — *Soient a une clique de $\mathcal{A}$ et Y une base de type de $\mathcal{B}$. Si $f \in \{a\} \to Y$, alors il existe une clique $b \in Y$ et une famille $(a_\beta)_{\beta\in b}$ de sous-cliques de a indicée par b telle que*

$$f \ = \ \{(a_\beta, \beta); \quad \beta \in b\}.$$

Preuve. Supposons $f \in \{a\} \to Y$. D'après le lemme 5.1.13, on a $a_0 \subset a$ et $\beta \in |Y|$ pour tout atome $(a_0, \beta) \in f$. Notons alors $b = f(a)$. Par définition de $\{a\} \to Y$, on a $b \in \uparrow Y$. Il est alors clair que pour tout $\beta \in b$, il existe une clique $a_0 \subset a$ telle que $(a_0, \beta) \in f$. Par ailleurs, s'il existe une autre clique $a_1 \subset a$ telle que $(a_1, \beta) \in f$, on a $a_1 = a_0$ d'après la définition de la relation de cohérence sur $\mathcal{A} \xrightarrow{q} \mathcal{B}$, ce qui montre que la clique a_0 dépend fonctionnellement de l'atome $\beta \in b$. Pour tout $\beta \in b$, notons donc a_β l'unique clique $a_\beta \subset a$ telle que $(a_\beta, \beta) \in f$. Pour achever la démonstration du lemme, il s'agit de vérifier que b n'est pas seulement une clique du type sémantique $\uparrow Y$, mais une clique de la base de type Y, c'est-à-dire $b = \lfloor b \rfloor_Y$. Pour cela, considérons la fonction f_0 définie par la trace

$$f_0 \ = \ \{(a_\beta, \beta); \quad \beta \in \lfloor b \rfloor_Y\} \ \subset \ f.$$

On a alors immédiatement $f_0(a) = \lfloor b \rfloor_Y \in Y$, d'où $f_0 \in \uparrow(\{a\} \to Y)$. Comme f est minimale dans $\uparrow(\{a\} \to Y)$, on a donc $f_0 = f$, et $b = \lfloor b \rfloor_Y \in Y$. □

La construction $\{a\} \to Y$ peut encore se décomposer de la manière suivante :

Lemme 5.1.18 (Décomposition de $\{a\} \to Y$) — *Pour toute clique $a \in \mathcal{A}$ et pour toute base de type $Y \subset \mathcal{B}$ on a*

$$\{a\} \to Y \quad = \quad \bigcup_{b \in Y} (\{a\} \to \{b\}) \qquad \textit{(union disjointe).}$$

Preuve. Conséquence immédiate du lemme 5.1.17. Il est clair par ailleurs que les types sémantiques $\{a\} \to \{b_1\}$ et $\{a\} \to \{b_2\}$ sont disjoints dès que b_1 et b_2 sont deux cliques distinctes de Y (puisque $b_1 \neq b_2$ entraîne $b_1 \cup b_2 \notin \mathcal{B}$). □

Proposition 5.1.19 (Caractérisation interne de $\{a\} \to \{b\}$) — *Soient $a \in \mathcal{A}$ et $b \in \mathcal{B}$. Pour tout $f \in \mathcal{A} \xrightarrow{q} \mathcal{B}$ on a*

$$f \ \in \{a\} \to \{b\} \qquad \Leftrightarrow \qquad \exists (a_\beta)_{\beta \in b} \in \mathfrak{P}(a)^b \quad f \ = \ \{(a_\beta, \beta); \quad \beta \in b\}.$$

Preuve. La partie directe a été établie par le lemme 5.1.17. Réciproquement, si $(a_\beta)_{\beta \in b}$ est une famille de sous-cliques de a indicée par b, on vérifie aisément que les atomes fonctionnels (a_{β_1}, β_1) et (a_{β_2}, β_2) sont cohérents dans $\mathcal{A} \xrightarrow{q} \mathcal{B}$ pour tous $\beta_1, \beta_2 \in b$. La fonction f définie par $f = \{(a_\beta, \beta); \ \ \beta \in b\}$ est alors clairement une clique de la base de type $\{a\} \to \{b\}$. □

5.1.5 Types μ-finis

Dans ce paragraphe, μ désigne un cardinal régulier infini.

Définition 5.1.20 (Base de type μ-finie) — Soit $\mathcal{A}$ un espace cohérent. Une base de type $X \subset \mathcal{A}$ est dite *μ-finie* si

1. $\overline{\overline{X}} < \mu$;
2. pour tout $a \in X$, $\overline{a} < \mu$.

Remarquons que la notion de μ-finitude est associée aux bases de types et non aux types sémantiques qui leur sont associés. Par conséquent, l'attribut de μ-finitude ne dépend pas de l'espace cohérent dans lequel on considère la base de type, et le type sémantique qui lui est associé (lequel dépend de l'espace) peut être arbitrairement grand. Par ailleurs, l'hypothèse de régularité faite sur μ entraîne que pour toute base de type $X \subset \mathcal{A}$ on a

$$X \ \ \mu\text{-finie} \qquad \Rightarrow \qquad \overline{\overline{|X|}} < \mu.$$

(La réciproque est fausse en général.)

Proposition 5.1.21 (Produit dépendant μ-fini) — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a \in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X. Si X est μ-fini, alors le produit dépendant $\Pi(a \in X; Y_a) \subset [\mathcal{A} \xrightarrow{q} \mathcal{B}]$ ne contient que des (traces de) fonctions μ-stables, c'est-à-dire :*

$$X \ \ \mu\textit{-fini} \qquad \Rightarrow \qquad \Pi(a \in X; Y_a) \quad \subset \quad \mathcal{A} \xrightarrow{\mu} \mathcal{B} \quad \Subset \quad \mathcal{A} \xrightarrow{q} \mathcal{B}.$$

Preuve. Conséquence immédiate du lemme 5.1.13. □

Proposition 5.1.22 (Clôture des univers) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X. Si X et tous les Y_a ($a \in X$) sont des bases de types μ-finies, et si μ est dénombrable ou inaccessible, alors le produit dépendant $\Pi(a\in X; Y_a)$ et l'intersection $\forall(a\in X; Y_a)$ sont des types μ-finis.*

Preuve. Conséquence immédiate du lemme 5.1.13 et de la définition de la notion de cardinal inaccessible (le cas dénombrable étant quant à lui évident). □

5.1.6 Les types extrémaux 0 et 1

Soit $\mathcal{A}$ un espace cohérent. L'ensemble des bases de type de $\mathcal{A}$ admet un plus petit élément et un plus grand élément pour l'ordre du sous-typage, qui sont respectivement

- $\mathbf{0} = \varnothing$, qui représente le type vide puisque $\uparrow\mathbf{0} = \varnothing$;
- $\mathbf{1} = \{\varnothing\}$, qui représente le type de toutes les cliques de $\mathcal{A}$ puisque $\uparrow\mathbf{1} = \mathcal{A}$.

Pour toute base de type $X \subset \mathcal{A}$ on a $\mathbf{0} \leq X \leq \mathbf{1}$.

Remarquons que la définition de $\mathbf{0}$ et de $\mathbf{1}$ ne dépend pas de l'espace cohérent $\mathcal{A}$ considéré. En particulier, $\mathbf{1}$ représente donc le type universel, c'est-à-dire le type de toutes les cliques dans tous les espaces cohérents. Notons également que les types $\mathbf{0}$ et $\mathbf{1}$ sont μ-finis quel que soit le cardinal régulier infini μ, bien que $\mathbf{1}$ représente le type de tous les objets de l'univers.

Une des propriétés les plus remarquables des types $\mathbf{0}$ et $\mathbf{1}$ est qu'ils se comportent exactement comme les booléens $\varnothing$ et $\{\bullet\}$ du modèle classique de CCω, et en ont également le comportement imprédicatif :

Proposition 5.1.23 (Imprédicativité) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X. Si pour tout $a \in X$ on a $Y_a = \mathbf{0}$ ou $Y_a = \mathbf{1}$, alors*

$$\Pi(a\in X; Y_a) \quad = \quad \forall(a\in X; Y_a) \quad = \quad \begin{cases} \mathbf{0} & \text{si } Y_a = \mathbf{0} \text{ pour au moins une clique } a \in X \\ \mathbf{1} & \text{si } Y_a = \mathbf{1} \text{ pour tout } a \in X \end{cases}$$

Preuve. Résulte immédiatement de la définition des bases de types $\mathbf{0}$ et $\mathbf{1}$. □

Dans le modèle que nous allons construire dans les pages qui suivent, nous utiliserons les bases de type booléennes $\mathbf{0}$ et $\mathbf{1}$ pour représenter les types propositionnels.

Il est cependant important de remarquer que dans la proposition ci-dessus, l'égalité qui y est formulée est une égalité au sens de la théorie des ensembles, et non une égalité à isomorphisme près. Rappelons que dans le modèle ensembliste de CCω ce n'est pas le cas, puisque l'interprétation de l'imprédicativité est soumise à l'identification de toutes les fonctions constantes $(x_{\in T} \mapsto \bullet)$ à l'objet $\bullet$ lui-même (voir paragraphe 3.1.1), ce qui nécessite un travail de formalisation supplémentaire pour justifier la pertinence d'une telle identification.

Dans le cadre des espaces cohérents, cette tâche est complètement prise en charge par le codage à l'aide des traces. En effet, toute fonction constante $(a_{\in\mathcal{A}} \mapsto \varnothing)$ retournant systématiquement la clique vide est *égale* à la clique vide, puisque nous ne travaillons ici qu'avec des fonctions μ-stables représentées par leurs traces.

Nous verrons au chapitre 6 que l'adaptation de la notion de réalisabilité au cadre des espaces cohérents fait également disparaître l'équivalence de catégories entre PER et ω-sets modestes que nous avons évoquée au paragraphe 3.1.2, lesquels deviennent une seule et même entité dans la catégorie des $\mathcal{K}$-espaces cohérents (toujours grâce à la notion de trace). Ceci nous permettra donc de travailler systématiquement avec de vraies égalités, et non pas avec des égalités "à isomorphisme près" dont la justification n'est pas toujours très claire.

5.1.7 Types comme citoyens de première classe

Tels que nous les avons présentées jusqu'ici, les bases de types $X \subset \mathcal{A}$ sont de nature différente des cliques $a \in \mathcal{A}$: les premières sont des ensembles de cliques, donc des ensembles d'ensembles d'atomes, tandis que les secondes ne sont que des ensembles d'atomes. Notre objectif étant d'interpréter un calcul dans lequel les types sont des objets de première classe, il est donc nécessaire d'introduire un mécanisme permettant de transformer une base de type en une clique.

Pour cela, nous introduisons pour chaque base de type $X \subset \mathcal{A}$ un nouvel atome $\tau(X)$, qui représente la base de type X comme un citoyen de première classe. Par convention, deux atomes $\tau(X)$ et $\tau(Y)$ sont cohérents si et seulement si $X = Y$.

Définition 5.1.24 (Espace cohérent des types) — Soit $\mathcal{A}$ un espace cohérent. On note $\mathsf{Ty}(\mathcal{A})$ l'espace cohérent plat dont les atomes sont de la forme $\tau(X)$, où X décrit l'ensemble des bases de type de $\mathcal{A}$. Cet espace est caractérisé par :

- $|\mathsf{Ty}(\mathcal{A})| \quad = \quad \big\{\tau(X);\quad X \ \text{ base de type de } \mathcal{A}\big\}$;
- pour toutes bases de types $X, Y \subset \mathcal{A}$: $\qquad \tau(X) \supset \tau(Y) \ (\text{mod } \mathsf{Ty}(\mathcal{A})) \quad \Leftrightarrow \quad X = Y$.

L'espace cohérent $\mathsf{Ty}(\mathcal{A})$ étant plat, ses cliques sont soit la clique vide $\varnothing$, soit les singletons de la forme $\{\tau(X)\}$, qui représentent le type X comme citoyen de première classe.

Remarquons que deux types différents sont incomparables et incompatibles en tant qu'objets de première classe. La relation d'ordre sur $\mathsf{Ty}(\mathcal{A})$ n'a donc rien à voir avec la relation de sous-typage. Cette déconnexion des deux relations d'ordre est ici très importante, puisqu'elle va nous permettre de faire en sorte que les (interprétations des) produits dépendants $\Pi x : T \,.\, U$ et $\forall x : T \,.\, U$ restent monotones lorsque nous les considérerons comme des opérateurs de première classe (c'est-à-dire comme des fonctions μ-stables pour un cardinal μ suffisamment grand).

Rappelons que si $\mathcal{A} \Subset \mathcal{A}'$, alors toute base de type $X \subset \mathcal{A}$ est également une base de type de $\mathcal{A}'$. Par conséquent :

Proposition 5.1.25 (Covariance) — *La correspondance $\mathcal{A} \mapsto \mathsf{Ty}(\mathcal{A})$ est un foncteur covariant dans la catégorie des plongements rigides :*

$$\mathcal{A} \Subset \mathcal{A}' \quad \Rightarrow \quad \mathsf{Ty}(\mathcal{A}) \Subset \mathsf{Ty}(\mathcal{A}').$$

Cependant, le foncteur ci-dessus n'est jamais μ-continu, pour essentiellement la même raison qu'il n'existe pas d'ensemble équipotent à l'ensemble de ses parties. Afin de pouvoir résoudre des équations récursives de domaine impliquant des types comme citoyens de première classe, il est naturel de restreindre l'espace $\mathsf{Ty}(\mathcal{A})$ de la manière suivante :

Définition 5.1.26 (Espace cohérent des types μ-finis) — Soit $\mathcal{A}$ un espace cohérent, et μ un cardinal régulier infini. L'espace cohérent des types μ-finis, noté $\mathsf{Ty}_\mu(\mathcal{A})$, est le sous-espace cohérent de $\mathsf{Ty}(\mathcal{A})$ dont les atomes sont les atomes de la forme $\tau(X) \in |\mathsf{Ty}(\mathcal{A})|$ où X est μ-fini.

D'après la définition, on a $\mathsf{Ty}_\mu(\mathcal{A}) \Subset \mathsf{Ty}(\mathcal{A})$, et plus généralement

$$\mathsf{Ty}_{\mu_0}(\mathcal{A}) \;\Subset\; \mathsf{Ty}_{\mu_2}(\mathcal{A}) \;\Subset\; \ldots\; \mathsf{Ty}_{\mu_x}(\mathcal{A}) \;\Subset\; \mathsf{Ty}_{\mu_{x+1}}(\mathcal{A}) \;\Subset\; \ldots \;\Subset\; \mathsf{Ty}(\mathcal{A})$$

où (μ_x) désigne ici la suite transfinie des cardinaux réguliers infinis.

Proposition 5.1.27 (μ-continuité) — *Pour tout cardinal régulier infini μ, la correspondance $\mathcal{A} \mapsto \mathsf{Ty}_\mu(\mathcal{A})$ définit un bi-foncteur covariant μ-continu dans la catégorie des plongements rigides, c'est-à-dire :*

1. *Si $\mathcal{A} \Subset \mathcal{A}'$, alors $\mathsf{Ty}_\mu(\mathcal{A}) \;\Subset\; \mathsf{Ty}_\mu(\mathcal{A}')$.*
2. *Pour toute famille croissante d'espaces cohérents $(\mathcal{A}_i)_{i\in I}$ indicée par un ensemble I μ-dirigé, on a*
$$\operatorname*{colim}_{i\in I}\big(\mathsf{Ty}_\mu(\mathcal{A}_i)\big) \quad = \quad \mathsf{Ty}_\mu\Big(\operatorname*{colim}_{i\in I}\mathcal{A}_i\Big).$$

Preuve. Le premier point est immédiat. Le second point se montre de manière similaire à la proposition 4.3.18 dont la preuve est essentiellement basée sur la régularité du cardinal μ. □

Au paragraphe 5.1.3, nous avons montré comment il était possible de construire à partir d'une base de type $X \subset \mathcal{A}$ et d'une famille de bases de types $Y_a \subset \mathcal{B}$ indicée par X deux nouvelles bases de types

$$\forall(a{\in}X; Y_a) \;\subset\; \mathcal{B} \qquad \text{et} \qquad \Pi(a{\in}X; Y_a) \;\subset\; \mathcal{A} \xrightarrow{q} \mathcal{B}$$

représentant respectivement l'intersection et le produit dépendant de la famille de types $(Y_a)_{a\in X}$. Par ailleurs, nous avons remarqué (Prop. 5.1.21) que dans le cas où X est une base de type μ-finie, la base de type $\Pi(a{\in}X; Y_a)$ ne contient que des traces de fonctions μ-stables, c'est-à-dire :

$$X \text{ } \mu\text{-fini} \qquad \Rightarrow \qquad \Pi(a{\in}X; Y_a) \;\subset\; \mathcal{A} \xrightarrow{\mu} \mathcal{B}.$$

Maintenant que nous avons défini l'espace $\mathsf{Ty}_\mu(\mathcal{A})$, il est possible d'internaliser les constructions $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$ pour en faire des opérateurs Π et $\forall$ sur les types μ-finis comme citoyens de première classe. La seule difficulté vient de ce que l'intersection (resp. le produit dépendant) d'une famille de bases de types μ-finies n'est en général pas une base de type μ-finie, sauf si μ est dénombrable ou inaccessible (Prop. 5.1.22). Pour cette raison, la définition suivante définit des opérateurs partiels, qui échouent (en retournant la clique vide) lorsque le résultat n'est pas une base de type μ-finie.

Définition 5.1.28 (Opérateurs de produit et d'intersection) — Soient $\mathcal{A}$, $\mathcal{B}$ des espaces cohérents, et μ un cardinal régulier infini. Les opérateurs de produit et d'intersection sont les applications

$$\begin{aligned}
\Pi \;&:\; \mathsf{Ty}_\mu(\mathcal{A}) \times \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})\big) \;\to\; \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu} \mathcal{B}) \\
\forall \;&:\; \mathsf{Ty}_\mu(\mathcal{A}) \times \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})\big) \;\to\; \mathsf{Ty}_\mu(\mathcal{B})
\end{aligned}$$

définies pour tout $t \in \mathsf{Ty}_\mu(\mathcal{A})$ et tout $f \in \mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})$ par

$$\Pi(t,f) \;=\; \big\{\tau\big(\Pi(a{\in}X;Y_a)\big)\big\} \qquad \text{et} \qquad \forall(t,f) \;=\; \big\{\tau\big(\forall(a{\in}X;Y_a)\big)\big\}$$

s'il existe une base de type $X \subset \mathcal{A}$ et une famille $(Y_a)_{a\in X}$ de bases de types de $\mathcal{B}$ indicée par X telles que

1. $t \;=\; \{\tau(X)\}$
2. $f(a) \;=\; \{\tau(Y_a)\}$ pour tout $a \in X$
3. $\Pi(a{\in}X;Y_a)$ (resp. $\forall(a{\in}X;Y_a)$) est une base de type μ-finie de $\mathcal{A} \xrightarrow{\mu} \mathcal{B}$ (resp. $\mathcal{B}$)

et dans tous les autres cas, $\Pi(t,f) \;=\; \forall(t,f) \;=\; \varnothing$.

Insistons sur le fait que cette définition est correcte grâce à la troisième condition, qui impose la μ-finitude du résultat. Sans ce critère, les opérateurs Π et $\forall$ ne renverraient pas leur résultat dans les espaces $\mathsf{Ty}_\mu(\mathcal{B})$ et $\mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu} \mathcal{B})$, mais dans les espaces $\mathsf{Ty}(\mathcal{B})$ et $\mathsf{Ty}(\mathcal{A} \xrightarrow{\mu} \mathcal{B})$ (lesquels ne pourraient pas nous permettre d'effectuer la construction de point fixe que nous décrirons au paragraphe 5.2.2).

Les deux opérateurs Π et $\forall$ sont des fonctions μ-stables :

Proposition 5.1.29 (μ-stabilité) — *Soient $\mathcal{A}$, $\mathcal{B}$ des espaces cohérents, et μ un cardinal régulier infini. Les opérateurs*

$$\begin{aligned}\Pi \;&:\; \mathsf{Ty}_\mu(\mathcal{A}) \times \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})\big) \;\to\; \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu} \mathcal{B})\\ \forall \;&:\; \mathsf{Ty}_\mu(\mathcal{A}) \times \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})\big) \;\to\; \mathsf{Ty}_\mu(\mathcal{B})\end{aligned}$$

définis ci-dessus sont des fonctions binaires μ-stables.

Preuve. La preuve s'effectue de la même manière pour chacun des deux opérateurs Π et $\forall$. Traitons par exemple le cas de l'opérateur Π. La preuve s'effectue en trois temps.

1. (Π *est monotone*) Supposons $t \subset t'$ et $f \subset f'$. On distingue deux cas

   Cas 1. $\Pi(t,f) = \varnothing$. L'inclusion $\Pi(t,f) \subset \Pi(t',f')$ est immédiate.

   Cas 2. $\Pi(t,f) \neq \varnothing$. Dans ce cas, il existe $X \subset \mathcal{A}$ et $Y_a \subset \mathcal{B}$ pour tout $a \in X$ tels que $t = \{\tau(X)\}$, $f(a) = \{\tau(Y_a)\}$ pour tout $a \in X$, $\Pi(t,f) = \{\tau(\Pi(a{\in}X;Y_a))\}$ et $\Pi(a{\in}X;Y_a)$ est μ-finie.

   L'inclusion $t = \{\tau(X)\} \subset t'$ entraîne que $t' = t$ car l'espace $\mathsf{Ty}_\mu(\mathcal{A})$ est plat.

   De même, puisque $f \subset f'$, on a $f(a) = \{\tau(Y_a)\} \subset f(a')$ pour tout $a \in X$, d'où $f(a') = f(a)$, car l'espace $\mathsf{Ty}_\mu(\mathcal{B})$ est plat.

   Par conséquent, on a $\Pi(t',f') = \{\tau(\Pi(a{\in}X;Y_a))\} = \Pi(t,f)$.

2. (Π *est quasi-stable*) Soient $(t_i)_{i\in I}$ une famille non-vide de cliques compatibles dans $\mathsf{Ty}_\mu(\mathcal{A})$, et $(f_i)_{i\in I}$ une famille de cliques compatibles de $\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})$ indicée par le même ensemble I. On distingue deux cas.

   Cas 1. Il existe $i_0 \in I$ tel que $t = \varnothing$. Dans ce cas, on a manifestement

   $$\Pi(\textstyle\bigcap t_i, \bigcap f_i) \;=\; \Pi(\varnothing, \bigcap f_i) \;=\; \varnothing \;=\; \bigcap \Pi(t_i,f_i),$$

   la dernière égalité venant de ce que $\Pi(t_{i_0}, f_{i_0}) = \Pi(\varnothing, f_{i_0}) = \varnothing$.

Cas 2. $t \neq \varnothing$ pour tout $i \in I$. Les cliques $t_i \neq \varnothing$ étant compatibles dans l'espace plat $\mathsf{Ty}_\mu(\mathcal{A})$, la famille $(t_i)_{i\in I}$ est constante et il existe une base de type μ-finie $X \subset \mathcal{A}$ telle que $t_i = t = \{\tau(X)\}$ pour tout $i \in I$. On distingue deux sous-cas.

Cas 2.1. Il existe $i_0 \in I$ et $a_0 \in X$ tels que $f_{i_0}(a_0) = \varnothing$. Comme l'application est quasi-stable, on a $\big(\bigcap f_i\big)(a_0) = \bigcap(f_i(a_0)) = \varnothing$, d'où

$$\Pi(t, \bigcap f_i) \;=\; \varnothing \;=\; \bigcap \Pi(t, f_i)$$

(l'égalité de droite venant de ce que $\Pi(t, f_{i_0}) = \varnothing$).

Cas 2.2. $f_i(a) \neq \varnothing$ pour tout $i \in I$ et pour tout $a \in X$. Soit $a \in X$ fixé. Les cliques $f_i(a)$ $(i \in I)$ sont compatibles, car $f_i(a) \subset \big(\bigcup f_j\big)(a)$ pour tout $i \in \mathcal{A}$. L'espace $\mathsf{Ty}_\mu(\mathcal{B})$ étant plat, la famille de cliques non vides $(f_i(a))_{i\in I}$ est constante, et il existe une base de type μ-finie $Y_a \subset \mathcal{B}$ telle que $f_i(a) = \{\tau(Y_a)\}$ pour tout $i \in I$, et cela quel que soit $a \in X$. De plus, comme l'application est quasi-stable, on a

$$\big(\bigcap f_i\big)(a) \;=\; \bigcap(f_i(a)) \;=\; \{\tau(Y_a)\}$$

pour tout $a \in X$. On distingue alors deux sous-cas.

Cas 2.2.1 La base de type $\Pi(a{\in}X; Y_a) \subset \mathcal{A} \xrightarrow{\mu} \mathcal{B}$ est μ-finie. Dans ce cas, on a $\Pi(t_i, f_i) = \Pi(t, f_i) = \{\tau(\Pi(a{\in}X; Y_a))\}$ pour tout $i \in I$, mais aussi $\Pi(\bigcap t_i, \bigcap f_i) = \Pi(t, \bigcap f_i) = \{\tau(\Pi(a{\in}X; Y_a))\}$, d'où l'égalité

$$\Pi(\bigcap t_i, \bigcap f_i) \;=\; \big\{\tau(\Pi(a{\in}X; Y_a))\big\} \;=\; \bigcap \Pi(t_i, f_i).$$

Cas 2.2.2 La base de type $\Pi(a{\in}X; Y_a) \subset \mathcal{A} \xrightarrow{\mu} \mathcal{B}$ n'est pas μ-finie. Dans ce cas, on a $\Pi(t_i, f_i) = \Pi(t, f_i) = \varnothing$ pour tout $i \in I$, mais aussi $\Pi(\bigcap t_i, \bigcap f_i) = \Pi(t, \bigcap f_i) = \varnothing$, d'où l'égalité $\Pi(\bigcap t_i, \bigcap f_i) = \varnothing = \bigcap \Pi(t_i, f_i)$.

3. (Π *est μ-continue*) Comme l'espace d'arrivée $\mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu} \mathcal{B})$ est plat, il suffit pour établir la μ-continuité de Π de montrer que pour tout $t \in \mathsf{Ty}_\mu(\mathcal{A})$ et pour tout $f \in \mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})$ il existe $f_0 \subset f$ tel que $\overline{\overline{f}} < \mu$ et $\Pi(t, f_0) = \Pi(t, f)$. Il n'y a rien à faire pour la clique t, qui est déjà de cardinal $< \mu$. On distingue les deux cas suivants.

Cas 1. $\Pi(t, f) = \varnothing$. Dans ce cas, il suffit de poser $f_0 = \varnothing \subset f$ pour avoir les égalités $\Pi(t, f_0) = \varnothing = \Pi(t, f)$ (la première égalité résultant de la monotonie de Π).

Cas 2. $\Pi(t, f) \neq \varnothing$. Dans ce cas, il existe $X \subset \mathcal{A}$ et $Y_a \subset \mathcal{B}$ pour tout $a \in X$ tels que $t = \{\tau(X)\}$, $f(a) = \{\tau(Y_a)\}$ pour tout $a \in X$, $\Pi(t, f) = \{\tau(\Pi(a{\in}X; Y_a))\}$ et $\Pi(a{\in}X; Y_a)$ est μ-finie.

Puisque $f(a) = \{\tau(Y_a)\}$ pour tout $a \in X$, il existe pour chaque clique $a \in A$ une clique $c_a \subset a$ telle que $\overline{\overline{c_a}} < \mu$ et $(c_a, \tau(Y_a)) \in f$ (d'après la définition de la trace). Considérons alors la fonction μ-stable f_0 définie par

$$f_0 \;=\; \big\{(c_a, \tau(Y_a)); \quad a \in X\big\} \;\subset\; f.$$

On a alors $\overline{\overline{f_0}} \leq \overline{\overline{X}} < \mu$ (puisque X est μ-fini) et $f_0(a) = \{\tau(Y_a)\}$ pour tout $a \in X$ (d'après la formule de l'application), d'où

$$\Pi(t, f_0) \;=\; \big\{\tau(\Pi(a{\in}X; Y_a))\big\} \;=\; \Pi(t, f). \qquad \square$$

Dans ce qui suit, on considérera les opérateurs Π et $\forall$ sous leur forme curryfiée, c'est-à-dire :

$$\begin{array}{rcl} \Pi & : & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu} \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})\big) \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu} \mathcal{B}) \\ \forall & : & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu} \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B})\big) \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B}) \end{array}$$

5.2 Le modèle classique du calcul implicite restreint

Dans toute cette section, nous supposerons l'axiome de ω-inaccessibilité :

(AI^ω) *Il existe une infinité de cardinaux inaccessibles.*

On notera alors $\mu_1 = \aleph_0$, et pour tout $i \geq 2$, μ_i désignera le $(i-1)$-ème cardinal inaccessible. La suite $(\mu_i)_{i>0}$ ainsi définie est une suite strictement croissante de cardinaux tous inaccessibles sauf le premier.[5] Soit

$$\mu \quad = \quad \Big(\sup_{i>0} \mu_i\Big)^+ \quad = \quad \aleph_{(\sup \mu_i)+1}$$

le successeur de la borne supérieure de tous les cardinaux μ_i. Par définition, μ est le plus petit cardinal régulier supérieur à tous les μ_i. (Notons que la borne supérieure $\sup \mu_i$ n'est pas un cardinal régulier, puisque $\mathrm{cof}(\sup \mu_i) = \omega$.)

Dans la suite de cette section, nous nous proposons de construire un modèle 0-restreint du calcul implicite à partir de la plus petite solution de l'équation

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \;\oplus\; \mathsf{Ty}_\mu(\mathcal{A}).$$

Notons qu'un tel espace cohérent constitue de manière immédiate un λ-modèle sous-extensionnel en vertu de la proposition 4.3.20, puisque l'égalité ci-dessus entraîne manifestement l'existence d'un plongement rigide de $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$ dans $\mathcal{A}$.[6]

5.2.1 Le schéma général de l'interprétation

Intuitivement, l'équation ci-dessus exprime une décomposition de l'espace $\mathcal{A}$ (à l'aide d'une somme directe) en deux sous-espaces $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$ et $\mathsf{Ty}_\mu(\mathcal{A})$, où le premier de ces espaces représente la *partie fonctionnelle* de $\mathcal{A}$ — dans laquelle seront interprétées les fonctions du calcul implicite — et où le second sous-espace représente la *partie typique* de $\mathcal{A}$ — dans laquelle on interprétera les types du calcul implicite. Dans le sous-espace $\mathsf{Ty}_\mu(\mathcal{A})$, seules les cliques définies — de la forme $\{\tau(X)\}$ — serviront à interpréter les types du calcul implicite. Aussi la clique vide — qui est pourtant présente dans chacune des deux composantes — n'aura-t-elle par la suite qu'une interprétation strictement fonctionnelle (elle représentera la fonction indéfinie). En anticipant sur la suite (*cf* paragraphe 5.2.3), l'ensemble des types du modèle sera donné par

$$\mathcal{T} \;=\; \mathsf{Ty}_\mu(\mathcal{A}) \setminus \{\varnothing\}$$

[5] Formellement, le cardinal dénombrable $\mu_1 = \aleph_0$ partage de nombreuses propriétés communes avec les cardinaux inaccessibles, tel que le fait d'être fortement limite (*i.e.* $n < \aleph_0$ entraîne $2^n < \aleph_0$) et régulier.

[6] Nous ferons d'ailleurs en sorte que le plongement rigide $\phi : [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \rightarrowtail \mathcal{A}$ soit une simple inclusion rigide $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \Subset \mathcal{A}$, de manière à éviter de travailler à isomorphisme près.

(en reprenant les notations du paragraphe 3.3.3), et la fonction de décodage El qui lui est associée sera définie par :

$$\mathsf{El}(\{\tau(X)\}) \;=\; \uparrow X$$

Informellement, le décodage opéré par la fonction El consiste à prendre une clique $t \in \mathcal{T}$, à "décapsuler" la base de type X μ-finie contenue dans l'unique atome $\tau(X) \in t$, puis à calculer le type sémantique de X (c'est-à-dire sa clôture supérieure) dans l'espace $\mathcal{A}$. Dans ce cadre, la relation de typage entre deux cliques a et t du modèle sera donnée par

$$\begin{array}{lcl} a \text{ a le type } t & \Leftrightarrow & t \in \mathcal{T} \quad \text{et} \quad a \in \mathsf{El}(t) \\ & \Leftrightarrow & \exists\, X \quad t = \{\tau(X)\} \quad \text{et} \quad a \in \uparrow X. \end{array}$$

Remarquons qu'en général, le type sémantique $\uparrow X$ — qui est la clôture supérieure de X prise dans l'espace $\mathcal{A}$ — est beaucoup plus grand que la base de type X elle-même. En particulier, le type sémantique $\uparrow X$ est susceptible de contenir des cliques dont certains des atomes n'apparaissent pas dans la base de type X. Par exemple, le type sémantique $\mathbf{1} = \{\varnothing\}$ n'a aucun contenu atomique, mais son type sémantique $\uparrow\mathbf{1} = \mathcal{A}$ contient toutes les cliques du modèle. Aussi la clique $\{\tau(\mathbf{1})\}$ représente-t-elle donc (en tant que citoyen de première classe) le type de tous les objets du modèle, et est à ce titre son propre type :

$$\{\tau(\mathbf{1})\} \text{ a le type } \{\tau(\mathbf{1})\} \quad \Leftrightarrow \quad \{\tau(\mathbf{1})\} \;\in\; \mathsf{El}\big(\{\tau(\mathbf{1})\}\big) \quad \Leftrightarrow \quad \{\tau(\mathbf{1})\} \;\in\; \uparrow\mathbf{1} = \mathcal{A}$$

(c'est par ailleurs le seul objet du modèle à avoir cette propriété).

Dans le modèle $\mathcal{A}$, les types imprédicatifs seront naturellement interprétés par les cliques $\{\tau(\mathbf{0})\}$ (le type vide) et $\{\tau(\mathbf{1})\}$ (le type universel). La sorte Prop sera alors interprétée par la clique $\{\tau(X)\}$ où X est la base de type de $\mathsf{Ty}_\mu(\mathcal{A})$ formée par la paire

$$X \quad = \quad \big\{\{\tau(\mathbf{0})\};\; \{\tau(\mathbf{1})\}\big\}.$$

Les univers prédicatifs Type_i s'interpréteront quant à eux d'une manière très analogue à celle que nous avons déjà décrite aux paragraphes 3.1.1 et 3.1.2 (qui repose sur l'hypothèse d'inaccessibilité faite sur les cardinaux μ_i). Les opérateurs de produits Π et $\forall$ nécessaires pour compléter cette construction d'un modèle abstrait du calcul implicite seront simplement donnés par les fonctions Π et $\forall$ que nous avons introduites au paragraphe 5.1.7.

5.2.2 Résolution de $\mathcal{A} = [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \oplus \mathsf{Ty}_\mu(\mathcal{A})$

Avant de commencer la construction de $\mathcal{A}$, nous allons effectuer la convention suivante :

Convention 5.2.1 (Codage de $\tau(X)$) — On supposera que pour tout espace cohérent $\mathcal{A}$ et pour toute base de type $X \subset \mathcal{A}$, l'atome $\tau(X)$ représentant le type X comme un objet de première classe est codé de telle manière que $\tau(X)$ soit distinct de tout couple (x, y), c'est-à-dire :

$$\forall\, \mathcal{A} \qquad \forall\, X \subset \mathcal{A} \qquad \forall\, x, y \qquad\qquad \tau(X) \;\neq\; (x, y).$$

Dans ZFC, le couple (x, y) est codé par l'ensemble $(x, y) = \{\{x\}; \{x; y\}\}$ qui contient au plus deux éléments. Afin de prévenir toute ambiguïté de la forme $(x, y) = \tau(X)$, il suffit de coder l'atome $\tau(X)$ par un ensemble à trois éléments, en posant par exemple $\tau(X) =$

$\{\varnothing;\{\varnothing\};\{\{X\}\}\}$. Ce codage est clairement injectif (puisque $\tau(X)=\tau(Y)$ entraîne manifestement $X=Y$), et satisfait alors la convention selon laquelle

$$\forall\, x,y \qquad \tau(X)\neq(x,y).$$

Techniquement, cette convention est importante car elle nous permet d'utiliser la définition simplifiée de la somme directe $[\mathcal{A}\xrightarrow{\mu}\mathcal{A}]\oplus\mathsf{Ty}_\mu(\mathcal{A})$ (voir paragraphe 4.1.6) en posant pour tout espace cohérent $\mathcal{A}$

$$\Big|[\mathcal{A}\xrightarrow{\mu}\mathcal{A}]\oplus\mathsf{Ty}_\mu(\mathcal{A})\Big| \quad=\quad \mathcal{A}_{(\mu)}\times|\mathcal{A}| \;\cup\; \big|\mathsf{Ty}_\mu(\mathcal{A})\big|$$

(où $\mathcal{A}_{(\mu)}$ désigne l'ensemble des cliques de $\mathcal{A}$ de cardinal $<\mu$), ce qui est justifié ici par le fait que l'union ci-dessus est naturellement disjointe (grâce à la convention), et qu'il n'est par conséquent pas nécessaire d'introduire des *tags* artificiels afin de forcer cette propriété. Dans toute la suite de ce chapitre, nous ne considérerons que la définition simplifiée de la somme directe $[\mathcal{A}\xrightarrow{\mu}\mathcal{A}]\oplus\mathsf{Ty}_\mu(\mathcal{A})$, ce qui nous permettra de ne manipuler que des inclusions rigides et des égalités entre espaces cohérents, plutôt que des plongements rigides ou des égalités "à isomorphisme près".

Afin de construire la plus petite solution de l'équation $\mathcal{A}=[\mathcal{A}\xrightarrow{\mu}\mathcal{A}]\oplus\mathsf{Ty}_\mu(\mathcal{A})$, on considère la suite transfinie $(\mathcal{A}_x)_{x<\mu}$ définie par :

$$\begin{array}{lcll}\mathcal{A}_0 & = & \mathsf{0} & \text{(espace cohérent nul)}\\ \mathcal{A}_{x+1} & = & [\mathcal{A}_x\xrightarrow{\mu}\mathcal{A}_x]\;\oplus\;\mathsf{Ty}_\mu(\mathcal{A}_x) & \\ \mathcal{A}_x & = & \operatorname*{colim}_{y<x}\mathcal{A}_y & \text{(si } x \text{ est un ordinal limite)}\end{array}$$

Pour que cette définition ait un sens, il faut s'assurer à chaque étape de la construction que le dernier espace $\mathcal{A}_x$ introduit satisfait $\mathcal{A}_y\Subset\mathcal{A}_x$ pour tout $y<x$. Cette vérification est essentielle pour donner un sens à la colimite dans le cas où x est un ordinal limite.

Puisque la suite transfinie d'espaces cohérents $(\mathcal{A}_x)_{x<\mu}$ est croissante, on peut alors poser

$$\mathcal{A} \quad=\quad \operatorname*{colim}_{x<\mu}\mathcal{A}_x.$$

Lemme 5.2.2 (Point fixe) — *L'espace $\mathcal{A}$ défini comme la colimite des espaces $\mathcal{A}_x$ $(x<\mu)$ est la plus petite solution de l'équation*

$$\mathcal{A} \quad=\quad [\mathcal{A}\xrightarrow{\mu}\mathcal{A}]\;\oplus\;\mathsf{Ty}_\mu(\mathcal{A}).$$

Preuve. Comme $\mathcal{A}$ est défini à partir d'une colimite μ-dirigée (avec μ régulier), on a d'après les propositions 4.2.9, 4.3.18 et 5.1.27 :

$$\begin{array}{rcl}[\mathcal{A}\xrightarrow{\mu}\mathcal{A}]\;\oplus\;\mathsf{Ty}_\mu(\mathcal{A}) & = & \Big[\Big(\operatorname*{colim}_{x<\mu}\mathcal{A}_x\Big)\xrightarrow{\mu}\Big(\operatorname*{colim}_{x<\mu}\mathcal{A}_x\Big)\Big]\;\oplus\;\mathsf{Ty}_\mu\Big(\operatorname*{colim}_{x<\mu}\mathcal{A}_x\Big)\\ & = & \operatorname*{colim}_{x<\mu}\Big([\mathcal{A}_x\xrightarrow{\mu}\mathcal{A}_x]\;\oplus\;\mathsf{Ty}_\mu(\mathcal{A}_x)\Big)\\ & = & \operatorname*{colim}_{x<\mu}\mathcal{A}_{x+1} \;=\; \mathcal{A}.\end{array}$$

L'espace cohérent $\mathcal{A}$ est par ailleurs la plus petite solution possible : pour tout autre espace cohérent $\mathcal{A}'$ satisfaisant l'égalité $\mathcal{A}' = [\mathcal{A}' \xrightarrow{\mu} \mathcal{A}'] \oplus \mathsf{Ty}_\mu(\mathcal{A}')$, on montre par récurrence transfinie que $\mathcal{A}_x \Subset \mathcal{A}'$ pour tout $x < \mu$, d'où $\mathcal{A} \Subset \mathcal{A}'$ par passage à la colimite. □

Par définition, la trame de $\mathcal{A}$ est la réunion (croissante) des trames des espaces $\mathcal{A}_x$ lorsque $x < \mu$. Il est naturel de structurer cet ensemble d'atomes avec une notion de rang très similaire à celle qui a été introduite au paragraphe 3.4.4.

Définition 5.2.3 (Rang) — Soit $\alpha \in |\mathcal{A}|$ un atome de $\mathcal{A}$. On appelle *rang de* α le plus petit ordinal $x < \mu$ tel que $\alpha \in |\mathcal{A}_x|$. Le rang de α est noté $\mathsf{rk}(\alpha)$. La notation $\mathsf{rk}(\alpha)$ est étendue à toute clique $a \in \mathcal{A}$ et à toute base de type $X \subset \mathcal{A}$ en posant

$$\mathsf{rk}(a) \quad = \quad \sup_{\alpha \in a} \mathsf{rk}(\alpha) \qquad \text{et} \qquad \mathsf{rk}(X) \quad = \quad \sup_{a \in X} \mathsf{rk}(a).$$

Puisque $|\mathcal{A}|$ est la réunion des $|\mathcal{A}_x|$ pour $x < \mu$, on a $\mathsf{rk}(\alpha) < \mu$ pour tout $\alpha \in |\mathcal{A}|$. De plus, le rang d'un atome de $\mathcal{A}$ est toujours un ordinal successeur. Ceci n'est plus vrai si l'on considère une clique ou même une base de type : le rang d'une clique $a \in \mathcal{A}$ ou d'une base de type $X \subset \mathcal{A}$ peut être n'importe quel ordinal inférieur ou même égal à μ. Si l'on note $\mathcal{A}_\mu = \mathcal{A}$, il est facile de vérifier que pour toute clique $a \in \mathcal{A}$ et pour tout type sémantique $X \subset \mathcal{A}$, $\mathsf{rk}(a)$ est le plus petit ordinal x tel que $a \in \mathcal{A}_x$ $(0 \leq x \leq \mu)$ et $\mathsf{rk}(X)$ le plus petit ordinal x tel que $X \subset \mathcal{A}_x$ $(0 \leq x \leq \mu)$.

Lemme 5.2.4 (Structure inductive de $|\mathcal{A}|$) — *Tout atome de* $|\mathcal{A}|$ *est*
- *soit de la forme* (a, β)*, avec* $a \in \mathcal{A}$*,* $\overline{\overline{a}} < \mu$ *et* $\beta \in |\mathcal{A}|$ *;*
- *soit de la forme* $\tau(X)$*, avec* $X \subset \mathcal{A}$ *et* X *fini.*

De plus, on a les égalités suivantes :

$$\mathsf{rk}((a,\beta)) \quad = \quad \max\big(\mathsf{rk}(a), \mathsf{rk}(\beta)\big) + 1 \qquad \textit{et} \qquad \mathsf{rk}\big(\tau(X)\big) \quad = \quad \mathsf{rk}(X) + 1.$$

Preuve. Tout atome de l'espace $\mathcal{A} = [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \oplus \mathsf{Ty}_\mu(\mathcal{A})$ est clairement soit de la forme $(a, \beta) \in \big|\mathcal{A} \xrightarrow{\mu} \mathcal{A}\big|$, soit de la forme $\tau(X) \in \big|\mathsf{Ty}_\mu(\mathcal{A})\big|$ en vertu de l'utilisation de la définition simplifiée de la somme directe (laquelle est rendue possible grâce à la convention 5.2.1). Les deux égalités ci-dessus résultent de ce que par définition, l'espace $\mathcal{A}_x$ n'est rien d'autre que l'ensemble des atomes de $\mathcal{A}$ de rang strictement inférieur à x. □

Insistons sur le fait que l'utilisation de la définition simplifiée de la trame de la somme directe $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \oplus \mathsf{Ty}_\mu(\mathcal{A})$ nous permet ici d'avoir de vraies inclusions rigides $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \Subset \mathcal{A}$ et $\mathsf{Ty}_\mu(\mathcal{A}) \Subset \mathcal{A}$ plutôt que des plongements rigides effectuant des renommages d'atomes.

Lemme 5.2.5 (Cardinalité des espaces $(\mathcal{A}_x)$) — *La suite transfinie* $(\mathcal{A}_x)_{x<\mu}$ *a les propriétés suivantes :*

1. *pour tout* $1 \leq i < \omega$ *et pour tout* $x < \mu_i$*, on a* $\overline{\overline{|\mathcal{A}_x|}} < \mu_i$*.*
2. *pour tout* $1 \leq i < \omega$*, on a* $\overline{\overline{|\mathcal{A}_{\mu_i}|}} \leq \mu_i$*.*

Preuve. Le premier point (l'entier $i \geq 1$ étant fixé) se prouve à l'aide d'une récurrence transfinie jusqu'à μ, en utilisant les propriétés de dénombrabilité (pour $i = 1$) et d'inaccessibilité (pour $i \geq 2$) des cardinaux μ_i. Le second point résulte du premier point par passage à la limite (notons l'inégalité au sens large). □

Remarque 5.2.6 — Il est facile de prouver que l'inégalité au sens large du second point est en réalité une égalité. Il suffit pour cela d'injecter (pour tout $1 \leq i < \omega$) l'ensemble V_{μ_i} — dont le cardinal est μ_i — dans la partie typique de la trame $|\mathcal{A}_{\mu_i}|$. Cette injection $\Phi_i : V_{\mu_i} \to |\mathcal{A}_{\mu_i}|$ est définie par récurrence sur le rang de l'antécédent (au sens la hiérarchie cumulative (V_x)) en posant

$$\Phi_i(x) \quad = \quad \tau\big(\{\{\Phi_i(y)\};\quad y \in x\}\big).$$

On constate facilement que cette application est bien définie et envoie tout ensemble $x \in V_{\mu_i}$ sur un atome de la partie typique de $\mathcal{A}_{\mu_i}$, et cela de manière injective. Comme pour tout $1 \leq i < \omega$, V_{μ_i} est un modèle de la théorie des ensembles (sans axiome de l'infini dans le cas où $i = 1$), l'application $\Phi_i : V_{\mu_i} \to |\mathcal{A}_{\mu_i}|$ n'est rien d'autre qu'un plongement de la théorie des ensembles dans le modèle de la théorie des types que nous sommes en train de construire. Par ailleurs, on vérifie aisément que la relation d'appartenance au sein de V_{μ_i} se traduit en la relation de typage dans $\mathcal{A}_{\mu_i}$, et que l'intersection au sens de la théorie des ensembles devient un cas particulier de l'intersection de bases de types telle que nous l'avons définie sur les types comme objets de première classe au paragraphe 5.1.7 à l'aide de l'opérateur $\forall$.

Lemme 5.2.7 — *Pour tout $1 \leq i < \omega$ et pour tout $x < \mu_i$, on a :*

1. $\mathsf{Ty}_\mu(\mathcal{A}_x) \;=\; \mathsf{Ty}(\mathcal{A}_x)$ *(espace cohérent de toutes les bases de types)*
2. $(\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x) \;=\; (\mathcal{A}_x \xrightarrow{q} \mathcal{A}_x)$ *(espace cohérent de toutes les fonctions quasi-stables)*
3. $\mathcal{A}_{x+1} \;=\; [\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x] \oplus \mathsf{Ty}_\mu(\mathcal{A}_x) \;=\; [\mathcal{A}_x \xrightarrow{q} \mathcal{A}_x] \oplus \mathsf{Ty}(\mathcal{A}_x)$.

Preuve. Supposons que $1 \leq i < \omega$ et $x < \mu_i$. D'après le lemme précédent, on a $\overline{\overline{|\mathcal{A}_x|}} < \mu_i < \mu$, ce qui prouve que toute clique $a \in \mathcal{A}_x$ est de cardinal $\overline{\overline{a}} < \mu$, d'où il résulte immédiatement que toute fonction quasi-stable sur $\mathcal{A}_x$ est également μ-stable. Comme par ailleurs μ_i est dénombrable ou inaccessible, on a

$$\overline{\overline{\mathcal{A}_x}} \;\leq\; 2^{\overline{\overline{|\mathcal{A}_x|}}} \;<\; \mu_i \;<\; \mu,$$

d'où il ressort que toute base de type $X \subset \mathcal{A}$ est μ-finie, d'où $\mathsf{Ty}_\mu(\mathcal{A}_x) = \mathsf{Ty}(\mathcal{A}_x)$. Le troisième point est conséquence immédiate des deux premiers points. □

Intuitivement, le lemme ci-dessus exprime que tous les espaces cohérents $\mathcal{A}_x$ d'indice x strictement inférieur à $\sup \mu_i$ sont "beaucoup plus petits" que le cardinal de continuité μ lui-même, ce qui rend superflus dans les espaces $\mathcal{A}_x$ les critères de μ-stabilité et de μ-finitude, qui sont immédiatement satisfaits par n'importe quelle fonction quasi-stable $F : \mathcal{A}_x \xrightarrow{q} \mathcal{A}_x$ et n'importe quel type sémantique $X \subset \mathcal{A}_x$. Notons que ces égalités ne valent que pour $0 \leq x < \sup \mu_i$ (le fait que x soit majoré par au moins un des cardinaux μ_i est ici essentiel). Il est facile de vérifier que pour tout ordinal x compris entre $\sup \mu_i$ et $\mu = (\sup \mu_i)^+$, on a $[\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x] \neq [\mathcal{A}_x \xrightarrow{q} \mathcal{A}_x]$ et $\mathsf{Ty}_\mu(\mathcal{A}_x) \neq \mathsf{Ty}(\mathcal{A}_x)$.

5.2.3 Structure de modèle abstrait du calcul implicite

Comme $\mathcal{A}$ satisfait par définition l'inclusion rigide $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \Subset \mathcal{A}$, $\mathcal{A}$ est muni naturellement d'une structure de λ-modèle sous-extensionnel $(\mathcal{A}, \cdot, [\![]\!], \subset)$ en vertu de la proposition 4.3.20.

Rappelons que dans ce cadre (défini au paragraphe 4.3.3), l'application $f \cdot a$ de deux cliques $f \in \mathcal{A}$ et $a \in \mathcal{A}$ est définie par

$$\begin{aligned} f \cdot a &= \phi^-(f)(a) \\ &= \big(f \cap |\mathcal{A} \xrightarrow{\mu} \mathcal{A}|\big)(a) \\ &= \big\{\beta \in |\mathcal{A}|\,; \quad \exists\, a_0 \subset a \quad (a_0, \beta) \in f\big\}, \end{aligned}$$

où $\phi^- : \mathcal{A} \multimap [\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$ désigne la rétraction de $\mathcal{A}$ dans $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$ qui extrait de chaque clique $a \in \mathcal{A}$ sa partie fonctionnelle $\phi^-(a) \in [\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$. Remarquons également que pour tout $t \in \mathsf{Ty}_\mu(\mathcal{A}) \Subset \mathcal{A}$ et pour tout $a \in \mathcal{A}$, on a $t \cdot a = \varnothing$. (En tant que fonction, un type comme objet de première classe se comporte comme la fonction indéfinie).

Pour conférer à $\mathcal{A}$ une structure complète de modèle du calcul implicite, il nous faut d'abord définir le sous-ensemble $\mathcal{T} \subset \mathcal{A}$ contenant les objets représentant les types dans le modèle ainsi que la fonction de décodage $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{A})$. Pour cela on pose :

- $\mathcal{T} \;=\; \mathsf{Ty}_\mu(\mathcal{A}) \setminus \{\varnothing\} \;=\; \big\{\{\tau(X)\};\quad X \subset \mathcal{A} \text{ base de type } \mu\text{-finie}\big\}$
- $\mathsf{El}(\{\tau(X)\}) \;=\; \uparrow X$ pour tout $\{\tau(X)\} \in \mathcal{T}$.

On vérifie alors immédiatement que :

Lemme 5.2.8 (Validité des axiomes 1–3) — *Le sous-ensemble $\mathcal{T} \subset \mathcal{A}$ et la fonction de décodage $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{A})$ satisfont les axiomes 1, 2 et 3 des modèles abstraits du calcul implicite.*

Preuve. Les axiomes 1 et 2 sont immédiats par définition de $\mathcal{A}$ et d'après le fait que les types sémantiques de $\mathcal{A}$ sont clos supérieurement dans $\mathcal{A}$. L'axiome 3 est ici trivial, car les éléments de $\mathcal{T}$ sont des cliques-singleton deux à deux incompatibles et incomparables, d'où il ressort que si $t = \{\tau(X)\}$ et $t' = \{\tau(X')\}$, l'inclusion $t \subset t'$ entraîne manifestement $X = X'$ et $\mathsf{El}(t) = \uparrow X = \uparrow X' = \mathsf{El}(t')$. □

Au paragraphe 5.1.7, nous avons défini les opérateurs de produit dépendant (Π) et d'intersection ($\forall$) sur les types comme citoyens de première classe. Sous leur forme curryfiée, ces deux fonctions μ-stables

$$\begin{array}{lcccccccc} \Pi & : & \mathsf{Ty}_\mu(\mathcal{A}) & \xrightarrow{\mu} & \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{A})\big) & \xrightarrow{\mu} & \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu} \mathcal{A}) & \Subset & \mathcal{A} \\ \forall & : & \mathsf{Ty}_\mu(\mathcal{A}) & \xrightarrow{\mu} & \big(\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{A})\big) & \xrightarrow{\mu} & \mathsf{Ty}_\mu(\mathcal{A}) & \Subset & \mathcal{A} \end{array}$$

sont également des cliques de $\mathcal{A}$ d'après les inclusions rigides ci-dessus. Un rapide examen de la définition de Π et $\forall$ nous montre que :

Lemme 5.2.9 (Validité des axiomes 4 et 5) — *Les constantes $\Pi, \forall \in \mathcal{A}$ définies au paragraphe 5.1.7 satisfont les axiomes 4 et 5 des modèles abstraits du calcul implicite.*

Pour terminer cette construction de modèle, il ne nous reste plus qu'à définir les constantes $(\mathsf{u}_i)_{i\in\omega}$ destinées à interpréter Prop et Set (pour $i = 0$) ainsi que la hiérarchie d'univers prédicatifs Type_i (pour $i \geq 1$). On pose alors

- $\mathsf{U}_0 \;=\; \big\{\{\tau(\mathbf{0})\};\; \{\tau(\mathbf{1})\}\big\}$
- $\mathsf{U}_i \;=\; \big\{t = \{\tau(X)\} \in \mathcal{T};\quad \mathsf{rk}(X) < \mu_i\big\} \;=\; \bigcup_{x<\mu_i}(\mathsf{Ty}(A_x) \setminus \{\varnothing\}) \qquad (i \geq 1)$

et on vérifie immédiatement que les sous-ensembles $\mathsf{U}_i \subset \mathcal{T}$ sont des bases de types de $\mathcal{A}$ μ-finies, car

$$\overline{\overline{\mathsf{U}_0}} = 2 \qquad \text{et} \qquad \overline{\overline{\mathsf{U}_i}} \leq \mu_i \qquad (i \geq 1)$$

d'après le lemme 5.2.5. Notons alors $(\mathsf{u}_i)_{i\in\omega}$ la suite d'éléments de $\mathcal{T}$ définie par $\mathsf{u}_i = \{\tau(\mathsf{U}_i)\}$ pour tout $i \in \omega$. On vérifie alors que :

Lemme 5.2.10 (Validité des axiomes 6–10) — *Les constantes* u_i *(*$i \in \omega$*) satisfont les axiomes 6 à 10 des modèles abstraits du calcul implicite.*

Preuve. L'axiome 6 vient de ce que $\mathsf{El}(\mathsf{u}_i) = \uparrow\mathsf{U}_i = \mathsf{U}_i \subset \mathcal{T}$, et les axiomes 7 et 8 découlent de la définition des constantes u_i. L'axiome 9 est une conséquence de la proposition 5.1.22, et l'axiome 10 résulte de la proposition 5.1.23. □

En collectant les lemmes 5.2.8, 5.2.9 et 5.2.10 on a donc la proposition :

Proposition 5.2.11 — *La structure* $\big(\mathcal{A}, \cdot, [\![]\!], \subset, \mathsf{El}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega}\big)$ *est un modèle abstrait du calcul implicite.*

Par ailleurs, il est clair que ce modèle abstrait est un modèle 0-restreint au sens de la définition 3.3.18. En effet, si l'on pose $t_0 = \{\tau(\mathbf{0})\}$, on vérifie immédiatement que

$$t_0 \;\in\; \mathsf{El}(\mathsf{u}_0) \;=\; \uparrow\mathsf{U}_0 \;=\; \mathsf{U}_0 \;=\; \big\{\{\tau(\mathbf{0})\};\; \{\tau(\mathbf{1})\}\big\} \qquad \text{et} \qquad \mathsf{El}(t_0) \;=\; \uparrow\mathbf{0} \;=\; \varnothing.$$

D'après la proposition 3.3.17, la fonction d'interprétation $[\![]\!] : \Lambda_{\mathrm{CCI}} \times \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$ définie au paragraphe 3.3.4 valide toutes les règles du calcul implicite restreint, c'est-à-dire toutes les règles du Calcul des Constructions implicite sauf la règle de renforcement (Str), qui est ici invalidée d'après la proposition 3.3.19 puisque $\mathcal{A}$ est un modèle 0-restreint. En revanche, le corollaire 3.3.20 nous permet cependant de conclure que

Théorème 5.2.12 (Cohérence) — *Dans* $\mathrm{ZFC} + \mathrm{AI}^{\omega}$*, le Calcul des Constructions implicite restreint est cohérent.*

5.2.4 Un modèle classique

Nous avons vu au paragraphe 5.1.6 que les bases de types $\mathbf{0}$ et $\mathbf{1}$ se comportent de manière très similaire aux ensembles $\varnothing$ et $\{\bullet\}$ du modèle classique de CCω décrit au paragraphe 3.1.1. Il n'est donc pas étonnant que dans le modèle $\mathcal{A}$ que nous venons de construire, les codages imprédicatifs des connecteurs de la logique intuitionniste du calcul implicite se traduisent par les tables de vérité de la logique classique. En effet, si False, Not, And, Or et Imp désignent les termes définis par

$$\begin{array}{lcl}
\mathsf{False} & \equiv & \forall A : \mathsf{Prop}\,.\,A \\
\mathsf{Not} & \equiv & \lambda A\,.\,A \to \mathsf{False} \\
\mathsf{And} & \equiv & \lambda A, B\,.\,\forall X : \mathsf{Prop}\,.\,(A \to B \to X) \to X \\
\mathsf{Or} & \equiv & \lambda A, B\,.\,\forall X : \mathsf{Prop}\,.\,(A \to X) \to (B \to X) \to X \\
\mathsf{Imp} & \equiv & \lambda A, B\,.\,A \to B
\end{array}$$

il est facile de vérifier qu'on a $[\![\mathsf{False}]\!] = \{\tau(\mathbf{0})\}$, mais aussi

$$\begin{array}{rcl@{\qquad}rcl}
[\![\mathsf{Not}]\!]\big(\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{1})\} & [\![\mathsf{And}]\!]\big(\{\tau(\mathbf{0})\},\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{0})\} \\
[\![\mathsf{Not}]\!]\big(\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{0})\} & [\![\mathsf{And}]\!]\big(\{\tau(\mathbf{0})\},\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{0})\} \\
 & & & [\![\mathsf{And}]\!]\big(\{\tau(\mathbf{1})\},\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{0})\} \\
 & & & [\![\mathsf{And}]\!]\big(\{\tau(\mathbf{1})\},\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{1})\}
\end{array}$$

$$\begin{array}{rcl@{\qquad}rcl}
[\![\mathsf{Or}]\!]\big(\{\tau(\mathbf{0})\},\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{0})\} & [\![\mathsf{Imp}]\!]\big(\{\tau(\mathbf{0})\},\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{1})\} \\
[\![\mathsf{Or}]\!]\big(\{\tau(\mathbf{0})\},\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{1})\} & [\![\mathsf{Imp}]\!]\big(\{\tau(\mathbf{0})\},\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{1})\} \\
[\![\mathsf{Or}]\!]\big(\{\tau(\mathbf{1})\},\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{1})\} & [\![\mathsf{Imp}]\!]\big(\{\tau(\mathbf{1})\},\{\tau(\mathbf{0})\}\big) & = & \{\tau(\mathbf{0})\} \\
[\![\mathsf{Or}]\!]\big(\{\tau(\mathbf{1})\},\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{1})\} & [\![\mathsf{Imp}]\!]\big(\{\tau(\mathbf{1})\},\{\tau(\mathbf{1})\}\big) & = & \{\tau(\mathbf{1})\}
\end{array}$$

(Nous n'avons pas fait figurer les traces complètes des dénotations des termes Not, And, Or et Imp dont le domaine excède le simple cadre des propositions $\{\tau(\mathbf{0})\}$ et $\{\tau(\mathbf{1})\}$.) Un rapide calcul montre alors que le tiers-exclu est valide dans le modèle, puisque

$$[\![\forall A : \mathsf{Prop}\,.\,\mathsf{Or}\ A\ (\mathsf{Not}\ A)]\!] \quad = \quad \{\tau(\mathbf{1})\}.$$

De plus, nous verrons dans le paragraphe suivant que le modèle $\mathcal{A}$ vérifie la propriété d'*indiscernabilité des preuves*, puisqu'il valide l'axiome

$$\forall A : \mathsf{Prop}\,.\,\forall x, y : A\,.\,x =_A y,$$

mais d'une manière beaucoup moins triviale que dans le cadre de l'interprétation ensembliste booléenne de CCω.

5.2.5 Égalité dans le modèle

Dans ce paragraphe, nous nous proposons d'étudier l'interprétation de l'égalité de Leibniz dans un type T quelconque du calcul implicite restreint, laquelle est définie par

$$M_1 =_T M_2 \qquad \equiv \qquad \forall P : T \to \mathsf{Prop}\,.\,P\ M_1 \to P\ M_2,$$

et de montrer que par la fonction d'interprétation $[\![]\!] : \Lambda_{\mathrm{CCI}} \times \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$, cette relation d'égalité se traduit en la relation d'équivalence $a_1 =_X a_2$ introduite au paragraphe 5.1.1, où a_1 et a_2 sont les dénotations respectives de M_1 et de M_2, et où X désigne la base de type encapsulée dans la dénotation de T, c'est-à-dire telle que $[\![T]\!] = \{\tau(X)\}$.

Cette propriété repose sur le fait que dans le modèle, la dénotation d'un prédicat sur X (qui est une clique $f \in \uparrow(X \to \mathsf{U}_0)$, où $\mathsf{U}_0 = \{\{\tau(\mathbf{0})\};\{\tau(\mathbf{1})\}\}$ est la base de type interprétant la sorte Prop) est une fonction nécessairement constante sur chacune des classes d'équivalence de la relation $a_1 =_X a_2$:

Lemme 5.2.13 — *Soit $X \subset \mathcal{A}$ une base de type, et U_0 la base de type de $\mathcal{A}$ définie par $\mathsf{U}_0 = \{\{\tau(\mathbf{0})\};\{\tau(\mathbf{1})\}\}$. Pour toute clique $f \in \uparrow(X \to \mathsf{U}_0)$, la fonction f est constante sur chaque classe d'équivalence partielle de la relation $a_1 =_X a_2$, c'est-à-dire :*

$$f \ \in \ \uparrow(X \to \mathsf{U}_0) \qquad \Rightarrow \qquad \big(\forall\, a_1, a_2 \in \uparrow X \quad a_1 =_X a_2 \quad \Rightarrow \quad f(a_1) = f(a_2)\big).$$

Preuve. Ceci est une conséquence immédiate du fait que les cliques représentant les types comme objets de première classe dans $\mathcal{A}$ — lesquelles sont de la forme $\{\tau(X)\}$ — sont maximales dans $\mathcal{A}$. Par conséquent, si $f \in \uparrow(X \to \mathsf{U}_0)$, on a $f(a) = f(\lfloor a \rfloor_X)$ pour tout $a \in \uparrow X$ d'après un argument immédiat de monotonie, et la relation $a_1 =_X a_2$ entraîne alors que $f(a_1) = f(\lfloor a_1 \rfloor_X) = f(\lfloor a_2 \rfloor_X) = f(a_2)$ puisque $\lfloor a_1 \rfloor_X = \lfloor a_2 \rfloor_X$. □

Proposition 5.2.14 (Égalité dans le modèle) — *Soient Γ un contexte, T, M_1 et M_2 des termes tels que les jugements $\Gamma \vdash_r M_1 : T$ et $\Gamma \vdash_r M_2 : T$ soient dérivables dans le calcul implicite restreint. Pour toute valuation $\rho \in [\![\Gamma]\!]$ on a l'équivalence*

$$[\![M_1 =_T M_2]\!]_\rho = \{\tau(\mathbf{1})\} \qquad \Leftrightarrow \qquad [\![M_1]\!]_\rho =_X [\![M_2]\!]_\rho,$$

où X désigne la base de type de $\mathcal{A}$ telle que $[\![T]\!]_\rho = \{\tau(X)\}$.

Preuve. Soient $a_1 = [\![M_1]\!]_\rho$, $a_2 = [\![M_2]\!]_\rho$ et $X \subset \mathcal{A}$ la base de type telle que $[\![T]\!]_\rho = \{\tau(X)\}$. D'après la proposition 5.1.23, on a les équivalences suivantes :

$$\begin{aligned}
& [\![M_1 =_T M_2]\!]_\rho \;=\; \{\tau(\mathbf{1})\} \\
\Leftrightarrow \quad & [\![\forall P : T \to \mathsf{Prop}\,.\, P\ M_1 \to P\ M_2]\!]_\rho \;=\; \{\tau(\mathbf{1})\} \\
\Leftrightarrow \quad & \forall f \in \uparrow(X \to \mathsf{U}_0) \qquad [\![P\ M_1 \to P\ M_2]\!]_{(\rho; P \leftarrow f)} \;=\; \{\tau(\mathbf{1})\} \\
\Leftrightarrow \quad & \forall f \in \uparrow(X \to \mathsf{U}_0) \qquad f(a_1) = \{\tau(\mathbf{1})\} \quad \Rightarrow \quad f(a_2) = \{\tau(\mathbf{1})\}
\end{aligned}$$

On distingue alors les deux cas suivants :

- Soit $a_1 =_X a_2$, auquel cas $f(a_1) = f(a_2)$ pour tout $f \in \uparrow(X \to \mathsf{U}_0)$ (en vertu du lemme précédent), ce qui montre que le membre droit de l'équivalence ci-dessus est vrai.
- Soit $a_1 \neq_X a_2$, auquel cas la fonction f définie par

$$f(a) \quad = \quad \begin{cases} \{\tau(\mathbf{1})\} & \text{si } a \in \uparrow X \text{ et } a =_X a_1 \\ \{\tau(\mathbf{0})\} & \text{si } a \in \uparrow X \text{ et } a \neq_X a_1 \\ \varnothing & \text{si } a \notin \uparrow X \end{cases}$$

  est clairement μ-stable, et vérifie $f \in \uparrow(X \to \mathsf{U}_0)$, $f(a_1) = \{\tau(\mathbf{1})\}$ et $f(a_2) = \{\tau(\mathbf{0})\}$, ce qui montre que le membre droit de l'équivalence ci-dessus est faux.

Dans tous les cas, on a donc $[\![M_1 =_T M_2]\!]_\rho = \{\tau(\mathbf{1})\}$ si et seulement si $a_1 =_X a_2$. □

À ce stade, il est important de remarquer que, contrairement au modèle ensembliste booléen de CCω présenté au chapitre 3, l'égalité de Leibniz définie sur les termes d'un type T donné ne se traduit pas par l'égalité des dénotations correspondantes, mais par une relation d'équivalence $a_1 =_X a_2$ qui dépend profondément de la base de type $X \subset \mathcal{A}$ interprétant le type T.

Indiscernabilité des preuves Dans le modèle $\mathcal{A}$, les propositions sont interprétées par les cliques $\{\tau(\mathbf{0})\}$ et $\{\tau(\mathbf{1})\}$ formées par encapsulation des bases de types $\mathbf{0} = \varnothing$ et $\mathbf{1} = \{\varnothing\}$. Sur les types sémantiques $\uparrow\mathbf{0} = \varnothing$ et $\uparrow\mathbf{1} = \mathcal{A}$, la relation d'équivalence $a_1 =_X a_2$ est triviale puisqu'elle est vide dans le cas où $X = \mathbf{0}$ et qu'elle n'a qu'une seule classe d'équivalence (formée par l'espace $\mathcal{A}$ entier) dans le cas où $X = \mathbf{1}$. On a donc immédiatement

$$X = \mathbf{0} \;\text{ ou }\; X = \mathbf{1} \qquad \Rightarrow \qquad \big(\forall\, a_1, a_2 \in \uparrow X \quad a_1 =_X a_2\big).$$

La conséquence de cette propriété est que dans le modèle, toutes les dénotations de termes de preuves sont indiscernables *en tant que preuves*. En particulier, l'axiome d'*indiscernabilité des preuves*

$$\mathsf{proof\text{-}irrelevance} \;:\; \forall A : \mathsf{Prop} \,.\, \forall x, y : A \,.\, x =_A y$$

est valide dans le modèle, c'est-à-dire :

$$[\![\forall A : \mathsf{Prop} \,.\, \forall x, y : A \,.\, x =_A y]\!] \quad = \quad \{\tau(\mathbf{1})\}.$$

Égalité et sous-typage Ainsi que nous l'avons déjà évoqué, l'égalité de deux termes M_1 et M_2 au sein d'un type T (au sens de l'égalité de Leibniz définie par codage imprédicatif) ne signifie pas que leurs dénotations $[\![M_1]\!]$ et $[\![M_2]\!]$ sont égales. En fait, il est facile de donner un exemple plus intéressant, dans lequel l'égalité de deux mêmes termes est valide dans un type T, mais invalide dans un sous-type $T' \leq T$.

Pour cela, considérons les types

$$\begin{array}{rcll} \mathsf{bool}_0 & : & \mathsf{Prop} & := \quad \forall X : \mathsf{Prop} \,.\, X \to X \to X \\ \mathsf{bool}_1 & : & \mathsf{Type}_2 & := \quad \forall X : \mathsf{Type}_1 \,.\, X \to X \to X, \end{array}$$

ainsi que les termes

$$\mathsf{true} := \lambda xy \,.\, x \qquad \text{et} \qquad \mathsf{false} := \lambda xy \,.\, y$$

qui ont tous les deux à la fois le type bool_0 et le type bool_1. (Notons que dans le calcul implicite restreint, bool_1 est un sous-type de bool_0.) Dans le modèle $\mathcal{A}$, on a

$$\begin{array}{rcl} [\![\mathsf{true}]\!] & = & \big\{(\{\alpha\}, (\varnothing, \alpha)); \quad \alpha \in |\mathcal{A}|\big\} \\ [\![\mathsf{false}]\!] & = & \big\{(\varnothing, (\{\beta\}, \beta)); \quad \beta \in |\mathcal{A}|\big\} \\ [\![\mathsf{bool}_0]\!] & = & \{\tau(\mathbf{1})\} \\ [\![\mathsf{bool}_1]\!] & = & \big\{\tau(\{f_1; f_2\})\big\}, \end{array}$$

où $f_1, f_2 \in \mathcal{A}$ sont les cliques (μ_2-finies) données par

$$\begin{array}{rcl} f_1 & = & \big\{(\{\alpha\}, (\varnothing, \alpha)); \quad \alpha \in |\mathcal{A}| \;\text{ et }\; \mathsf{rk}(\alpha) < \mu_1\big\} \\ f_2 & = & \big\{(\varnothing, (\{\beta\}, \beta)); \quad \beta \in |\mathcal{A}| \;\text{ et }\; \mathsf{rk}(\beta) < \mu_1\big\}. \end{array}$$

On vérifie alors aisément que la proposition $\mathsf{true} =_{\mathsf{bool}_0} \mathsf{false}$ (égalité dans bool_0) est valide dans $\mathcal{A}$ puisque

$$\lfloor [\![\mathsf{true}]\!] \rfloor_{\mathbf{1}} \;=\; \varnothing \;=\; \lfloor [\![\mathsf{false}]\!] \rfloor_{\mathbf{1}},$$

mais que la proposition $\mathsf{true} =_{\mathsf{bool}_1} \mathsf{false}$ (égalité dans bool_1) n'est pas valide dans $\mathcal{A}$ puisque

$$\lfloor [\![\mathsf{true}]\!] \rfloor_{\{f_1;f_2\}} \;=\; f_1 \;\neq\; f_2 \;=\; \lfloor [\![\mathsf{false}]\!] \rfloor_{\{f_1;f_2\}}.$$

Remarquons par ailleurs que dans le second cas, la *négation* de l'égalité $\mathsf{true} =_{\mathsf{bool}_1} \mathsf{false}$ est prouvable dans le Calcul des Constructions implicite restreint, par le terme

$$\lambda f \,.\, f \;(\lambda x \,.\, x) \quad : \quad (\mathsf{true} =_{\mathsf{bool}_1} \mathsf{false}) \to \mathsf{False}$$

(où False désigne la proposition $\forall A : \mathsf{Prop} \,.\, A$).[7]

[7] On le vérifie simplement en instanciant l'hypothèse $f : \forall P : \mathsf{bool}_1 \to \mathsf{Prop} \,.\, P\ \mathsf{true} \to P\ \mathsf{false}$ par le prédicat $P := \lambda b \,.\, b\ \mathsf{True}\ \mathsf{False}$, où True désigne la proposition $\mathsf{False} \to \mathsf{False}$. Ceci n'est possible que parce que le prédicat $\lambda b \,.\, b\ \mathsf{True}\ \mathsf{False}$ a le type $(\mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}) \to \mathsf{Prop}$ qui est un sous-type de $\mathsf{bool}_1 \to \mathsf{Prop}$ (mais pas un sous-type de $\mathsf{bool}_0 \to \mathsf{Prop}$).

Invalidité de l'extensionnalité fonctionnelle Dans tout ce qui précède, nous avons utilisé l'ordre stable sur les fonctions $F : \mathcal{A} \xrightarrow{\mu} \mathcal{A}$ (qui correspond à l'inclusion de traces) plutôt que l'ordre extensionnel, défini par

$$F \leq_{\mathbf{ext}} G \qquad \equiv \qquad \forall\, a \in \mathcal{A} \quad F(a) \subset G(a).$$

Une conséquence de ce choix est que dans le modèle, les axiomes d'extensionnalité

$$\forall f, g : T \to U\,.\ \ (\forall x : T\,.\ \ f\ x =_U g\ x) \ \ \to \ \ f =_{T\to U} g$$

(où T et U sont deux types quelconques) ne sont pas toujours valides. Pour cela, nous allons donner un contre-exemple en construisant deux termes (clos) F et G de type $T \to U$ et montrer que la proposition

$$\forall x : T\,.\ \ F\ x =_U G\ x$$

est valide dans $\mathcal{A}$, mais que l'égalité $F =_{T\to U} G$ est invalide.

Remarquons tout d'abord que dans le modèle $\mathcal{A}$, l'axiome

$$\forall A : \mathsf{Prop}\,.\ \ (A =_{\mathsf{Prop}} \mathsf{False}) \ \vee \ (A =_{\mathsf{Prop}} \mathsf{True})$$

(ou $\vee$ désigne le codage imprédicatif de la disjonction) est valide, puisque toutes les propositions sont interprétées par les cliques $\{\tau(\mathbf{0})\} = [\![\mathsf{False}]\!]$ et $\{\tau(\mathbf{1})\} = [\![\mathsf{True}]\!]$. Posons alors $T = U = \mathsf{Prop}$, et

$$\begin{array}{rcl} F & : & \mathsf{Prop} \to \mathsf{Prop} \ \ := \ \ \lambda A\,.\,\mathsf{True} \\ G & : & \mathsf{Prop} \to \mathsf{Prop} \ \ := \ \ \lambda A\,.\,A \to A \end{array}$$

D'après la remarque précédente, la proposition $\forall x : \mathsf{Prop}\,.\,F\ x =_{\mathsf{Prop}} G\ x$ est valide dans $\mathcal{A}$ puisque

$$[\![F\ \mathsf{True}]\!] \ = \ [\![G\ \mathsf{True}]\!] \ = \ [\![F\ \mathsf{False}]\!] \ = \ [\![G\ \mathsf{False}]\!] \ = \ \{\tau(\mathbf{1})\}.$$

En revanche, il est facile de vérifier que $[\![F =_{\mathsf{Prop}\to\mathsf{Prop}} G]\!] = \{\tau(\mathbf{0})\}$. En effet, les dénotations de F et de G sont données par

$$\begin{array}{rcl} [\![F]\!] & = & \big\{(\varnothing, \tau(\mathbf{1}))\big\} \\ [\![G]\!] & = & \Big\{\big(\{\tau(X)\}, \tau(X \to X)\big); \quad X \subset \mathcal{A}\ \mu\text{-fini}\Big\} \end{array}$$

et la dénotation du type $\mathsf{Prop} \to \mathsf{Prop}$ est la clique $\{\tau(X)\}$ où X est la base de type contenant les six cliques suivantes :

$$\begin{array}{rcll} f_1 & = & \big\{(\varnothing, \tau(\mathbf{0}))\big\} & \text{(constante } \mathbf{0} \text{ par vocation)} \\ f_2 & = & \big\{(\varnothing, \tau(\mathbf{1}))\big\} & \text{(constante } \mathbf{1} \text{ par vocation)} \\ f_3 & = & \big\{(\{\tau(\mathbf{0})\}, \tau(\mathbf{0}));\ (\{\tau(\mathbf{1})\}, \tau(\mathbf{0}))\big\} & \text{(constante } \mathbf{0} \text{ calculée)} \\ f_4 & = & \big\{(\{\tau(\mathbf{0})\}, \tau(\mathbf{1}));\ (\{\tau(\mathbf{1})\}, \tau(\mathbf{1}))\big\} & \text{(constante } \mathbf{1} \text{ calculée)} \\ f_5 & = & \big\{(\{\tau(\mathbf{0})\}, \tau(\mathbf{0}));\ (\{\tau(\mathbf{1})\}, \tau(\mathbf{1}))\big\} & \text{(identité)} \\ f_6 & = & \big\{(\{\tau(\mathbf{0})\}, \tau(\mathbf{1}));\ (\{\tau(\mathbf{1})\}, \tau(\mathbf{0}))\big\} & \text{(négation)} \end{array}$$

On vérifie alors aisément que

$$\begin{array}{rcl} \lfloor [\![F]\!] \rfloor_X & = & f_2 \ = \ \big\{(\varnothing, \tau(1))\big\} \qquad \text{(constante } \mathbf{1} \text{ par vocation)} \\ \lfloor [\![G]\!] \rfloor_X & = & f_4 \ = \ \big\{(\{\tau(0)\}, \tau(\mathbf{1}));\ (\{\tau(\mathbf{1})\}, \tau(\mathbf{1}))\big\} \qquad \text{(constante } \mathbf{1} \text{ calculée)} \end{array}$$

et $f_2 \neq f_4$, ce qui prouve que la proposition $F =_{\mathsf{Prop}\to\mathsf{Prop}} G$ est invalide dans $\mathcal{A}$, en vertu de la proposition 5.2.14.

Chapitre 6

Réalisabilité dans les espaces cohérents

Ce chapitre est consacré à la construction d'un modèle intuitionniste du calcul implicite restreint. Pour cela, nous dégagerons une notion d'espace cohérent avec contenu calculatoire — la notion de $\mathcal{K}$-*espace cohérent*, basée sur un *espace de calcul* $\mathcal{K}$ — et nous montrerons que cette notion induit naturellement une relation de réalisabilité entre les cliques de l'espace de calcul (qui joueront ici le même rôle que les codes de Gödel des fonctions récursives partielles dans la catégorie des ω-sets) et les cliques du modèle. Nous verrons en particulier que cette interprétation concrète de la réalisabilité fait disparaître les problèmes de représentation liés à l'équivalence de catégories entre ω-sets modestes et PER que nous avons évoquée au chapitre 3, puisque dans la catégorie des $\mathcal{K}$-espaces cohérents, ces deux notions sont remplacées par une seule et même entité : la notion de *base de type calculatoire*.

La seconde partie de ce chapitre sera consacrée à la construction du modèle proprement dit, dont nous montrerons qu'il invalide le principe du tiers-exclu, mais aussi le principe d'indiscernabilité des preuves.[1] Précisons que les caractéristiques intuitionnistes que nous évoquons ici seront l'attribut exclusif de la sorte Set. Dans le modèle que nous allons construire, nous continuerons en effet à interpréter la sorte Prop de manière classique, ne serait-ce que pour montrer que les interprétations classique et intuitionniste peuvent cohabiter sans danger dans un même cadre.

6.1 La catégorie des $\mathcal{K}$-espaces cohérents

6.1.1 Espace de calcul

Définition 6.1.1 (Espace de calcul) — On appelle *espace de calcul* tout espace cohérent $\mathcal{K}$ tel que $[\mathcal{K} \to \mathcal{K}] \Subset \mathcal{K}$.

D'après la proposition 4.3.20, tout espace de calcul $\mathcal{K}$ est muni d'une structure de λ-modèle sous-extensionnel $(\mathcal{K}, \cdot, [\![]\!], \subset)$. Dans ce qui suit, nous identifierons les termes du λ-calcul pur avec leur image dans $\mathcal{K}$ par la fonction d'interprétation $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{K}} \to \mathcal{K}$. On écrira par exemple

$$\forall\, k \in \mathcal{K} \qquad (\lambda x\,.\,k\ x) \subset k$$

pour

$$\forall\, k \in \mathcal{K} \qquad [\![\lambda x\,.\,y\ x]\!]_{y \leftarrow k} \subset k,$$

[1] Ce modèle sera donc un modèle de l'élimination forte [6].

ce qui exprime ici que la partie fonctionnelle $\lambda x\,.\,k\ x$ de n'importe quelle clique $k \in \mathcal{K}$ est une sous-clique de k. (On reconnaîtra dans cette écriture une formulation alternative de l'axiome de sous-extensionnalité.)

Dans la définition ci-dessus, l'inclusion rigide $[\mathcal{K} \to \mathcal{K}] \Subset \mathcal{K}$ fait référence au cadre des fonctions ($\aleph_0$-)stables. Bien que ce cadre soit sans doute le plus naturel pour exprimer une notion de réalisabilité (qui est une notion intimement liée à la calculabilité), le choix du cardinal de continuité $\mu = \aleph_0$ n'a d'un point de vue technique pas une grande importance, et on aurait pu définir une notion d'espace de calcul pour chaque cardinal régulier infini.

Dans tout ce qui suit, $\mathcal{K}$ désigne un espace de calcul fixé une fois pour toutes, et par lequel seront paramétrées toutes les constructions de ce chapitre. Un exemple d'espace de calcul convenable est donné par n'importe quelle solution de l'équation récursive de domaines

$$\mathcal{K} \quad = \quad [\mathcal{K} \to \mathcal{K}] \ \oplus \ \mathbb{N}_\perp$$

où $\mathbb{N}_\perp$ désigne l'espace plat dont les atomes sont les entiers naturels, qui jouent ici le rôle de *valeurs de base* de l'espace $\mathcal{K}$.[2]

6.1.2 $\mathcal{K}$-espaces cohérents

Définition 6.1.2 ($\mathcal{K}$-espace cohérent) — On appelle *$\mathcal{K}$-espace cohérent* tout espace cohérent $\mathcal{A}$ muni

1. d'un sous-espace $\|\mathcal{A}\| \Subset \mathcal{A}$, appelé *partie calculatoire* de $\mathcal{A}$;
2. d'un plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$.

Intuitivement, un $\mathcal{K}$-espace cohérent est un espace cohérent $\mathcal{A}$ dans lequel on a distingué un sous-espace particulier $\|\mathcal{A}\|$, lequel se plonge rigidement dans $\mathcal{K}$. Si $\mathcal{A}$ est un $\mathcal{K}$-espace cohérent, on dit d'un atome $\alpha \in |\mathcal{A}|$ qu'il est

- *calculatoire* si α appartient à la trame de $\|\mathcal{A}\|$, la partie calculatoire de $\mathcal{A}$.
- *non-calculatoire* si α appartient au complémentaire de la trame de $\|\mathcal{A}\|$ (dans $|\mathcal{A}|$).

Si a est une clique de $\mathcal{A}$, on dit que a est

- *purement calculatoire* si a ne contient que des atomes calculatoires ;
- *purement non-calculatoire* si a ne contient que des atomes non-calculatoires ;
- *mixte* si a contient à la fois un atome calculatoire et un atome non-calculatoire.

Ces trois classes de cliques sont essentiellement disjointes, mis à part le cas particulier de la clique vide qui est à la fois purement calculatoire et purement non-calculatoire.

Dans ce qui suit, on identifiera chaque atome calculatoire $\alpha \in \big|\|\mathcal{A}\|\big|$ avec son image $\phi_{\mathcal{A}}(\alpha)$ dans la trame de $\mathcal{K}$, de même qu'on identifiera chaque clique purement calculatoire $a \in \|\mathcal{A}\|$ avec son image $\phi^+_{\mathcal{A}}(a)$ dans l'espace de calcul $\mathcal{K}$. Au paragraphe 6.1.8, nous étudierons plus dans le détail les conditions dans lesquelles une telle identification peut avoir lieu.

[2]Dans cet exemple, il est nécessaire d'utiliser la définition simplifiée de la somme directe (*cf* paragraphe 4.1.6) afin d'obtenir une inclusion rigide $[\mathcal{K} \to \mathcal{K}] \Subset \mathcal{K}$ et non un simple plongement rigide. Si on utilise les entiers de la théorie des ensembles (avec le codage standard de von Neumann), il est facile de vérifier que les entiers ne sont jamais des couples, et que par conséquent, les trames $|\mathcal{A} \to \mathcal{A}| = \mathcal{A}_{\mathbf{fin}} \times |\mathcal{A}|$ et $|\mathbb{N}_\perp| = \mathbb{N}$ sont naturellement disjointes quel que soit l'espace cohérent $\mathcal{A}$.

Définition 6.1.3 (Contenu calculatoire d'une clique) — Soient $\mathcal{A}$ un $\mathcal{K}$-espace cohérent, et a une clique de $\mathcal{A}$. On appelle *contenu calculatoire* ou *partie calculatoire* de a la sous-clique de a notée $\|a\|$ et définie par :

$$\|a\| \quad = \quad a \cap \big|\|\mathcal{A}\|\big|.$$

Le contenu calculatoire d'une clique $a \in \mathcal{A}$, formé par l'ensemble des atomes calculatoires de a, est la plus grande sous-clique purement calculatoire de a. Bien entendu, on identifiera le contenu calculatoire d'une clique $a \in \mathcal{A}$ avec son image $\phi^+_{\mathcal{A}}(\|a\|)$ dans l'espace de calcul $\mathcal{K}$.

$\mathcal{K}$-espace cohérents et réalisabilité La notion de contenu calculatoire dans les $\mathcal{K}$-espaces cohérents peut également être exprimée sous la forme d'une relation de réalisabilité qui, à la différence de la réalisabilité "syntaxique" usuelle (*i.e.* la réalisabilité des ω-sets ou la réalisabilité modifiée des Λ-sets [6]) est une réalisabilité "sémantique" mettant en relation une clique de l'espace de calcul $\mathcal{K}$ avec une clique du $\mathcal{K}$-espace cohérent $\mathcal{A}$. Cette relation de réalisabilité sur $\mathcal{A}$ est définie de la manière suivante :

Définition 6.1.4 (Réalisabilité sur un $\mathcal{K}$-espace cohérent) — Soit $\mathcal{A}$ un $\mathcal{K}$-espace cohérent. On appelle relation de *réalisabilité* sur $\mathcal{A}$ la relation binaire $(\models_{\mathcal{A}}) \subset \mathcal{K} \times \mathcal{A}$ définie pour tout $k \in \mathcal{K}$ et pour tout $a \in \mathcal{A}$ par :

$$k \models_{\mathcal{A}} a \qquad \equiv \qquad \phi^-_{\mathcal{A}}(k) = \|a\|\,.$$

Lorsque $k \models_{\mathcal{A}} a$, on dit que k *réalise* a, ou encore que k *est un réalisateur de* a (dans $\mathcal{A}$).

Intuitivement, la relation $k \models_{\mathcal{A}} a$ signifie que les cliques a et k coïncident sur la trame de $\|\mathcal{A}\|$, la partie calculatoire de $\mathcal{A}$ (en considérant ici le plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$ comme une simple inclusion rigide). On vérifie aisément que toute clique $a \in \mathcal{A}$ admet au moins un réalisateur, qui est la clique $\phi^+_{\mathcal{A}}(\|a\|)$, et qui est par ailleurs le plus petit de tous les réalisateurs de a dans $\mathcal{K}$.

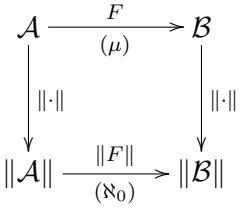
La réalisabilité sémantique que nous venons de définir est en général plus fine que la réalisabilité syntaxique usuelle, ce qui est dû principalement au fait qu'elle met en relation des objets de même nature. En particulier, le nombre de réalisateurs d'une clique $a \in \mathcal{A}$ donnée sera d'autant plus faible que la partie calculatoire de $\mathcal{A}$ représentera un sous-espace plus grand de $\mathcal{K}$ (modulo le plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$). Le cas limite correspond à la situation dans laquelle le plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$ est un isomorphisme entre $\|\mathcal{A}\|$ et l'espace de calcul $\mathcal{K}$. Dans ce cas, chaque clique $a \in \mathcal{A}$ admet un réalisateur et un seul, qui est la clique $\phi^+_{\mathcal{A}}(\|a\|) \in \mathcal{K}$ (c'est-à-dire la partie calculatoire de a modulo l'isomorphisme $\phi_{\mathcal{A}} : \|\mathcal{A}\| \approx \mathcal{K}$).

6.1.3 $(\mu, \mathcal{K})$-morphismes

Dans ce paragraphe, $\mathcal{A}$ et $\mathcal{B}$ désignent des $\mathcal{K}$-espaces cohérents, et μ un cardinal régulier infini.

Définition 6.1.5 ($(\mu, \mathcal{K})$-morphisme) — On appelle $(\mu, \mathcal{K})$-morphisme toute fonction μ-stable $F : \mathcal{A} \to \mathcal{B}$ telle qu'il existe une fonction $\aleph_0$-stable $\|F\| : \|\mathcal{A}\| \to \|\mathcal{B}\|$ telle que le

diagramme



commute, c'est-à-dire :

$$\forall\, a \in \mathcal{A} \qquad \|F(a)\| \quad = \quad \|F\|\,(\|a\|). \tag{6.1}$$

Intuitivement, une fonction μ-stable $F : \mathcal{A} \to \mathcal{B}$ est un $(\mu, \mathcal{K})$-morphisme si

1. le contenu calculatoire de l'image d'une clique $a \in \mathcal{A}$ par F ne dépend que du contenu calculatoire de a ;
2. cette dépendance entre les contenus calculatoires de a et de $F(a)$ est exprimée par une fonction $\|F\| : \|\mathcal{A}\| \to \|\mathcal{B}\|$ qui est $\aleph_0$-stable, et pas seulement μ-stable.

Par ailleurs, cette fonction $\|F\|$, lorsqu'elle existe, est unique et est caractérisée par

$$\forall\, a \in \|\mathcal{A}\| \qquad\qquad \|F\|\,(a) \quad = \quad \|F(a)\|$$

(en utilisant l'inclusion rigide $\|\mathcal{A}\| \Subset \mathcal{A}$).

La fonction identité $\mathsf{id}_{\mathcal{A}}$ est de manière immédiate un $(\mu, \mathcal{K})$-morphisme (il suffit de prendre $\|\mathsf{id}_{\mathcal{A}}\| = \mathsf{id}_{\|\mathcal{A}\|}$) et la composée de deux $(\mu, \mathcal{K})$-morphismes est encore un $(\mu, \mathcal{K})$-morphisme (en posant $\|G \circ F\| = \|G\| \circ \|F\|$). Par conséquent, la classe des $\mathcal{K}$-espaces cohérents munie des $(\mu, \mathcal{K})$-morphismes constitue une catégorie, que l'on appelle *catégorie des* $(\mu, \mathcal{K})$*-espaces cohérents*.

La notion de $(\mu, \mathcal{K})$-morphisme peut être caractérisée de manière directe au niveau des traces des fonctions μ-stables sous-jacentes :

Proposition 6.1.6 (Caractérisation de la trace d'un $(\mu, \mathcal{K})$-morphisme) — *Une fonction μ-stable $F : \mathcal{A} \xrightarrow{\mu} \mathcal{B}$ est un $(\mu, \mathcal{K})$-morphisme si et seulement si sa trace $\mathsf{Tr}(F)$ satisfait la condition suivante :*

$$\forall\, (a_0, \beta) \in \mathsf{Tr}(F) \qquad\qquad \beta \in \big|\|\mathcal{B}\|\big| \quad \Rightarrow \quad a_0 \in \|\mathcal{A}\|_{\mathbf{fin}}\,.$$

Lorsque c'est le cas, la fonction $\|F\| : \|\mathcal{A}\| \to \|\mathcal{B}\|$ est donnée par

$$\mathsf{Tr}(\|F\|) \quad = \quad \Big\{(a_0, \beta) \in \mathsf{Tr}(F); \quad \beta \in \big|\|\mathcal{B}\|\big|\Big\}.$$

Preuve. (Partie directe) Supposons que $F : \mathcal{A} \to \mathcal{B}$ est un $(\mu, \mathcal{K})$-morphisme, et considérons un atome $(a_0, \beta) \in \mathsf{Tr}(F)$ tel que $\beta \in \big|\|\mathcal{B}\|\big|$. Comme β est calculatoire, $\beta \in \|F(a_0)\|$ d'où l'on tire que $\beta \in \|F\|\,(\|a_0\|)$ d'après (6.1). Par définition de $\mathsf{Tr}(\|F\|)$, il existe $a_1 \subset \|a_0\|$ tel que $(a_1, \beta) \in \mathsf{Tr}(\|F\|)$. Comme $\beta \in \|F\|\,(a_1)$ et

$$\|F\|\,(a_1) \;\subset\; \|F\|\,(\|a_0\|) \;=\; \|F(a_0)\| \;\subset F(a_0)$$

on a $\beta \in F(a_0)$, d'où il ressort que $a_1 = a_0$ puisque $a_1 \subset \|a_0\| \subset a_0$ et $(a_0, \beta) \in \mathsf{Tr}(F)$. Par conséquent, l'atome (a_0, β), égal à (a_1, β), est aussi un élément de la trace de $\|F\|$, d'où l'on tire que $a_0 \in \|\mathcal{A}\|_{\mathbf{fin}}$.

(Partie réciproque) Supposons que $F : \mathcal{A} \to \mathcal{B}$ est une fonction μ-stable telle que pour tout atome $(a_0, \beta) \in \mathsf{Tr}(F)$, on ait $a_0 \in \|\mathcal{A}\|_{\mathbf{fin}}$ dès lors que $\beta \in \big|\|\mathcal{B}\|\big|$. Considérons la fonction $\aleph_0$-stable $F' : \|\mathcal{A}\| \to \|\mathcal{B}\|$ définie par sa trace

$$\mathsf{Tr}(F') \quad = \quad \Big\{(a_0, \beta) \in \mathsf{Tr}(F); \quad \beta \in \big|\|\mathcal{B}\|\big|\Big\} \quad \in \quad \|\mathcal{A}\| \stackrel{\aleph_0}{\to} \|\mathcal{B}\|$$

(cette définition est correcte d'après notre hypothèse). On vérifie aisément que pour toute clique $a \in \mathcal{A}$ et pour tout atome calculatoire $\beta \in \big|\|\mathcal{B}\|\big|$, on a $\beta \in F(a)$ si et seulement si $\beta \in F'(\|a\|)$. Par conséquent, on a $\|F(a)\| = F'(\|a\|)$ pour tout $a \in \mathcal{A}$, d'où il ressort que F est un $(\mu, \mathcal{K})$-morphisme et $\|F\| = F'$. La formule caractérisant la trace de $\|F\|$ est alors donnée par la définition de F'. □

Intuitivement, la proposition ci-dessus exprime qu'au niveau atomique, un $(\mu, \mathcal{K})$-morphisme est une fonction μ-stable qui pour produire un atome calculatoire $\beta \in \big|\|\mathcal{B}\|\big|$ ne consulte que des atomes calculatoires $\alpha_1, \ldots, \alpha_1 \in \big|\|\mathcal{A}\|\big|$, en nombre fini. Le positionnement d'un atome (a_0, β) de la trace d'un $(\mu, \mathcal{K})$-morphisme par rapport aux catégories calculatoire/non-calculatoire est résumé par le tableau suivant

$(a_0, \beta) \in \mathsf{Tr}(F)$	$\beta \in \big\|\|\mathcal{B}\|\big\|$	$\beta \notin \big\|\|\mathcal{B}\|\big\|$
$a_0 \in \|\mathcal{A}\|_{\mathbf{fin}}$	$*$	$*$
$a_0 \notin \|\mathcal{A}\|_{\mathbf{fin}}$	$\varnothing$	$*$

dans lequel les positions autorisées sont marquées par le symbole "$*$". La proposition 6.1.6 justifie par ailleurs la définition suivante :

Définition 6.1.7 (Espace cohérent des $(\mu, \mathcal{K})$-morphismes) — On appelle *espace cohérent des $(\mu, \mathcal{K})$-morphismes* le sous-espace de $\mathcal{A} \stackrel{\mu}{\to} \mathcal{B}$ noté $\mathcal{A} \stackrel{\mu}{\to}_* \mathcal{B}$ dont la trame est définie par

$$\big|\mathcal{A} \stackrel{\mu}{\to}_* \mathcal{B}\big| \quad = \quad \Big\{(a_0, \beta) \in \mathcal{A}_{(\mu)} \times |\mathcal{B}|\,; \quad \beta \in \big|\|\mathcal{B}\|\big| \quad \Rightarrow \quad a_0 \in \|\mathcal{A}\|_{\mathbf{fin}}\Big\}.$$

Cet espace a une structure de $\mathcal{K}$-espace cohérent dont la partie calculatoire est donnée par

$$\big\|\mathcal{A} \stackrel{\mu}{\to}_* \mathcal{B}\big\| \quad = \quad \|\mathcal{A}\| \to \|\mathcal{B}\|$$

et où le plongement rigide $\phi : \big\|\mathcal{A} \stackrel{\mu}{\to}_* \mathcal{B}\big\| \rightarrowtail \mathcal{K}$ qui lui est associé est défini par

$$\phi(a_0, \beta) \quad = \quad \big(\phi^+_{\mathcal{A}}(a_0), \phi_{\mathcal{B}}(\beta)\big) \quad \in \quad |\mathcal{K} \to \mathcal{K}| \quad \subset \quad |\mathcal{K}|$$

pour tout couple $(a_0, \beta) \in \|\mathcal{A}\|_{\mathbf{fin}} \times \big|\|\mathcal{B}\|\big|$.

Si $F : \mathcal{A} \to \mathcal{B}$ est un $(\mu, \mathcal{K})$-morphisme et $\|F\| : \|\mathcal{A}\| \to \|\mathcal{B}\|$ l'unique fonction stable satisfaisant l'équation (6.1), la trace de $\|F\|$ est donnée par

$$\begin{aligned}
\mathsf{Tr}(\|F\|) \quad &= \quad \Big\{(a_0, \beta) \in \mathsf{Tr}(F); \quad \beta \in \big|\|\mathcal{B}\|\big|\Big\} \\
&= \quad \Big\{(a_0, \beta) \in \mathsf{Tr}(F); \quad a_0 \in \|\mathcal{A}\|_{\mathbf{fin}} \quad \text{et} \quad \beta \in \big|\|\mathcal{B}\|\big|\Big\} \qquad \text{(Prop. 6.1.6)} \\
&= \quad \mathsf{Tr}(F) \ \cap \ \big(\|\mathcal{A}\|_{\mathbf{fin}} \times \big|\|\mathcal{B}\|\big|\big) \quad = \quad \mathsf{Tr}(F) \ \cap \ \Big|\big\|\mathcal{A} \stackrel{\mu}{\to}_* \mathcal{B}\big\|\Big|,
\end{aligned}$$

ce qui montre que la fonction $\|F\|$ n'est autre que la partie calculatoire de F dans l'espace des $(\mu,\mathcal{K})$-morphismes, et justifie *a posteriori* la notation $\|F\|$.

Dans le modèle que nous construirons dans la section suivante, les types imprédicatifs seront interprétés par des bases de types formées uniquement avec des cliques purement calculatoires. Le comportement des types imprédicatifs vis-à-vis du produit dépendant (explicite) peut déjà s'observer dans le lemme suivant :

Lemme 6.1.8 — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux $\mathcal{K}$-espaces cohérents, et $(a_0,\beta) \in \left|\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}\right|$ un atome de l'espace des $(\mu,\mathcal{K})$-morphismes de $\mathcal{A}$ dans $\mathcal{B}$. Pour que (a_0,β) soit un atome calculatoire (resp. non-calculatoire) de $\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}$, il faut et il suffit que β soit un atome calculatoire (resp. non-calculatoire) de $\mathcal{B}$.*

Preuve. Cette propriété résulte immédiatement de la définition précédente. □

Autrement dit, un $(\mu,\mathcal{K})$-morphisme $F : \mathcal{A} \overset{\mu}{\to}_* \mathcal{B}$ est purement calculatoire si et seulement si l'image par F de n'importe quelle clique de $\mathcal{A}$ est une clique purement calculatoire de $\mathcal{B}$, et cela quel que soit le $\mathcal{K}$-espace cohérent $\mathcal{A}$.

Réalisabilité et $(\mu,\mathcal{K})$-morphismes Au paragraphe 6.1.2 nous avons montré que la structure de $\mathcal{K}$-espace cohérent induit naturellement une relation de réalisabilité sémantique sur l'espace cohérent sous-jacent. La notion de $(\mu,\mathcal{K})$-morphisme peut être elle même caractérisée par cette relation de réalisabilité de la manière suivante :

Lemme 6.1.9 (Caractérisation des $(\mu,\mathcal{K})$-morphismes) — *Soit $F : \mathcal{A} \to \mathcal{B}$ une fonction μ-stable. Pour que F soit un $(\mu,\mathcal{K})$-morphisme, il faut et il suffit qu'il existe une clique $k \in \mathcal{K}$ telle que*

$$\forall\, l \in \mathcal{K} \quad \forall\, a \in \mathcal{A} \qquad l \models_{\mathcal{A}} a \quad \Rightarrow \quad k \cdot l \models_{\mathcal{B}} F(a) \tag{6.2}$$

Preuve. La condition d'existence est nécessaire, puisque la clique calculatoire $k \in \mathcal{K}$ définie par $k = \phi^+(\|F\|)$ satisfait (6.2), où $\phi : \left\|\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}\right\| \rightarrowtail \mathcal{K}$ désigne le plongement rigide de la partie calculatoire de $\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}$ dans $\mathcal{K}$ induit par la structure de $\mathcal{K}$-espace cohérent sur $\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}$. Pour montrer que cette condition est suffisante, supposons qu'il existe une clique $k \in \mathcal{K}$ satisfaisant (6.2). On pose alors

$$\|F\| \quad = \quad \Big\{(a_0,\beta) \in \big|\|\mathcal{A}\| \to \|\mathcal{B}\|\big|; \qquad \big(\phi^+_{\mathcal{A}}(a_0), \phi_{\mathcal{B}}(\beta)\big) \in k\Big\}$$

et on vérifie aisément en utilisant (6.2) que $\|F(a)\| = \|F\|\,(\|a\|)$ pour tout $a \in \mathcal{A}$. □

Il est par ailleurs clair que la relation de réalisabilité sur l'espace $\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}$ des $(\mu,\mathcal{K})$-morphismes est caractérisée par une formule très analogue à celle qui définit la relation de réalisabilité sur l'espace des fonctions dans les ω-sets.

Lemme 6.1.10 (Réalisabilité sur l'espace des $(\mu,\mathcal{K})$-morphismes) — *Pour toute clique $k \in \mathcal{K}$ et pour tout $(\mu,\mathcal{K})$-morphisme $F : \mathcal{A} \to \mathcal{B}$ on a l'équivalence :*

$$k \models_{\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}} F \qquad \Leftrightarrow \qquad \big(\forall\, l \in \mathcal{K} \quad \forall\, a \in \mathcal{A} \qquad l \models_{\mathcal{A}} a \quad \Rightarrow \quad k \cdot l \models_{\mathcal{B}} F(a)\big).$$

Quasi-$\mathcal{K}$-morphismes De manière analogue à la notion de $(\mu,\mathcal{K})$-morphisme, on définit la notion de *quasi-$\mathcal{K}$-morphisme*, en remplaçant dans la définition 6.1.5 le critère de μ-stabilité par le critère (plus faible) de quasi-stabilité. (Notons que la condition de $\aleph_0$-stabilité sur la fonction $\|F\|$ reste inchangée.) Bien entendu, les propriétés énoncées ci-dessus se généralisent immédiatement aux quasi-$\mathcal{K}$-morphismes, simplement en remarquant que la notion de quasi-$\mathcal{K}$-morphisme se confond avec celle de $(\mu,\mathcal{K})$-morphisme lorsque le cardinal de continuité μ est suffisamment grand (les $\mathcal{K}$-espaces cohérents $\mathcal{A}$ et $\mathcal{B}$ étant fixés).

Dans ce qui suit, on notera $\mathcal{A} \overset{q}{\to}_* \mathcal{B}$ l'espace cohérent des quasi-$\mathcal{K}$-morphismes de $\mathcal{A}$ dans $\mathcal{B}$, dont la trame est donnée par

$$\left|\mathcal{A} \overset{q}{\to}_* \mathcal{B}\right| \quad = \quad \Big\{(a_0,\beta) \in \mathcal{A} \times |\mathcal{B}| \,;\quad \beta \in \big|\|\mathcal{B}\|\big| \quad \Rightarrow \quad a_0 \in \|\mathcal{A}\|_{\mathbf{fin}}\Big\},$$

et dont la structure de $\mathcal{K}$-espace cohérent est donnée par les mêmes formules que pour l'espace des $(\mu,\mathcal{K})$-morphismes.

6.1.4 Structure de catégorie cartésienne fermée

Avant de pouvoir montrer que la catégorie des $(\mu,\mathcal{K})$-espaces cohérents est une catégorie cartésienne fermée (dont l'exponentielle est l'espace des $(\mu,\mathcal{K})$-morphismes), nous devons d'abord dégager une notion de produit cartésien dans cette catégorie, ce qui revient essentiellement à montrer que le produit direct $\mathcal{A}\&\mathcal{B}$ de deux $\mathcal{K}$-espaces cohérents peut être également muni d'une structure de $\mathcal{K}$-espace cohérent. Bien entendu, si $\mathcal{A}$ et $\mathcal{B}$ sont deux $\mathcal{K}$-espaces cohérents, il est naturel de définir la partie calculatoire de $\mathcal{A}\,\&\,\mathcal{B}$ par $\|\mathcal{A}\,\&\,\mathcal{B}\| = \|\mathcal{A}\|\,\&\,\|\mathcal{B}\|$. La seule difficulté vient de ce qu'il n'existe pas de manière immédiate un plongement rigide de $\|\mathcal{A}\|\,\&\,\|\mathcal{B}\|$ dans l'espace $\mathcal{K}$, puisque nous n'avons pas supposé que $[\mathcal{K}\,\&\,\mathcal{K}] \Subset \mathcal{K}$.

Pour résoudre ce problème, nous allons utiliser le fait que $\mathcal{K}$ est un modèle du λ-calcul pour simuler le produit cartésien à l'intérieur de $\mathcal{K}$. Remarquons d'abord que si $\mathcal{K}$ est l'espace cohérent nul, le problème est résolu puisque la notion de $\mathcal{K}$-espace cohérent se confond avec la notion d'espace cohérent. Dans ce qui suit, on suppose donc que l'espace de calcul $\mathcal{K}$ est non nul, et on se donne un atome arbitraire $\bullet \in |\mathcal{K}|$. On définit alors les cliques mono-atomiques $1_* \in \mathcal{K}$ et $2_* \in \mathcal{K}$ en posant

$$1_* \;=\; \big\{(\{\bullet\},(\varnothing,\bullet))\big\} \qquad \text{et} \qquad 2_* \;=\; \big\{(\varnothing,(\{\bullet\},\bullet))\big\}$$

et on vérifie aisément que les cliques fonctionnelles 1_* et 2_* sont incompatibles entre elles,[3] ce qui entraîne immédiatement le résultat suivant :

Lemme 6.1.11 (Produit direct dans $\mathcal{K}$) — *Le sous-ensemble*

$$\{1_*;2_*\} \times |\mathcal{K}| \quad \subset \quad \mathcal{K}_{\mathbf{fin}} \times |\mathcal{K}| \quad = \quad |\mathcal{K} \to \mathcal{K}| \quad \subset \quad |\mathcal{K}|$$

définit un sous-espace de $\mathcal{K}$ qui est isomorphe au produit direct $\mathcal{K}\,\&\,\mathcal{K}$.

Preuve. L'isomorphisme au niveau des relations de cohérence repose essentiellement sur le fait que les cliques 1_* et 2_* sont incompatibles dans $\mathcal{K} \to \mathcal{K}$ et que par conséquent, la relation de cohérence dans $\{1_*;2_*\} \times |\mathcal{K}|$ issue de l'espace des fonctions stables $\mathcal{K} \to \mathcal{K}$ est donnée par la même formule que celle qui définit la relation de cohérence sur $\{1;2\} \times |\mathcal{K}| = |\mathcal{K}\,\&\,\mathcal{K}|$. □

[3] Ces deux cliques n'ont pas été choisies au hasard, puisqu'on a $1_* \subset \lambda xy\,.\,x$ et $2_* \subset \lambda xy\,.\,y$.

Définition 6.1.12 (Produit direct de deux $\mathcal{K}$-espaces cohérents) — Soient $\mathcal{A}$ et $\mathcal{B}$ des $\mathcal{K}$-espaces cohérents. Le produit direct $\mathcal{A} \,\&\, \mathcal{B}$ est naturellement muni d'une structure de $\mathcal{K}$-espace cohérent dans laquelle la partie calculatoire de $\mathcal{A} \,\&\, \mathcal{B}$ est définie par $\|\mathcal{A} \,\&\, \mathcal{B}\| = \|\mathcal{A}\| \,\&\, \|\mathcal{B}\|$, et où le plongement rigide $\phi_{\mathcal{A}\&\mathcal{B}} : \|\mathcal{A} \,\&\, \mathcal{B}\| \rightarrowtail \mathcal{K}$ est défini par

$$\phi_{\mathcal{A}\&\mathcal{B}}((1,\alpha)) \;=\; (1_*, \phi_{\mathcal{A}}(\alpha)) \qquad \text{et} \qquad \phi_{\mathcal{A}\&\mathcal{B}}((2,\beta)) \;=\; (2_*, \phi_{\mathcal{B}}(\beta))$$

pour tous $\alpha \in \big|\|\mathcal{A}\|\big|$ et $\beta \in \big|\|\mathcal{B}\|\big|$.

On vérifie immédiatement que la somme directe $\mathcal{A} \,\&\, \mathcal{B}$ munie de la structure de $\mathcal{K}$-espace cohérent que nous venons de définir est un produit cartésien (au sens catégorique) dans la catégorie des $(\mu, \mathcal{K})$-espaces cohérents, laquelle est donc de ce fait une catégorie cartésienne.

Proposition 6.1.13 (Fermeture cartésienne) — *La catégorie des $(\mu, \mathcal{K})$-espaces cohérents est une catégorie cartésienne fermée dont l'exponentielle est le bi-foncteur*

$$(\mathcal{A}, \mathcal{B}) \quad \mapsto \quad [\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}].$$

Preuve. Soient $\mathcal{A}$, $\mathcal{B}$ et $\mathcal{C}$ des $\mathcal{K}$-espaces cohérents. Considérons la fonction

$$\mathbf{App} \quad : \quad [\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}] \,\&\, \mathcal{A} \to \mathcal{B}$$

définie par $\mathbf{App}(F, a) = F(a)$ pour tout $F \in \mathcal{A} \xrightarrow{\mu}_* \mathcal{B}$ et pour tout $a \in \mathcal{A}$ (en identifiant le produit direct $[\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}] \,\&\, \mathcal{A}$ avec $[\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}] \times \mathcal{A}$). D'après la proposition 4.3.19, cette fonction est clairement μ-stable, et sa trace est donnée par

$$\mathsf{Tr}(\mathbf{App}) \quad = \quad \Big\{ \big(\{(1,(a,\beta))\} \;\cup\; \{2\} \times a,\; \beta \big) \Big\}$$

où (a, β) décrit l'ensemble des couples de $\mathcal{A}_{(\mu)} \times |\mathcal{B}|$ tels que $\beta \in \big|\|\mathcal{B}\|\big|$ entraîne $a \in \|\mathcal{A}\|_{\mathbf{fin}}$. D'après la proposition 6.1.6, il est immédiat que $\mathbf{App}$ est un $(\mu, \mathcal{K})$-morphisme. Considérons à présent un $(\mu, \mathcal{K})$-morphisme $F : \mathcal{A} \,\&\, \mathcal{B} \xrightarrow{\mu}_* \mathcal{C}$. Comme F est une fonction μ-stable, il existe d'après la proposition 4.3.19 une fonction $F' : \mathcal{A} \xrightarrow{\mu} (\mathcal{B} \xrightarrow{\mu} \mathcal{C})$ telle que

$$F'(a)(b) \quad = \quad F(\{1\} \times a \;\cup\; \{2\} \times b)$$

pour toutes cliques $a \in \mathcal{A}$ et $b \in \mathcal{B}$, et dont la trace est donnée par

$$\mathsf{Tr}(F') \quad = \quad \Big\{ \big(a, (b,\gamma)\big); \quad \big(\{1\}\times a \cup \{2\}\times b, \gamma\big) \in \mathsf{Tr}(F) \Big\}.$$

D'après la proposition 6.1.6, il est clair que $F' \in \mathcal{A} \xrightarrow{\mu}_* (\mathcal{B} \xrightarrow{\mu}_* \mathcal{C})$. □

6.1.5 Plongements rigides et colimites

Dans la catégorie des $\mathcal{K}$-espaces cohérents, nous ne pouvons plus travailler avec les notions usuelles de plongement rigide et d'inclusion rigide qui, en général, ne préservent pas la structure calculatoire des $\mathcal{K}$-espaces cohérents. Pour cela, il est donc nécessaire d'introduire des notions de plongement et d'inclusion rigides calculatoires :

Définition 6.1.14 (Plongement rigide calculatoire) — Soient $\mathcal{A}$ et $\mathcal{A}'$ deux $\mathcal{K}$-espaces cohérents. On appelle *plongement rigide calculatoire* de $\mathcal{A}$ dans $\mathcal{A}'$ tout plongement rigide $\phi : \mathcal{A} \rightarrowtail \mathcal{A}'$ tel que pour tout atome $\alpha \in |\mathcal{A}|$ on ait l'équivalence

$$\alpha \in \big|\|\mathcal{A}\|\big| \qquad \Leftrightarrow \qquad \phi(\alpha) \in \big|\|\mathcal{A}'\|\big|.$$

Si ϕ est plongement rigide calculatoire de $\mathcal{A}$ dans $\mathcal{B}$, on le note $\phi : \mathcal{A} \rightarrowtail_* \mathcal{B}$.

De même, nous dirons qu'une inclusion rigide $\mathcal{A} \Subset \mathcal{A}'$ est calculatoire, ce que l'on notera $\mathcal{A} \Subset_* \mathcal{A}'$, si le plongement rigide sous-jacent est un plongement rigide calculatoire, c'est-à-dire :

$$\mathcal{A} \Subset_* \mathcal{A}' \qquad \Leftrightarrow \qquad \mathcal{A} \Subset \mathcal{A} \quad \text{et} \quad \big|\|\mathcal{A}\|\big| \;=\; \big|\|\mathcal{A}'\|\big| \cap \big|\mathcal{A}\big|.$$

On vérifie immédiatement qu'une inclusion rigide $\mathcal{A} \Subset \mathcal{A}'$ est calculatoire si et seulement si le plongement rigide induit par l'inclusion de $|\mathcal{A}|$ dans $|\mathcal{A}'|$ est un plongement rigide calculatoire au sens de la définition précédente.

Définition 6.1.15 (Colimite de $\mathcal{K}$-espaces cohérents) — Soit $(\mathcal{A})_{i\in I}$ une famille croissante de $\mathcal{K}$-espaces cohérents (au sens des inclusions rigides calculatoires) indicée par un ensemble dirigé I. La colimite de la famille $(\mathcal{A}_i)_{i\in I}$ est naturellement munie d'une structure de $\mathcal{K}$-espace cohérent en posant

$$\Big\|\operatorname*{colim}_{i\in I} \mathcal{A}_i\Big\| \quad = \quad \operatorname*{colim}_{i\in I} \|\mathcal{A}_i\|$$

et où le plongement rigide $\phi : \|\operatorname{colim} \mathcal{A}_i\| \rightarrowtail \mathcal{K}$ correspondant est défini par $\phi = \bigcup \phi_{\mathcal{A}_i}$.

De la même manière que l'exponentielle μ-stable est un bi-foncteur μ-continu dans la catégorie des plongements rigides (Prop. 4.3.18), le foncteur $(\mathcal{A}, \mathcal{B}) \mapsto [\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}]$ associant à chaque couple de $\mathcal{K}$-espaces cohérents l'espace des $(\mu, \mathcal{K})$-morphismes correspondant est lui aussi un bi-foncteur μ-continu dans la catégorie des plongements rigides calculatoires :

Proposition 6.1.16 (Colimites μ-dirigées) — *La correspondance $(\mathcal{A}, \mathcal{B}) \mapsto (\mathcal{A} \xrightarrow{\mu}_* \mathcal{B})$ définit un bi-foncteur covariant μ-continu dans la catégorie des plongements rigides calculatoires, c'est-à-dire :*

1. *Si $\mathcal{A} \Subset_* \mathcal{A}'$ et $\mathcal{B} \Subset_* \mathcal{B}'$, alors $(\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}) \Subset_* (\mathcal{A}' \xrightarrow{\mu}_* \mathcal{B}')$.*

2. *Si $(\mathcal{A}_i)_{i\in I}$ et $(\mathcal{B}_i)_{i\in I}$ sont deux familles croissantes d'espaces cohérents (au sens des inclusions rigides calculatoires) indicées par un même ensemble μ-dirigé I, on a :*

$$\operatorname*{colim}_{i\in I}\big(\mathcal{A}_i \xrightarrow{\mu}_* \mathcal{B}_i\big) \quad = \quad \Big(\operatorname*{colim}_{i\in I} \mathcal{A}_i\Big) \xrightarrow{\mu}_* \Big(\operatorname*{colim}_{i\in I} \mathcal{B}_i\Big).$$

Preuve. Le premier point est immédiat. La preuve du second point s'effectue de la même manière que pour la proposition 4.3.18, et repose essentiellement sur le fait que μ est un cardinal régulier. □

Plongements rigides et réalisabilité Soient $\mathcal{A}$ et $\mathcal{A}'$ deux $\mathcal{K}$-espaces cohérents. La relation de réalisabilité sur $\mathcal{A}$ n'est en général pas préservée par un plongement rigide calculatoire $\phi : \mathcal{A} \rightarrowtail_* \mathcal{A}'$ (ou même une inclusion rigide calculatoire), c'est-à-dire :

$$k \Vdash_{\mathcal{A}} a \qquad \not\Rightarrow \qquad k \Vdash_{\mathcal{A}'} \phi^+(a).$$

Ceci est dû au fait qu'en général la partie calculatoire de $\mathcal{A}'$ est plus grande que celle de $\mathcal{A}$ (modulo le plongement rigide de $\|\mathcal{A}\|$ dans $\|\mathcal{A}'\|$ induit par ϕ). Pour cette raison, la relation de réalisabilité sur le sur-espace $\mathcal{A}'$ est en réalité plus fine que la relation de réalisabilité sur le sous-espace $\mathcal{A}$, c'est-à-dire :

$$\forall\, k \in \mathcal{K} \quad \forall\, a \in \mathcal{A}' \qquad k \Vdash_{\mathcal{A}'} a \qquad \Rightarrow \qquad k \Vdash_{\mathcal{A}} \phi^-(a).$$

Autrement dit, la relation de réalisabilité est contravariante vis-à-vis des plongements rigides (ou inclusions rigides) calculatoires : plus le $\mathcal{K}$-espace cohérent $\mathcal{A}$ est grand, et plus la relation de réalisabilité sur cet espace est fine. Dans le cas de la colimite d'une famille croissante $(\mathcal{A}_i)_{i\in I}$ de $\mathcal{K}$-espaces cohérents, la relation de réalisabilité est égale à l'intersection des relations de réalisabilité sur chacun des espaces $\mathcal{A}_i$:

$$\forall\, k \in \mathcal{K} \qquad \forall\, a \in \operatorname*{colim}_{i\in I} \mathcal{A}_i \qquad k \Vdash_{\operatorname{colim} \mathcal{A}_i} a \qquad \Leftrightarrow \qquad \forall\, i \in I \quad k \Vdash_{\mathcal{A}_i} (a \cap |\mathcal{A}_i|).$$

Dans le cas où le plongement rigide calculatoire $\phi : \mathcal{A} \rightarrowtail_* \mathcal{A}'$ induit un isomorphisme entre les parties calculatoires $\|\mathcal{A}\|$ et $\|\mathcal{A}'\|$, on a cependant les équivalences suivantes :

$$\begin{array}{llll} \forall\, k \in \mathcal{K} & \forall\, a \in \mathcal{A} & k \Vdash_{\mathcal{A}} a & \Leftrightarrow \quad k \Vdash_{\mathcal{A}'} \phi^+(a) \\ \forall\, k \in \mathcal{K} & \forall\, a' \in \mathcal{A}' & k \Vdash_{\mathcal{A}'} a' & \Leftrightarrow \quad k \Vdash_{\mathcal{A}} \phi^-(a'). \end{array}$$

6.1.6 Types dans les $\mathcal{K}$-espaces cohérents

Dans ce paragraphe, nous nous proposons d'étudier le comportement des bases de types et des types sémantiques dans le cadre des $\mathcal{K}$-espaces cohérents. Dans ce nouveau cadre, la définition des bases de types et des types sémantiques (paragraphe 5.1.1) est inchangée. En revanche, l'utilisation d'une structure calculatoire nécessite deux changements dans la théorie.

Le premier changement concerne la définition de la base de type $\Pi(a{\in}X; Y_a)$ désignant le produit d'une famille de bases de types, qu'il s'agit de définir dans l'espace cohérent des quasi-$\mathcal{K}$-morphismes et non dans l'espace des fonctions quasi-stables.

Définition 6.1.17 (Produit dépendant) — Soient $\mathcal{A}$ et $\mathcal{B}$ deux $\mathcal{K}$-espaces cohérents, et μ un cardinal régulier. Si X est une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$, on note $\Pi(a{\in}X; Y_a)$ la base de type de $\mathcal{A} \overset{q}{\to}_* \mathcal{B}$ définie par :

$$\uparrow\Pi(a{\in}X; Y_a) \quad = \quad \big\{F : \mathcal{A} \overset{q}{\to}_* \mathcal{B}; \qquad \forall\, a \in X; \quad F(a) \in \uparrow Y_a\big\}.$$

(La définition de $\forall(a{\in}X; Y_a)$, qui est une base de type de $\mathcal{B}$, reste inchangée.)

On vérifiera sans peine que les divers résultats énoncés au paragraphe 5.1.3 (indépendance de l'univers du discours, sous-typage, etc.) s'adaptent de manière immédiate au cadre des $\mathcal{K}$-espaces cohérents avec la nouvelle définition du produit dépendant. Parmi ces résultats, mentionnons les deux propositions suivantes qui concernent les questions de μ-continuité :

Proposition 6.1.18 (Produit dépendant μ-fini) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des $\mathcal{K}$-espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X. Si X est μ-fini, alors le produit dépendant $\Pi(a\in X; Y_a) \subset [\mathcal{A} \overset{q}{\to}_* \mathcal{B}]$ ne contient que des $(\mu,\mathcal{K})$-morphismes, c'est-à-dire :*

$$X \ \ \mu\text{-fini} \qquad \Rightarrow \qquad \Pi(a\in X; Y_a) \ \ \subset \ \ \mathcal{A} \overset{\mu}{\to}_* \mathcal{B} \ \ \Subset \ \ \mathcal{A} \overset{q}{\to}_* \mathcal{B}.$$

Proposition 6.1.19 (Clôture des univers) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des $\mathcal{K}$-espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X. Si X et tous les Y_a ($a \in X$) sont des bases de types μ-finies, et si μ est dénombrable ou inaccessible, alors le produit dépendant $\Pi(a\in X; Y_a)$ est une base de type μ-finie de $\mathcal{A} \overset{\mu}{\to}_* \mathcal{B}$.*

Le second changement dans la théorie des bases de types est un changement mineur, puisqu'il consiste simplement à munir les espaces $\mathsf{Ty}(\mathcal{A})$ et $\mathsf{Ty}_\mu(\mathcal{A})$ d'une structure de $\mathcal{K}$-espace cohérent dans laquelle, par définition, les atomes typiques $\tau(X)$ sont des atomes non-calculatoires.

Définition 6.1.20 (Structure de $\mathcal{K}$-espace cohérent sur $\mathsf{Ty}(\mathcal{A})$) — Pour tout $\mathcal{K}$-espace cohérent $\mathcal{A}$, on munit les espaces $\mathsf{Ty}(\mathcal{A})$ et $\mathsf{Ty}_\mu(\mathcal{A})$ d'une structure de $\mathcal{K}$-espace cohérent en posant

$$\|\mathsf{Ty}(\mathcal{A})\| \quad = \quad \big\|\mathsf{Ty}_\mu(\mathcal{A})\big\| \quad = \quad \mathsf{0}.$$

D'après cette définition, il est clair que n'importe quelle clique de l'espace $\mathsf{Ty}(\mathcal{A})$ (resp. $\mathsf{Ty}_\mu(\mathcal{A})$) est réalisée par n'importe quelle clique de l'espace de calcul $\mathcal{K}$ (les relations de réalisabilité $\Vdash_{\mathsf{Ty}(\mathcal{A})}$ et $\Vdash_{\mathsf{Ty}(\mathcal{A}')}$ sont des relations pleines).

Dans la construction de modèle que nous allons effectuer dans la section suivante, les types imprédicatifs de la sorte Set seront interprétés par les bases de types calculatoires, c'est-à-dire par les bases de types du modèle formées par des cliques purement calculatoires :

Définition 6.1.21 (Base de type calculatoire) — Soit $\mathcal{A}$ un $\mathcal{K}$-espace cohérent. On dit qu'une base de type $X \subset \mathcal{A}$ est une *base de type calculatoire* si X ne contient que des cliques purement calculatoires, c'est-à-dire si $X \subset \|\mathcal{A}\|$.

Remarquons que dans un $\mathcal{K}$-espace cohérent $\mathcal{A}$, les bases de types calculatoires peuvent toutes être vues comme des bases de types de l'espace de calcul $\mathcal{K}$, modulo le plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$. Les bases de types $\mathbf{0}$ et $\mathbf{1}$ sont bien entendu des bases de types calculatoires de $\mathcal{A}$, et conservent le comportement imprédicatif décrit par la proposition 5.1.23.[4] Cependant, la structure calculatoire inhérente à la notion de $\mathcal{K}$-espace cohérent nous permet d'étendre ce comportement imprédicatif à toutes les bases de types calculatoires :

Proposition 6.1.22 (Imprédicativité) — *Soient $\mathcal{A}$ et $\mathcal{B}$ des $\mathcal{K}$-espaces cohérents, X une base de type de $\mathcal{A}$ et $(Y_a)_{a\in X}$ une famille de bases de types de $\mathcal{B}$ indicée par X. Si toutes les bases de types Y_a sont calculatoires lorsque a décrit X, alors les bases de types*

$$\forall(a\in X; Y_a) \qquad \textit{et} \qquad \Pi(a\in X; Y_a)$$

sont des bases de types calculatoires de $\mathcal{A}$ et de $\mathcal{A} \overset{q}{\to}_* \mathcal{B}$ respectivement.

[4] Ce qui nous permettra de continuer à interpréter les types imprédicatifs *propositionnels* de manière classique par les bases de types $\mathbf{0}$ et $\mathbf{1}$.

Preuve. D'après la proposition 5.1.13 adaptée de manière immédiate au cadre des $\mathcal{K}$-espaces cohérents dans le cas du produit dépendant, on a

$$|\forall(a{\in}X;Y_a)| \quad\subset\quad \bigcup_{a\in X}|Y_a| \qquad\text{et}\qquad |\Pi(a{\in}X;Y_a)| \quad\subset\quad \mathfrak{P}(|X|)\times\bigcup_{a\in X}|Y_a|\,.$$

La première inclusion prouve immédiatement que toutes les cliques de $\forall(a{\in}X;Y_a)$ sont des cliques purement calculatoires de $\mathcal{B}$, puisque la réunion $\bigcup|Y_a|$ ne contient par hypothèse que des atomes calculatoires. Pour le produit dépendant, il suffit de remarquer que pour toute clique $f\in\Pi(a{\in}X;Y_a)$ et pour tout atome $(a_0,\beta)\in f$, l'inclusion de droite entraîne que β est un atome calculatoire de $\mathcal{B}$. Mais comme f est un quasi-$\mathcal{K}$-morphisme, a_0 est par conséquent une clique finie de $\|\mathcal{A}\|$, d'où il ressort que (a_0,β) est un atome calculatoire de $\mathcal{A}\overset{q}{\to}_*\mathcal{B}$. □

6.1.7 Opérateurs de produit dépendant et d'intersection

Comme au chapitre précédent, les constructions $\Pi(a{\in}X;Y_a)$ et $\forall(a{\in}X;Y_a)$ définissent des opérateurs Π et $\forall$ dans les espaces de types comme objets de première classe. Ces opérateurs sont définis par :

Définition 6.1.23 (Opérateurs de produit et d'intersection) — Soient $\mathcal{A}$ et $\mathcal{B}$ des $\mathcal{K}$-espaces cohérents, et μ un cardinal régulier infini. Les opérateurs de produit et d'intersection sont les applications

$$\begin{aligned}\Pi \quad &: \quad \mathsf{Ty}_\mu(\mathcal{A})\times\big(\mathcal{A}\overset{\mu}{\to}_*\mathsf{Ty}_\mu(\mathcal{B})\big)\;\to\;\mathsf{Ty}_\mu(\mathcal{A}\overset{\mu}{\to}_*\mathcal{B})\\ \forall \quad &: \quad \mathsf{Ty}_\mu(\mathcal{A})\times\big(\mathcal{A}\overset{\mu}{\to}_*\mathsf{Ty}_\mu(\mathcal{B})\big)\;\to\;\mathsf{Ty}_\mu(\mathcal{B})\end{aligned}$$

définies pour tout $t\in\mathsf{Ty}_\mu(\mathcal{A})$ et tout $f\in\mathcal{A}\overset{\mu}{\to}\mathsf{Ty}_\mu(\mathcal{B})$ par

$$\Pi(t,f)\;=\;\big\{\tau\big(\Pi(a{\in}X;Y_a)\big)\big\}\qquad\text{et}\qquad\forall(t,f)\;=\;\big\{\tau\big(\forall(a{\in}X;Y_a)\big)\big\}$$

s'il existe une base de type $X\subset\mathcal{A}$ et une famille $(Y_a)_{a\in X}$ de bases de types de $\mathcal{B}$ indicée par X telles que

1. $t=\{\tau(X)\}$,
2. $f(a)=\{\tau(Y_a)\}$ pour tout $a\in X$ et $\Pi(a{\in}X;Y_a)$ (resp. $\forall(a{\in}X;Y_a)$) est une base de type μ-finie de $\mathcal{A}\overset{\mu}{\to}_*\mathcal{B}$ (resp. $\mathcal{B}$) ;

et dans tous les autres cas par $\Pi(t,f)\;=\;\forall(t,f)\;=\;\varnothing$.

Proposition 6.1.24 — *Soient $\mathcal{A}$, $\mathcal{B}$ des $\mathcal{K}$-espaces cohérents, et μ un cardinal régulier infini. Les opérateurs*

$$\begin{aligned}\Pi \quad &: \quad \mathsf{Ty}_\mu(\mathcal{A})\;\times\;\big(\mathcal{A}\overset{\mu}{\to}_*\mathsf{Ty}_\mu(\mathcal{B})\big)\;\to\;\mathsf{Ty}_\mu(\mathcal{A}\overset{\mu}{\to}_*\mathcal{B})\\ \forall \quad &: \quad \mathsf{Ty}_\mu(\mathcal{A})\;\times\;\big(\mathcal{A}\overset{\mu}{\to}_*\mathsf{Ty}_\mu(\mathcal{B})\big)\;\to\;\mathsf{Ty}_\mu(\mathcal{B})\end{aligned}$$

définis ci-dessus sont des fonctions binaires μ-stables.

Preuve. La preuve suit le même schéma que la preuve de la proposition 5.1.29. □

Notons que nous venons seulement de prouver que les fonctions Π et $\forall$ sont μ-stables, mais pas encore qu'il s'agit de $(\mu, \mathcal{K})$-morphismes. En identifiant les opérateurs Π et $\forall$ avec leurs versions curryfiées, on a donc :

$$\begin{array}{lcl} \Pi & \in & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu} \big(\mathcal{A} \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{B})\big) \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}) \\ \forall & \in & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu} \big(\mathcal{A} \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{B})\big) \xrightarrow{\mu} \mathsf{Ty}_\mu(\mathcal{B}) \end{array}$$

Le fait que Π et $\forall$ sont également des $(\mu, \mathcal{K})$-morphismes résulte du lemme suivant :

Lemme 6.1.25 — *Soient $\mathcal{A}$ et $\mathcal{B}$ deux $\mathcal{K}$-espaces cohérents, et μ un cardinal régulier infini. Si $\mathcal{B}$ n'a aucun contenu calculatoire, c'est-à-dire $\|\mathcal{B}\| = \mathsf{0}$, alors toutes les fonctions μ-stables de $\mathcal{A}$ dans $\mathcal{B}$ sont des $(\mu, \mathcal{K})$-morphismes, c'est-à-dire :*

$$\mathcal{A} \xrightarrow{\mu}_* \mathcal{B} \quad = \quad \mathcal{A} \xrightarrow{\mu} \mathcal{B}.$$

De plus, on a $\big\|\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}\big\| = \mathsf{0}$.

Preuve. Conséquence immédiate de la proposition 6.1.6. □

Comme les espaces $\mathsf{Ty}_\mu(\mathcal{A})$ et $\mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu}_* \mathcal{B})$ n'ont aucun contenu calculatoire, on a par conséquent

$$\begin{array}{lcl} \Pi & \in & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu}_* \big(\mathcal{A} \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{B})\big) \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}) \\ \forall & \in & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu}_* \big(\mathcal{A} \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{B})\big) \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{B}) \end{array}$$

en vertu du lemme précédent.

6.1.8 $\mathcal{K}$-espaces cohérents locatifs

Dans l'introduction que nous venons d'effectuer, nous avons donné une définition "catégorique" [5] de la notion de $\mathcal{K}$-espace cohérent, en ce sens que cette définition sépare nettement l'espace de calcul $\mathcal{K}$ d'un côté, et l'espace cohérent $\mathcal{A}$ muni de son sous-espace $\|\mathcal{A}\|$ de l'autre, le pont étant réalisé par le plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{A}$.

Bien entendu, la notion de $\mathcal{K}$-espace cohérent repose déjà sur une approche plus concrète de la réalisabilité que l'approche standard basée sur les ω-sets puisqu'ici, la relation de réalisabilité relie deux objets de même nature (c'est-à-dire des cliques), et que la notion de *contenu* calculatoire est interprétée de la manière la plus naïve qui soit, c'est-à-dire par une notion d'inclusion entre deux cliques.[6] Cependant, le fait d'être un $\mathcal{K}$-espace cohérent n'est pas une propriété intrinsèque d'un espace cohérent $\mathcal{A}$, mais fait intervenir une structuration auxiliaire donnée par le sous-espace $\|\mathcal{A}\| \Subset \mathcal{A}$ et le plongement rigide $\phi_{\mathcal{A}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$.

Dans ce paragraphe, nous nous proposons de suivre une approche purement locative, qui consiste essentiellement à remplacer le plongement rigide $\phi_{\mathcal{K}} : \|\mathcal{A}\| \rightarrowtail \mathcal{K}$ par une inclusion rigide $\|\mathcal{A}\| : \mathcal{A} \Subset \mathcal{K}$ et à caractériser le sous-espace $\|\mathcal{A}\|$ par une notion ne dépendant que de l'espace cohérent $\mathcal{A}$ lui-même. Bien entendu, ce changement de point de vue a un coût sur le plan technique, car il nécessite des hypothèses supplémentaires sur la structure concrète de l'espace de calcul $\mathcal{K}$. Pour cette raison, nous allons introduire les trois conventions suivantes.

[5] Ou "spirituelle" (*i.e.* "à isomorphisme près") pour reprendre la terminologie de J.-Y. Girard [30]. Selon cette terminologie, l'adjectif "spirituel" s'oppose à l'adjectif "locatif" (d'où le titre de ce paragraphe).

[6] La notion de $\mathcal{K}$-espace cohérent prend l'expression "contenu calculatoire" au pied de la lettre : le contenu calculatoire d'une clique est précisément l'ensemble des atomes calculatoires que cette clique contient.

Convention 6.1.26 (Atomes fonctionnels de $\mathcal{K}$) — Dans l'espace de calcul $\mathcal{K}$, les seuls atomes de la forme (x, y) (*i.e.* les couples) sont issus de l'espace des fonctions stables $\mathcal{K} \to \mathcal{K}$, qui est par hypothèse rigidement inclus dans $\mathcal{K}$. Formellement, nous supposerons donc que :

$$\forall\, x, y \qquad (x, y) \in |\mathcal{K}| \quad \Rightarrow \quad x \in \mathcal{K}_{\mathbf{fin}} \;\text{ et }\; y \in |\mathcal{K}|\,.$$

Autrement dit, la trame $|\mathcal{K}|$ ne contient pas d'autres couples que ceux qui proviennent du sous-ensemble $|\mathcal{K} \to \mathcal{K}| = (\mathcal{K}_{\mathbf{fin}} \times |\mathcal{K}|) \subset |\mathcal{K}|$. Cette convention nous permettra de faire en sorte que la définition locative des $\mathcal{K}$-espaces cohérents se transporte automatiquement lors du passage à l'espace des $(\mu, \mathcal{K})$-morphismes. La seconde convention concerne le codage du produit direct :

Convention 6.1.27 (Produit direct) — Dans tout ce qui suit, on suppose que la trame du produit direct (délocalisé) de deux espaces cohérents $\mathcal{A}$ et $\mathcal{B}$ est donnée par

$$|\mathcal{A} \,\&\, \mathcal{B}| \quad = \quad \{1_*\} \times |\mathcal{A}| \;\cup\; \{2_*\} \times |\mathcal{B}|\,,$$

où 1_* et 2_* sont les cliques de $\mathcal{K}$ introduites au paragraphe 6.1.4.

La dernière convention que nous allons adopter est liée au codage des types par des atomes de la forme $\tau(X)$, et exprime que de tels atomes non-calculatoires ne peuvent jamais être présents dans l'espace de calcul $\mathcal{K}$.

Convention 6.1.28 (Absence d'atomes typiques) — La trame de $\mathcal{K}$ ne contient aucun atome de la forme $\tau(X)$. Formellement, nous supposerons donc que

$$\forall\, X \qquad \tau(X) \notin |\mathcal{K}|\,.$$

Bien entendu, cette dernière convention est liée au codage par lequel on a représenté l'atome $\tau(X)$ en théorie des ensembles. Dans toute la seconde partie de ce chapitre, nous supposerons que l'espace $\mathcal{K}$ respecte ces trois conventions, grâce auxquelles nous pourrons nous permettre de ne travailler qu'avec des $\mathcal{K}$-espaces cohérents locatifs.

Définition 6.1.29 ($\mathcal{K}$-espace cohérent locatif) — Un *$\mathcal{K}$-espace cohérent locatif* est un espace cohérent $\mathcal{A}$ tel que les relations de cohérence issues de $\mathcal{A}$ et de $\mathcal{K}$ coïncident sur l'intersection des trames $|\mathcal{A}| \cap |\mathcal{K}|$.

Notons que cette définition repose sur les propriétés intrinsèques de l'ensemble $\mathcal{A}$, et ne dépend plus de la présence de structures auxiliaires. De manière équivalente, un espace cohérent $\mathcal{A}$ est un $\mathcal{K}$-espace cohérent locatif si et seulement si la trame de l'intersection des espaces $\mathcal{A}$ et $\mathcal{K}$[7] est égale à l'intersection des trames de ces deux espaces :

$$\mathcal{A} \;\; \mathcal{K}\text{-espace cohérent locatif} \qquad \Leftrightarrow \qquad |\mathcal{A} \;\cap\; \mathcal{K}| \quad = \quad |\mathcal{A}| \;\cap\; |\mathcal{K}|\,.$$

Intuitivement, la formule ci-dessus exprime que les espaces cohérents $\mathcal{A}$ et $\mathcal{K}$ "interfèrent" d'une manière qui est compatible avec leur structure d'espace cohérent.

Si $\mathcal{A}$ est un $\mathcal{K}$-espace cohérent locatif, $\mathcal{A}$ est naturellement muni d'une structure de $\mathcal{K}$-espace cohérent au sens de la définition 6.1.2, en posant $\|\mathcal{A}\| = \mathcal{A} \cap \mathcal{K}$, et en considérant l'inclusion rigide naturelle $\|\mathcal{A}\| = (\mathcal{A} \cap \mathcal{K}) \Subset \mathcal{K}$.

[7] Ce qui a un sens puisque l'intersection de deux espaces cohérents est clairement un espace cohérent.

Proposition 6.1.30 (Propriétés des $\mathcal{K}$-espaces cohérents locatifs) — *Si $\mathcal{A}$ et $\mathcal{B}$ sont des $\mathcal{K}$-espaces cohérents locatifs, alors*

1. *les espaces cohérents $\mathcal{A} \xrightarrow{\mu}_* \mathcal{B}$, $\mathcal{A} \,\&\, \mathcal{B}$ et $\mathsf{Ty}_\mu(\mathcal{A})$ sont également des $\mathcal{K}$-espaces cohérents locatifs, et leur structure de $\mathcal{K}$-espace cohérent induite par la locativité coïncident avec celle qui leur a été conférée par les définitions 6.1.7, 6.1.12 et 6.1.20 respectivement ;*
2. *les assertions $\mathcal{A} \Subset \mathcal{B}$ et $\mathcal{A} \Subset_* \mathcal{B}$ sont logiquement équivalentes, d'où il ressort que la colimite d'une famille croissante de $\mathcal{K}$-espaces cohérents locatifs $(\mathcal{A}_i)_{i\in I}$ indicée par un ensemble dirigé I est un $\mathcal{K}$-espace cohérent locatif, dont la structure de $\mathcal{K}$-espace cohérent coïncide avec celle que lui confère la définition 6.1.15.*

Dans un $\mathcal{K}$-espace cohérent locatif $\mathcal{A}$, le contenu calculatoire d'une clique $a \in \mathcal{A}$ est donné par

$$\|a\| \quad = \quad a \cap |\mathcal{K}|\,.$$

Notons que le membre droit de l'égalité ci-dessus ne dépend pas de l'espace cohérent $\mathcal{A}$, puisque dans les $\mathcal{K}$-espaces cohérents locatifs la notion de contenu calculatoire est une notion absolue, et non une notion relative à un $\mathcal{K}$-espace cohérent donné. En revanche, la relation de réalisabilité sur $\mathcal{A}$ est caractérisée par une formule

$$k \Vdash_{\mathcal{A}} a \qquad \Leftrightarrow \qquad a \cap |\mathcal{K}| \;=\; k \cap |\mathcal{A}|$$

dont le second membre dépend du $\mathcal{K}$-espace cohérent $\mathcal{A}$ (*i.e.* la relation de réalisabilité est toujours relative au $\mathcal{K}$-espace cohérent dans lequel on se place).

6.2 Le modèle intuitionniste

Dans toute cette section, nous travaillerons dans $\mathrm{ZF} + \mathrm{AI}^\omega$ et nous supposerons que :

- $(\mu_i)_{i>0}$ désigne la suite (croissante) des ω-premiers cardinaux inaccessibles ;
- les atomes typiques $\tau(X)$ sont codés de telle manière que $\tau(X) \neq (x, y)$ pour tous x et y (convention 5.2.1 du chapitre 5) ;
- $\mathcal{K}$ est un espace de calcul satisfaisant les conditions de locativité exprimées par les conventions 6.1.26, 6.1.27 et 6.1.28 ;
- Le cardinal de $\mathcal{K}$ est strictement plus petit que μ_1.[8]

On notera alors μ le plus petit cardinal régulier supérieur à tous les cardinaux μ_i, c'est-à-dire :

$$\mu \quad = \quad \Big(\sup_{i>0} \mu_i\Big)^+ \quad = \quad \aleph_{(\sup \mu_i)+1}.$$

6.2.1 L'équation du modèle intuitionniste

Comme dans le chapitre précédent, le modèle que nous allons construire est le plus petit point fixe d'un endo-foncteur Ψ défini dans une certaine catégorie de plongements rigides (ici, la catégorie des inclusions rigides calculatoires, basée sur la classe des $\mathcal{K}$-espaces cohérents locatifs). Le foncteur Ψ que nous allons considérer dans ce chapitre est défini de la manière suivante :

[8] Cette hypothèse est essentielle pour interpréter la règle de typage $\mathsf{Set} : \mathsf{Type}_1$.

Définition 6.2.1 (Foncteur Ψ) — Soit $\mathcal{A}$ un $\mathcal{K}$-espace cohérent locatif. On note $\Psi(\mathcal{A})$ l'espace cohérent dont la trame est donnée par

$$|\Psi(\mathcal{A})| \quad = \quad |\mathcal{K}| \ \cup \ \left|\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}\right| \ \cup \ \mathsf{Ty}_\mu(\mathcal{A})$$

et dont la relation de cohérence est obtenue par recollement des relations de cohérence issues de $\mathcal{K}$, de $\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}$ et de $\mathsf{Ty}_\mu(\mathcal{A})$.

Notons que dans la définition ci-dessus, les trames $|\mathcal{K}|$ et $\left|\mathsf{Ty}_\mu(\mathcal{A})\right|$ sont disjointes (d'après la convention 6.1.28), de même que les trames $\left|\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}\right|$ et $\left|\mathsf{Ty}_\mu(\mathcal{A})\right|$ (d'après la convention 5.2.1). En revanche, les trames $|\mathcal{K}|$ et $\left|\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}\right|$ peuvent se recouper, mais uniquement sur la trame de l'espace cohérent $\mathcal{K} \to \mathcal{K}$ des fonctions calculatoires (d'après la convention 6.1.26). Dans le cas général, on a donc la relation d'inclusion

$$|\mathcal{K}| \ \cap \ \left|\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}\right| \quad \subset \quad |\mathcal{K} \to \mathcal{K}|\,,$$

et il est facile de vérifier que les relations de cohérence issues des espaces cohérents $\mathcal{K}$, $\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}$ et $\mathcal{K} \to \mathcal{K}$ coïncident sur $|\mathcal{K}| \cap \left|\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}\right|$. Intuitivement, l'espace cohérent $\Psi(\mathcal{A})$ n'est donc rien d'autre que la *somme amalgamée*

$$\Psi(\mathcal{A}) \quad = \quad \Big(\mathcal{K} \underset{\mathcal{K}\to\mathcal{K}}{\oplus} [\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}]\Big) \ \oplus \ \mathsf{Ty}_\mu(\mathcal{A})$$

dans laquelle les deux premières composantes $\mathcal{K}$ et $[\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}]$ ne partagent pas seulement la clique vide (comme c'est le cas pour une somme directe), mais plus généralement toutes les cliques calculatoires de $\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}$, c'est-à-dire :

$$\mathcal{K} \ \cap \ [\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}] \quad = \quad \left\|\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}\right\| \quad = \quad \|\mathcal{A}\| \to \|\mathcal{A}\| \quad \Subset \quad \mathcal{K}.$$

Il est par ailleurs clair que Ψ transforme tout $\mathcal{K}$-espace cohérent locatif $\mathcal{A}$ en un $\mathcal{K}$-espace cohérent locatif $\Psi(\mathcal{A})$ tel que $\|\Psi(\mathcal{A})\| = \mathcal{K}$.

Proposition 6.2.2 (Colimites μ-dirigées) — *Le foncteur Ψ est un foncteur covariant μ-continu dans la catégorie des inclusions rigides calculatoires :*

1. *si $\mathcal{A} \Subset \mathcal{B}$ alors $\Psi(\mathcal{A}) \Subset \Psi(B)$;*
2. *si $(\mathcal{A}_i)_{i\in I}$ est une famille croissante de $\mathcal{K}$-espaces cohérents locatifs indicée par un ensemble μ-dirigé I, alors*

$$\operatorname*{colim}_{i\in I} \Psi(\mathcal{A}_i) \quad = \quad \Psi\Big(\operatorname*{colim}_{i\in I} \mathcal{A}_i\Big)$$

Preuve. Le premier point est immédiat, et le second point résulte de la régularité de μ. □

6.2.2 Résolution de l'équation $\mathcal{A} = \Psi(\mathcal{A})$

Dans ce paragraphe, nous nous proposons de construire la plus petite solution $\mathcal{A}$ de l'équation récursive de domaines

$$\mathcal{A} \quad = \quad \Psi(\mathcal{A}) \quad = \quad \Big(\mathcal{K} \underset{\mathcal{K}\to\mathcal{K}}{\oplus} [\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}]\Big) \ \oplus \ \mathsf{Ty}_\mu(\mathcal{A}). \tag{6.3}$$

Comme dans le chapitre 5, cette plus petite solution est construite à partir d'une suite transfinie croissante de $\mathcal{K}$-espaces cohérents locatifs $(\mathcal{A}_x)_{x<\mu}$ indicée par μ et définie par :

$$\begin{array}{lcll} \mathcal{A}_0 & = & \mathcal{K} & \\ \mathcal{A}_{x+1} & = & \Psi(\mathcal{A}) & \\ \mathcal{A}_x & = & \operatorname*{colim}_{y<x} \mathcal{A}_y & \text{(si } x \text{ est un ordinal limite)} \end{array}$$

D'après la proposition 6.2.2, chaque terme $\mathcal{A}_x$ de cette suite est un $\mathcal{K}$-espace cohérent locatif qui contient rigidement tous les termes précédents, ce qui justifie le passage à la colimite dans le cas où x est un ordinal limite. On pose alors

$$\mathcal{A} \quad = \quad \operatorname*{colim}_{x<\mu} \mathcal{A}_x.$$

Lemme 6.2.3 (Point fixe) — *L'espace $\mathcal{A}$ défini comme la colimite des espaces $\mathcal{A}_x$ $(x < \mu)$ est la plus petite solution de l'équation* (6.3)*.*

Preuve. L'égalité $\mathcal{A} = \Psi(\mathcal{A})$ est une conséquence immédiate de la proposition 6.1.16, et la minimalité du point fixe $\mathcal{A}$ résulte d'un argument analogue à celui que nous avons employé au chapitre précédent dans le cadre du lemme 5.2.2. □

Les deux lemmes suivants sont les analogues des lemmes 5.2.5 et 5.2.7 établis lors de la construction du modèle du chapitre précédent, et nous serviront essentiellement à interpréter la hiérarchie d'univers du calcul implicite :

Lemme 6.2.4 (Cardinalité des espaces $(\mathcal{A}_x)$) — *La suite transfinie $(\mathcal{A}_x)_{x<\mu}$ a les propriétés suivantes :*

1. *pour tout $1 \leq i < \omega$ et pour tout $x < \mu_i$, on a $\overline{\overline{|\mathcal{A}_x|}} < \mu_i$.*
2. *pour tout $1 \leq i < \omega$, on a $\overline{\overline{|\mathcal{A}_{\mu_i}|}} \leq \mu_i$.*

Remarquons que dans la preuve du lemme ci-dessus, l'hypothèse $\overline{\overline{\mathcal{K}}} < \mu_1$ est nécessaire pour pouvoir amorcer la récurrence.

Lemme 6.2.5 — *Pour tout $1 \leq i < \omega$ et pour tout $x < \mu_i$, on a :*

1. $\mathsf{Ty}_\mu(\mathcal{A}_x) \;=\; \mathsf{Ty}(\mathcal{A}_x)$ *($\mathcal{K}$-espace cohérent de toutes les bases de types)*
2. $(\mathcal{A}_x \xrightarrow{\mu}_* \mathcal{A}_x) \;=\; (\mathcal{A}_x \xrightarrow{q}_* \mathcal{A}_x)$ *($\mathcal{K}$-espace cohérent de tous les quasi-$\mathcal{K}$-morphismes)*

Définition 6.2.6 (Rang) — Soit $\alpha \in |\mathcal{A}|$ un atome de $\mathcal{A}$. On appelle *rang de α* le plus petit ordinal $x < \mu$ tel que $\alpha \in |\mathcal{A}_x|$. Le rang de α est noté $\mathsf{rk}(\alpha)$. La notation $\mathsf{rk}(\alpha)$ est étendue à toute clique $a \in \mathcal{A}$ et à toute base de type $X \subset \mathcal{A}$ en posant

$$\mathsf{rk}(a) \quad = \quad \sup_{\alpha \in a} \mathsf{rk}(\alpha) \qquad \text{et} \qquad \mathsf{rk}(X) \quad = \quad \sup_{a \in X} \mathsf{rk}(a).$$

Remarque 6.2.7 — Le rang d'un atome $\alpha \in |\mathcal{A}|$ peut être n'importe quel ordinal successeur strictement plus petit que μ, ou 0 dans le cas où α est un atome calculatoire (*i.e.* $\alpha \in |\mathcal{K}|$). Il est par ailleurs immédiat que le rang d'une clique $a \in \mathcal{A}$ est le plus petit ordinal $x \leq \mu$ tel que $a \in \mathcal{A}_x$ (avec $\mathcal{A}_\mu = \mathcal{A}$) et que le rang d'une base de type $X \subset \mathcal{A}$ est le plus petit ordinal $x \leq \mu$ tel que $X \subset \mathcal{A}_x$. On a par ailleurs les équivalences suivantes :

$$\begin{array}{lcll} \mathsf{rk}(\alpha) & = & 0 \quad \Leftrightarrow \quad \alpha \in |\mathcal{K}| & \text{(atome calculatoire)} \\ \mathsf{rk}(a) & = & 0 \quad \Leftrightarrow \quad a \in \mathcal{K} & \text{(clique purement calculatoire)} \\ \mathsf{rk}(X) & = & 0 \quad \Leftrightarrow \quad X \subset \mathcal{K} & \text{(base de type calculatoire).} \end{array}$$

Lemme 6.2.8 (Structure inductive de $|\mathcal{A}|$) — *Tout atome de $|\mathcal{A}|$ est*

- *soit un atome calculatoire $\kappa \in |\mathcal{K}|$;*
- *soit un atome fonctionnel non-calculatoire, de la forme (a, β), où $a \in \mathcal{A}$, $\overline{\overline{a}} < \mu$ et $\beta \in |\mathcal{A}| \setminus |\mathcal{K}|$;*
- *soit un atome typique, de la forme $\tau(X)$, où $X \subset \mathcal{A}$ est une base de type μ-finie.*

De plus, on a les égalités suivantes :

$$\mathsf{rk}(\kappa) \;=\; 0, \qquad \mathsf{rk}((a,\beta)) \;=\; \max\big(\mathsf{rk}(a), \mathsf{rk}(\beta)\big) + 1 \qquad \textit{et} \qquad \mathsf{rk}\big(\tau(X)\big) \;=\; \mathsf{rk}(X) + 1,$$

l'égalité centrale ne valant que dans la mesure où $\beta \notin |\mathcal{K}|$.

Preuve. Par une récurrence immédiate sur le rang de l'atome considéré. Dans le cas d'un atome fonctionnel $(a_0, \beta) \in |\mathcal{A}|$, on doit cependant distinguer deux cas :

- Soit $\beta \in |\mathcal{K}|$. Dans ce cas on a $a_0 \in \mathcal{K}_{\mathbf{fin}}$, d'où $(a_0, \beta) \in |\mathcal{K}|$ et $\mathsf{rk}((a_0, \beta)) = 0$.
- Soit $\beta \notin |\mathcal{K}|$. Dans ce cas, (a_0, β) est un atome non-calculatoire de $\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}$. L'égalité $\mathsf{rk}((a_0, \beta)) = \max(\mathsf{rk}(a_0), \mathsf{rk}(\beta))$ est alors une conséquence immédiate de la définition de $\mathcal{A}$ et de la notion de rang. □

Lemme 6.2.9 (Atomes non-calculatoires) — *Les atomes non-calculatoires du modèle $\mathcal{A}$ sont les atomes typiques et les atomes de fonctions représentant des prédicats, c'est-à-dire les atomes de la forme*

$$(a_1, (a_2, \ldots, (a_n, \tau(X))) \ldots) \qquad (n \geq 0)$$

où $a_1, \ldots, a_n$ sont des cliques μ-finies arbitraires de $\mathcal{A}$, et X une base de type μ-finie de $\mathcal{A}$.

Preuve. On raisonne par récurrence sur le rang de l'atome considéré. Supposons que $\alpha \in |\mathcal{A}|$ est un atome non-calculatoire. D'après l'équation $\mathcal{A} = \Psi(\mathcal{A})$, l'atome α est :

- soit un atome typique $\alpha = \tau(X)$, auquel cas le résultat recherché est immédiat ;
- soit un atome fonctionnel non-calculatoire $\alpha = (a_0, \beta)$, avec $\beta \notin |\mathcal{K}|$. Comme β est non-calculatoire et $\mathsf{rk}(\beta) < \mathsf{rk}(\alpha) = \max(\mathsf{rk}(a_0), \mathsf{rk}(\beta)) + 1$, l'hypothèse de récurrence s'applique à β. Par conséquent, il existe des cliques μ-finies $a_1, \ldots, a_n \in \mathcal{A}$ $(n \geq 0)$ et une base de type μ-finie $X \subset \mathcal{A}$ telles que $\beta = (a_1, (\ldots, (a_n, \tau(X)) \ldots))$, d'où

$$\alpha \quad = \quad (a_0, \beta) \quad = \quad (a_0, (a_1, \ldots, (a_n, \tau(X)) \ldots)).$$ □

Remarque 6.2.10 — Dans le cas où l'espace de calcul $\mathcal{K}$ est la plus petite solution de l'équation $\mathcal{K} = [\mathcal{K} \to \mathcal{K}] \oplus \mathbb{N}_\perp$ (c'est-à-dire un modèle du λ-calcul avec entiers primitifs), les atomes de $\mathcal{A}$ sont

- soit des atomes calculatoires, de la forme

$$\kappa \;=\; (k_1, (\ldots, (k_n, p)\ldots)) \qquad (n \geq 0),$$

où $k_1, \ldots, k_n$ sont des cliques finies purement calculatoires, et $p \in \mathbb{N}$ un entier naturel quelconque ;

- soit des atomes non-calculatoires, de la forme

$$\alpha \;=\; (a_1, (\ldots, (a_n, \tau(X))\ldots)) \qquad (n \geq 0),$$

où $a_1, \ldots, a_n$ sont des cliques μ-finies de $\mathcal{A}$, et $X \subset \mathcal{A}$ une base de type μ-finie.

D'après la structure de $\mathcal{A}$, il n'existe aucun atome fonctionnel de la forme (a_0, κ) où κ est un atome calculatoire et a_0 une clique non purement calculatoire, puisque les fonctions construisant des objets calculatoires ne peuvent le faire qu'en déstructurant d'autres objets calculatoires. Cette restriction est importante, car c'est elle qui justifie le fait que notre interprétation du polymorphisme (dans Set) est paramétrique, c'est-à-dire qu'une fonction de type $\Pi A : \mathsf{Set} \,.\, T_A$ (où T_A est une famille de types de Set indicée par la variable de type $A : \mathsf{Set}$) ne dépend pas de son premier argument $A : \mathsf{Set}$, qui est purement non-calculatoire.

En revanche, le modèle comporte des atomes de la forme (k, β), où k est une clique purement calculatoire et β un atome non-calculatoire. Autrement dit, il est possible de construire des fonctions déstructurant des objets calculatoires pour construire des objets non-calculatoires (tels que des types par exemple). Ce mécanisme — qui correspond à l'interprétation de la notion d'*élimination forte* [6, 63] dans les $\mathcal{K}$-espaces cohérents — nous permettra en pratique de construire des prédicats qui pourront distinguer deux programmes de même type A, où A est un type de la sorte Set (nous en donnerons un exemple au paragraphe 6.2.5).

6.2.3 Structure de λ-modèle sous-extensionnel

Pour dégager la structure de λ-modèle sous-extensionnel de $\mathcal{A}$, nous ne pouvons pas utiliser la proposition 4.3.20, qui présuppose l'existence d'un plongement rigide de $\mathcal{A} \xrightarrow{\mu} \mathcal{A}$ dans $\mathcal{A}$, et non de $\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}$ dans $\mathcal{A}$ comme c'est ici le cas. Bien entendu, il n'est pas difficile d'adapter la preuve de cette proposition au cadre des $\mathcal{K}$-espaces cohérents.

$\mathcal{K}$-espace cohérent des valuations Rappelons qu'au paragraphe 4.3.3, nous avons identifié l'ensemble $\mathbf{Val}_{\mathcal{A}}$ des valuations sur $\mathcal{A}$ à l'espace des fonctions linéaires de l'espace plat des variables $\mathcal{V}_\perp$ dans $\mathcal{A}$:

$$\mathbf{Val}_{\mathcal{A}} \quad = \quad \mathcal{V}_\perp \multimap \mathcal{A}.$$

On donne alors à cet espace une structure de $\mathcal{K}$-espace cohérent en posant

$$\|\mathbf{Val}_{\mathcal{A}}\| \quad = \quad \mathcal{V}_\perp \multimap \mathcal{K} \quad \Subset \quad \mathcal{V}_\perp \multimap \mathcal{A}$$

et en considérant le plongement rigide $\phi : (\mathcal{V}_\perp \multimap \mathcal{K}) \rightarrowtail \mathcal{K}$ défini par

$$\phi((x, \kappa)) \quad = \quad (x_*, \kappa)$$

où $(x_*)_{x\in\mathcal{V}}$ est une famille de cliques mono-atomiques de $\mathcal{K}$ indicée par l'ensemble (dénombrable) des variables, et dont les termes sont deux à deux incompatibles dans $\mathcal{K}$.[9] On vérifie alors aisément que les fonctions $\mathsf{proj}_x : \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$ (projection) et $\mathsf{extend}_x : \mathbf{Val}_{\mathcal{A}} \,\&\, \mathcal{A} \to \mathbf{Val}_{\mathcal{A}}$ (extension) définies par

$$\mathsf{proj}_x(\rho) \;=\; \rho(x) \qquad \text{et} \qquad \mathsf{extend}_x(\rho, a) \;=\; \rho; x \leftarrow a.$$

sont des $(\mu,\mathcal{K})$-morphismes linéaires dont les traces linéaires sont données par

$$\begin{array}{rcl}
\mathsf{TrLin}(\mathsf{proj_x}) & = & \big\{((x,\alpha),\alpha);\quad \alpha \in |\mathcal{A}|\big\} \\
\mathsf{TrLin}(\mathsf{extend_x}) & = & \big\{\big((1,(y,\alpha)),(y,\alpha)\big);\quad y \neq x,\ \alpha \in |\mathcal{A}|\big\} \cup \\
& & \big\{\big((2,\alpha),(x,\alpha)\big);\quad \alpha \in |\mathcal{A}|\big\}.
\end{array}$$

La fonction d'interprétation Dans le $\mathcal{K}$-espace cohérent $\mathcal{A}$, l'application $(f,a) \mapsto f \cdot a$ est définie par

$$f \cdot a \quad = \quad \big\{\beta \in |\mathcal{A}|\,;\quad \exists a_0 \subset a \quad (a_0,\beta) \in f\big\}$$

pour toutes cliques $f, a \in \mathcal{A}$. Par construction, l'opérateur $(\cdot) : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ est un $(\mu,\mathcal{K})$-morphisme, issu par une rétraction immédiate de la fonction d'application

$$\mathbf{App} \quad : \quad [\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}] \,\&\, \mathcal{A} \xrightarrow{\mu}_* \mathcal{A}$$

donnée par la structure de la catégorie cartésienne fermée des $(\mu,\mathcal{K})$-espaces cohérents.

À chaque terme $M \in \Lambda$, on associe un $(\mu,\mathcal{K})$-morphisme $[\![M]\!] : \mathbf{Val}_{\mathcal{A}} \to \mathcal{A}$ défini par récurrence sur M à l'aide des équations suivantes

$$\begin{array}{rcl}
[\![x]\!](\rho) & = & \rho(x) \\
[\![M\ N]\!](\rho) & = & [\![M]\!](\rho) \cdot [\![N]\!](\rho) \\
[\![\lambda x\,.\,M]\!](\rho) & = & \mathsf{Tr}\big(a_{\in\mathcal{A}} \mapsto [\![M]\!](\rho; x \leftarrow a)\big).
\end{array}$$

Notons que cette définition est correcte d'après le fait que les fonctions de projection et d'extension sont des $(\mu,\mathcal{K})$-morphismes. Par la suite, on notera $[\![M]\!]_\rho = [\![M]\!](\rho)$. Ces définitions étant posées, on vérifie alors aisément que :

Proposition 6.2.11 (λ-modèle sous-extensionnel) — *Le quadruplet $(\mathcal{A},\cdot,[\![]\!],\subset)$ est un λ-modèle sous-extensionnel.*

[9] Cette construction n'est possible que dans la mesure où $\mathcal{K}$ est non nul. Si c'est le cas, la trame de $\mathcal{K}$ est infinie (d'après un raisonnement immédiat basé sur l'inclusion rigide $[\mathcal{K} \to \mathcal{K}] \Subset \mathcal{K}$), ce qui nous permet de considérer une famille d'atomes $(\kappa_x)_{x\in\mathcal{V}}$ qui est injective. On pose alors

$$x_* \quad = \quad \Big\{\big(\{(\{\kappa_x\},\kappa_x)\},\bullet\big)\Big\}$$

pour tout $x \in \mathcal{V}$, où $\bullet$ est un atome de $\mathcal{K}$ fixé une fois pour toutes, et on vérifie immédiatement que x_* et y_* sont incompatibles pour tout couple (x,y) de variables distinctes.

6.2.4 Structure de modèle du calcul implicite

Pour terminer cette construction de modèle, nous devons transformer le λ-modèle sous-extensionnel $(\mathcal{A}, \cdot, [\![]\!], \subset)$ en un modèle du calcul implicite, en le munissant

1. d'un sous-ensemble $\mathcal{T} \subset \mathcal{A}$ dont les éléments sont les types du modèle,
2. d'une fonction de décodage $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{A})$,
3. de deux constantes $\Pi, \forall \in \mathcal{A}$ représentant les opérateurs de produits dépendants et
4. d'une famille de constantes $(\mathsf{u}_i)_{i\in\omega} \in \mathcal{T}^\omega$ représentant la hiérarchie d'univers.

Nous dérogerons cependant au cadre défini au paragraphe 3.3.3 en introduisant une constante supplémentaire u_* destinée à représenter la sorte Set (la sorte Prop sera toujours interprétée par la constante u_0). De cette manière, nous pourrons distinguer dans le modèle les types imprédicatifs propositionnels (les habitants de Prop) des types de données imprédicatifs (les habitants de Set) : les premiers seront interprétés par les bases de types booléennes $\mathbf{0}$ et $\mathbf{1}$, tandis que les seconds seront interprétés par n'importe quel base de type de l'espace de calcul $\mathcal{K}$. De cette manière, nous conserverons le tiers-exclu et l'indiscernabilité des preuves dans Prop, mais dans Set, les principes correspondants (obtenus des premiers par une transposition immédiate) seront invalidés, ainsi que nous le verrons aux paragraphes 6.2.5 et 6.2.6.

Bien entendu, l'introduction d'une constante spécifique pour interpréter Set nécessite d'étendre les axiomes des modèles abstraits du calcul implicite (Fig 3.2). Cette modification concerne les axiomes 6, 7, 8, 9 et 10, qu'il s'agit d'étendre au cas où $i = *$ (en considérant que $* + 1 = 1$ dans le cas des axiomes 7 et 8). De plus, on introduit un onzième axiome (calqué sur l'axiome 10) destiné à rendre compte du comportement imprédicatif de u_*, et dont la formulation est la suivante :

(Axiome 11) *Pour tous $t, u \in \mathcal{T}$ et pour tout $i \in \omega \cup \{*\}$, les relations $t \in \mathsf{El}(\mathsf{u}_i)$ et $ua \in \mathsf{El}(\mathsf{u}_*)$ pour tout $a \in \mathsf{El}(t)$ entraînent que $\Pi tu \in \mathsf{El}(\mathsf{u}_*)$ et $\forall tu \in \mathsf{El}(\mathsf{u}_*)$.*

Enfin, nous devrons également modifier la définition de la fonction d'interprétation étendue[10] en posant $[\![\mathsf{Prop}]\!]_\rho = \mathsf{u}_0$ et $[\![\mathsf{Set}]\!]_\rho = \mathsf{u}_*$ (quelle que soit la valuation ρ). Bien entendu, ceci ne changera rien au propriétés de l'interprétation étendue dégagées au paragraphe 3.3.4, mis à part le fait que les dénotations de Set et Prop sont à présent distinctes.

Interprétation des types L'ensemble $\mathcal{T}$ des types du modèle est donné par

$$\mathcal{T} \quad = \quad \Big\{\{\tau(X)\}; \quad X \text{ base de type } \mu\text{-finie de } \mathcal{A}\Big\}$$

et la fonction de décodage qui lui est associée est définie par

$$\mathsf{El}(t) \quad = \quad \uparrow X \qquad \text{avec} \quad t = \{\tau(X)\}$$

pour tout $t \in \mathcal{T}$.

Lemme 6.2.12 (Validité des axiomes 1–3) — *Le sous-ensemble $\mathcal{T} \subset \mathcal{A}$ et la fonction de décodage $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{A})$ satisfont les axiomes 1, 2 et 3 des modèles abstraits du calcul implicite.*

[10] Rappelons que la fonction d'interprétation donnée avec la structure de λ-modèle sous-extensionnel sur $\mathcal{A}$ n'interprète que les termes du λ-calcul pur. Au paragraphe 3.3.4, nous avons montré que la structure des modèles abstraits du calcul implicite permettait d'étendre cette fonction d'interprétation à tous les termes du calcul implicite, en interprétant Prop et Set par la même constante u_0.

Preuve. Comme $\mathcal{T}$ ne contient que des cliques maximales dans $\mathcal{A}$, les axiomes 1 et 2 sont immédiats. L'axiome 3 découle du fait que $\mathsf{El}(t)$ est par définition un type sémantique de $\mathcal{A}$ pour tout $t \in \mathcal{T}$. □

Interprétation des produits dépendants Au paragraphe 6.1.7, nous avons défini deux $(\mu, \mathcal{K})$-morphismes

$$\begin{array}{lcl} \Pi & \in & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu}_* \left(\mathcal{A} \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{A})\right) \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}) \\ \forall & \in & \mathsf{Ty}_\mu(\mathcal{A}) \xrightarrow{\mu}_* \left(\mathcal{A} \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{A})\right) \xrightarrow{\mu}_* \mathsf{Ty}_\mu(\mathcal{A}) \end{array}$$

calculant respectivement le produit dépendant et l'intersection de bases de types dans les espaces $\mathsf{Ty}_\mu(\mathcal{A} \xrightarrow{\mu}_* \mathcal{A})$ et $\mathsf{Ty}_\mu(\mathcal{A})$. Comme le $\mathcal{K}$-espace cohérent locatif que nous avons construit satisfait les inclusions rigides $[\mathcal{A} \xrightarrow{\mu}_* \mathcal{A}] \Subset \mathcal{A}$ et $\mathsf{Ty}_\mu(\mathcal{A}) \Subset \mathcal{A}$, les deux espaces ci-dessus sont rigidement inclus dans $\mathcal{A}$ d'où il ressort que $\Pi \in \mathcal{A}$ et $\forall \in \mathcal{A}$.

Lemme 6.2.13 (Validité des axiomes 4 et 5) — *Les constantes* Π *et* $\forall$ *satisfont les axiomes 4 et 5 des modèles abstraits du calcul implicite.*

Preuve. Ce résultat découle directement de la définition des cliques Π et $\forall$. □

Interprétation des univers Pour terminer cette structuration du $\mathcal{K}$-espace cohérent $\mathcal{A}$ en modèle du calcul implicite, nous devons à présent définir les constantes de sortes u_i (pour tout $i \in \omega \cup \{*\}$). Ces constantes sont construites à partir des ensembles U_i définis par :

$$\begin{array}{lcll} \mathsf{U}_0 & = & \{\{\tau(\mathbf{0})\};\ \{\tau(\mathbf{1})\}\}; & \\ \mathsf{U}_* & = & \Big\{\{\tau(X)\};\quad X \text{ base de type de } \mathcal{K}\Big\}; & \\ \mathsf{U}_i & = & \Big\{\{\tau(X)\};\quad X \text{ base de type } \mu\text{-finie de } \mathcal{A} \text{ t.q } \mathsf{rk}(X) < \mu_i\Big\} & (i > 0). \end{array}$$

Chacun des ensembles $\mathsf{U}_i \subset \mathcal{A}$ ($i \in \omega \cup \{*\}$) est clairement une base de type, qui est par ailleurs μ-finie d'après le lemme 6.2.4 (pour $i \in \omega$) et l'hypothèse initiale selon laquelle le cardinal de $\mathcal{K}$ est strictement plus petit que μ_1 (pour $i = *$). Pour tout $i \in \omega \cup \{*\}$, on pose $\mathsf{u}_i = \{\tau(\mathsf{U}_i)\}$, et on vérifie que :

Lemme 6.2.14 (Validité des axiomes 6–11) — *Les constantes* u_i *satisfont les axiomes 6 à 10 des modèles abstraits du calcul implicite (quel que soit* $i \in \omega \cup \{*\}$*) ainsi que l'axiome 11 introduit au début de ce paragraphe.*

Preuve. Les axiomes 6, 7 et 8 sont immédiats. L'axiome 9 découle de la proposition 6.1.19, l'axiome 10 résulte de la proposition 5.1.23 adaptée au cadre des $\mathcal{K}$-espaces cohérents, et l'axiome 11 est conséquence de la proposition 6.1.22. □

Proposition 6.2.15 (Validité) — *La fonction d'interprétation des* λ*-termes donnée par la structure de* λ*-modèle sous-extensionnel de* $\mathcal{A}$ *peut être étendue en une fonction d'interpréta-*

tion $[\![]\!] : \Lambda_{\mathrm{CCI}} \times \mathcal{A} \to \mathcal{A}$ *sur l'ensemble des termes du calcul implicite par les équations*

$$\begin{array}{lcllcl}
[\![x]\!]_\rho & = & \rho(x) & [\![\mathsf{Prop}]\!]_\rho & = & \mathsf{u}_0 \\
[\![M\ N]\!]_\rho & = & [\![M]\!]_\rho \cdot [\![N]\!]_\rho & [\![\mathsf{Set}]\!]_\rho & = & \mathsf{u}_* \\
[\![\lambda x\,.\,M]\!]_\rho & = & \mathsf{Tr}\big(a_{\in\mathcal{A}} \mapsto [\![M]\!]_{\rho;x\leftarrow a}\big) & [\![\mathsf{Type}_i]\!]_\rho & = & \mathsf{u}_i \qquad (i>0) \\
[\![\Pi x : T\,.\,M]\!]_\rho & = & \Pi \cdot [\![T]\!]_\rho \cdot [\![\lambda x\,.\,U]\!]_\rho & [\![\forall x : T\,.\,M]\!]_\rho & = & \forall \cdot [\![T]\!]_\rho \cdot [\![\lambda x\,.\,U]\!]_\rho .
\end{array}$$

Cette fonction d'interprétation étendue valide tous les jugements du calcul implicite restreint, c'est-à-dire

$$\Gamma \vdash_r M : T \quad et \quad \rho \in [\![\Gamma]\!] \qquad \Rightarrow \qquad [\![T]\!]_\rho \in \mathcal{T} \quad et \quad [\![M]\!]_\rho \in \mathsf{El}\big([\![T]\!]_\rho\big).$$

Preuve. L'existence de la fonction d'interprétation étendue est immédiate (*cf* Prop. 3.3.12). La validité de l'interprétation vis-à-vis du typage dans CCI^- se prouve par une récurrence sur la dérivation du jugement $\Gamma \vdash_r M : T$, en utilisant les lemmes 6.2.12, 6.2.13 et 6.2.14. Le cas de la règle de $\beta\eta$-conversion se traite en adaptant la preuve de la proposition 3.3.14 au cadre défini dans ce paragraphe (l'adaptation est immédiate). □

6.2.5 Discrimination des programmes dans Set

Au chapitre précédent, nous avons construit un modèle du calcul implicite dans lequel les habitants des types imprédicatifs sont indiscernables (paragraphe 5.2.5). Dans le $\mathcal{K}$-espace cohérent $\mathcal{A}$ que nous venons de construire, les types propositionnels sont toujours interprétés par les bases de types booléennes $\mathbf{0}$ et $\mathbf{1}$, et il est facile de vérifier que la propriété d'indiscernabilité des preuves reste valable dans tous les types de la sorte Prop, c'est-à-dire :

Proposition 6.2.16 (Indiscernabilité des types propositionnels) — *Dans le $\mathcal{K}$-espace cohérent $\mathcal{A}$, la proposition $\forall A : \mathsf{Prop}\,.\,\forall x, y : A\,.\ x =_A y$ est valide, c'est-à-dire :*

$$[\![\forall A : \mathsf{Prop}\,.\,\forall x, y : A\,.\ x =_A y]\!] \quad = \quad \{\tau(\mathbf{1})\}$$

(où $x =_A y$ désigne la proposition $\forall P : A \to \mathsf{Prop}\,.\,P\ x \to P\ y$).

En revanche, ce résultat n'est plus vrai dans la sorte Set, dont les habitants sont maintenant interprétés par toutes les bases de types calculatoires $X \subset \mathcal{K}$, et non plus par les seules bases de types booléennes $\mathbf{0}$ et $\mathbf{1}$. Bien entendu, ceci n'est valable que dans le cas où l'espace de calcul $\mathcal{K}$ est non-nul, puisque dans le cas contraire, $\mathbf{0}$ et $\mathbf{1}$ sont les seules bases de types calculatoires, et les sortes Prop et Set se confondent dans le modèle $\mathcal{A}$.

Dans ce paragraphe, nous allons donc supposer que l'espace de calcul $\mathcal{K}$ est non nul, et montrer que dans le type $\mathsf{bool} : \mathsf{Set}$ défini par $\mathsf{bool} \equiv \forall A : \mathsf{Set}\,.\,A \to A \to A$, les termes $\mathsf{true} \equiv \lambda xy\,.\,x$ et $\mathsf{false} \equiv \lambda xy\,.\,y$ sont différents, ce que nous allons faire en exhibant dans le modèle $\mathcal{A}$ un prédicat $P : \mathsf{bool} \to \mathsf{Prop}$ capable de distinguer true et false.

Proposition 6.2.17 — *Si $|\mathcal{K}| \neq \varnothing$, alors la proposition $\mathsf{true} =_{\mathsf{bool}} \mathsf{false}$ est fausse dans le modèle, c'est-à-dire :*

$$[\![\forall P : \mathsf{bool} \to \mathsf{Prop}\,.\,P\ \mathsf{true} \to P\ \mathsf{false}]\!] \quad = \quad \{\tau(\mathbf{0})\}.$$

Preuve. Comme le terme $\mathsf{bool} \equiv \forall A : \mathsf{Set} \,.\, A \to A \to A$ a pour type Set dans le calcul implicite restreint, il existe une base de type calculatoire $Z \subset \mathcal{K}$ telle que $[\![\mathsf{bool}]\!] = \{\tau(Z)\}$. Par ailleurs, le type sémantique $\uparrow Z \subset \mathcal{A}$ est caractérisé par

$$\uparrow Z \quad = \quad \bigcap_{\substack{X \subset \mathcal{K} \\ \text{base de type}}} \uparrow(X \to X \to X)$$

d'après la définition de la fonction d'interprétation étendue. Considérons à présent un atome calculatoire $\bullet \in |\mathcal{K}|$ quelconque, et notons X la base de type définie par $X = \{\{\bullet\}\}$. Par un calcul élémentaire, on montre que la base de type $X \to X \to X$ est formée par les quatre cliques mono-atomiques $k_1 = \{\kappa_1\}$, $k_2 = \{\kappa_2\}$, $k_3 = \{\kappa_3\}$ et $k_4 = \{\kappa_4\}$ où κ_1, κ_2, κ_3 et κ_4 sont les atomes calculatoires définis par :

$$\begin{array}{rclcrcl} \kappa_1 & = & (\varnothing, (\varnothing, \bullet)), & \qquad & \kappa_2 & = & (\{\bullet\}, (\varnothing, \bullet)), \\ \kappa_3 & = & (\varnothing, (\{\bullet\}, \bullet)), & & \kappa_4 & = & (\{\bullet\}, (\{\bullet\}, \bullet)). \end{array}$$

Considérons la clique $p \in \mathcal{A}$ définie par

$$p \quad = \quad \Big\{(k_1, \tau(\mathbf{0}));\ (k_2, \tau(\mathbf{1}));\ (k_3, \tau(\mathbf{0}));\ (k_4, \tau(\mathbf{1}))\Big\}.$$

Par définition, la clique $p \in \mathcal{A}$ satisfait les équations

$$p \cdot k_1 = \{\tau(\mathbf{0})\}, \qquad p \cdot k_2 = \{\tau(\mathbf{1})\}, \qquad p \cdot k_3 = \{\tau(\mathbf{0})\}, \qquad \text{et} \qquad p \cdot k_4 = \{\tau(\mathbf{1})\},$$

d'où il ressort que $p \in \uparrow((X \to X \to X) \to \mathsf{U}_0)$. Par un argument de sous-typage immédiat dans $\mathcal{A}$, on a donc

$$p \quad \in \quad \uparrow(Z \to \mathsf{U}_0) \quad = \quad \mathsf{El}([\![\mathsf{bool} \to \mathsf{Prop}]\!]),$$

puisque $\uparrow Z \subset \uparrow(X \to X \to X)$. Par ailleurs, nous avons les égalités

$$\begin{array}{rcll} p \cdot [\![\mathsf{true}]\!] & = & \{\tau(\mathbf{1})\} & \quad (\text{puisque } k_2 \subset [\![\lambda xy \,.\, x]\!]) \\ p \cdot [\![\mathsf{false}]\!] & = & \{\tau(\mathbf{0})\} & \quad (\text{puisque } k_3 \subset [\![\lambda xy \,.\, y]\!]) \end{array}$$

d'où il ressort que

$$[\![P\ \mathsf{true} \to P\ \mathsf{false}]\!]_{(P \leftarrow p)} \quad = \quad \{\tau(\mathbf{1} \to \mathbf{0})\} \quad = \quad \{\tau(\mathbf{0})\},$$

sachant que $p \in \mathsf{El}\big([\![\mathsf{bool} \to \mathsf{Prop}]\!]\big)$. Par passage à l'intersection, il est donc clair que

$$[\![\forall P : \mathsf{bool} \to \mathsf{Prop} \,.\, P\ \mathsf{true} \to P\ \mathsf{false}]\!] \quad = \quad \{\tau(\mathbf{0})\}. \qquad \square$$

6.2.6 Invalidité du tiers-exclu

Dans ce paragraphe, on suppose que l'espace de calcul $\mathcal{K}$ est non nul. Notre but est de montrer que le tiers-exclu transposé à Set est invalide, c'est-à-dire que le type

$$\Pi A : \mathsf{Set} \,.\, ((A \to \mathsf{void}) \to \mathsf{void}) \to A$$

(où $\mathsf{void} \equiv \forall A : \mathsf{Set} \,.\, A$ désigne le codage imprédicatif du type vide dans Set) est dénoté par la clique $\{\tau(\mathbf{0})\}$ dans $\mathcal{A}$.[11] Ce résultat est essentiellement une conséquence des deux lemmes suivants, dont la preuve est immédiate.

[11] Cette formulation est logiquement équivalente (dans le calcul implicite restreint) à la formulation usuelle $\Pi A : \mathsf{Set} \,.\, A + (A \to \mathsf{void})$, où "+" désigne un codage imprédicatif quelconque de l'union disjointe dans Set.

Lemme 6.2.18 — *Soit $X \subset \mathcal{A}$ une base de type μ-finie. La base de type $((X \to \mathbf{0}) \to \mathbf{0}) \to X$ est caractérisée par*

$$((X \to \mathbf{0}) \to \mathbf{0}) \to X \quad = \quad \begin{cases} \mathbf{1} & si\ X = \mathbf{0} \\ \mathbf{1} \to X & si\ X \neq \mathbf{0} \end{cases}$$

Lemme 6.2.19 — *Soit $X \subset \mathcal{A}$ une base de type μ-finie. La base de type $\mathbf{1} \to X$ est l'ensemble des traces des fonctions constantes par vocation à valeurs dans X :*

$$f \ \in \ (\mathbf{1} \to X) \qquad \Leftrightarrow \qquad \exists b \in X \quad f \ = \ \big\{(\varnothing, \beta); \quad \beta \in b\big\}.$$

Proposition 6.2.20 (Invalidité du tiers-exclu) — *Dans le modèle $\mathcal{A}$, la proposition*

$$\Pi A : \mathsf{Set}\,.\,((A \to \mathsf{void}) \to \mathsf{void}) \to A$$

est dénotée par la clique $\{\tau(\mathbf{0})\}$.

Preuve. Comme le terme $\Pi A : \mathsf{Set}\,.\,((A \to \mathsf{void}) \to \mathsf{void}) \to A$ est de type Set dans le calcul implicite restreint, sa dénotation est de la forme $\{\tau(Z)\}$, où $Z \subset \mathcal{K}$ est une base de type calculatoire. Supposons qu'il existe un élément $f \in \uparrow Z$. D'après la définition de la fonction d'interprétation, on a $f \cdot \{\tau(X)\} \in \uparrow\big(((X \to \mathbf{0}) \to \mathbf{0}) \to X\big)$ pour toute base de type calculatoire $X \subset \mathcal{K}$, ce qui équivaut à

$$\forall\, X \subset \mathcal{K} \text{ base de type} \qquad X \neq \mathbf{0} \quad \Rightarrow \quad f \cdot \{\tau(X)\} \ \in \ \uparrow(1 \to X)$$

d'après le lemme 6.2.18. Considérons deux atomes calculatoires κ_1 et κ_2 incohérents entre eux (il en existe car $|\mathcal{K}| \neq \varnothing$ et $[\mathcal{K} \to \mathcal{K}] \Subset \mathcal{K}$), et formons les bases de types calculatoires $X_1 = \{\{\kappa_1\}\}$ et $X_2 = \{\{\kappa_2\}\}$. D'après ce qui précède, nous avons donc

$$f \cdot \{\tau(X_1)\} \ \in \ \uparrow(\mathbf{1} \to X_1) \qquad \text{et} \qquad f \cdot \{\tau(X_2)\} \ \in \ \uparrow(\mathbf{1} \to X_2),$$

d'où il ressort d'après le lemme 6.2.19 que

$$\lfloor f \cdot \{\tau(X_1)\}\rfloor_{\mathbf{1} \to X_1} \ = \ \{(\varnothing, \kappa_1)\} \qquad \text{et} \qquad \lfloor f \cdot \{\tau(X_2)\}\rfloor_{\mathbf{1} \to X_2} \ = \ \{(\varnothing, \kappa_2)\},$$

puisque les cliques $\{(\varnothing, \kappa_1)\}$ et $\{(\varnothing, \kappa_1)\}$ sont les seules fonctions constantes par vocation à valeurs dans X_1 et X_2 respectivement. Puisque $(\varnothing, \kappa_1) \in f \cdot \{\tau(X_1)\}$ et $(\varnothing, \kappa_2) \in f \cdot \{\tau(X_2)\}$, il existe (d'après la définition de l'application) des cliques $a_1 \subset \{\tau(X_1)\}$ et $a_2 \subset \{\tau(X_2)\}$ telles que $(a_1, (\varnothing, \kappa_1)) \in f$ et $(a_2, (\varnothing, \kappa_2)) \in f$. C'est ici qu'on utilise la forme particulière des atomes fonctionnels du modèle, en remarquant que puisque $(\varnothing, \kappa_1)$ et $(\varnothing, \kappa_2)$ sont des atomes calculatoires, les cliques a_1 et a_2 qui les produisent doivent être purement calculatoires, d'où il ressort que $a_1 = a_2 = \varnothing$ (les atomes $\tau(X_1)$ et $\tau(X_2)$ susceptibles de former les cliques a_1 et a_2 étant non-calculatoires). Par conséquent, nous avons donc

$$(\varnothing, (\varnothing, \kappa_1)) \ \in \ f \qquad \text{et} \qquad (\varnothing, (\varnothing, \kappa_2)) \ \in \ f,$$

ce qui est absurde puisque ces deux atomes sont manifestement incohérents. L'hypothèse $f \in \uparrow Z$ est donc absurde, d'où il ressort que $Z = \mathbf{0}$. □

Chapitre 7

Le théorème de normalisation forte

Ce chapitre est consacré à la preuve du théorème de normalisation forte du calcul implicite :

Théorème 7.0.1 (Normalisation forte) — *Tous les termes bien typés du Calcul des Constructions implicite sont fortement normalisables.*

En vertu de la proposition 2.6.3, ce théorème entraîne la cohérence logique du formalisme :

Corollaire 7.0.2 (Cohérence logique) — *Dans le Calcul des Constructions implicite, la proposition* $\forall A : \mathsf{Prop} \,.\, A$ *n'est pas prouvable.*

La preuve que nous allons exposer est essentiellement une preuve sémantique[1] basée sur les différentes notions que nous avons introduites aux chapitres 3, 4 et 5. À ce titre, la méthodologie que nous suivrons est très voisine de [6, 48], à cette différence près que notre interprétation des types est basée sur une version simplifiée de la réalisabilité modifiée [6],[2] et que le modèle de normalisation sous-jacent est construit dans la catégorie des μ-espaces cohérents, et non pas simplement dans la catégorie des ensembles.

Comme pour une preuve de cohérence relative (telle que celle effectuée au chapitre 5 par exemple), la preuve du théorème de normalisation forte du calcul implicite repose sur la construction d'un modèle, appelé *modèle de normalisation*, qui se distingue des modèles que nous avons introduits aux chapitres 5 et 6 sur essentiellement deux points :

1. Dans un modèle de normalisation, chaque type t du modèle doit être accompagné d'une certaine information de *réductibilité*, qui permet de conserver dans toute la structure du modèle l'invariant syntaxique dont nous déduirons la propriété de normalisation forte. Traditionnellement, cette information de réductibilité est donnée par un candidat de réductibilité [31, 63], un ensemble saturé [59, 63] ou une structure de Λ-set saturé [6, 48].

[1] En dépit d'un énoncé syntaxique, le théorème de normalisation forte énonce une propriété sémantique du formalisme, dont la cohérence logique est une conséquence immédiate. Contrairement à la plupart des propriétés syntaxiques du (telles que la préservation du typage par $\beta\eta$-réduction), le théorème de normalisation forte ne peut pas être prouvé dans le formalisme lui-même, sauf si celui-ci est incohérent. Nous verrons au chapitre 9 que la cohérence logique du Calcul des Constructions avec univers (trivialement impliquée par la cohérence logique du calcul implicite) entraîne la cohérence logique de la théorie des ensembles de Zermelo intuitionniste. À moins que cette dernière théorie ne soit incohérente, une preuve de normalisation forte du calcul implicite ne peut donc pas être une preuve élémentaire (*i.e.* que l'on pourrait effectuer dans la théorie des ensembles de Zermelo intuitionniste).

[2] Voir à ce sujet la discussion du paragraphe 7.1.6.

2. Dans un modèle de normalisation, tous les types du modèle doivent être habités afin de permettre l'interprétation de tous les contextes, y compris les contextes incohérents. Dans le calcul implicite, ce point revêt une autre importance, car il fait de tout modèle de normalisation un modèle *non-restreint*, au sens de la définition 3.3.18. D'après la proposition 3.3.21, nous serons donc en mesure d'interpréter toutes les règles de typage du calcul implicite, y compris la règle de renforcement.

Dans la suite de ce chapitre, nous suivrons l'esprit de la preuve de cohérence logique du calcul implicite restreint présentée aux chapitres 3 et 5, en décomposant la preuve de normalisation forte en deux étapes distinctes.

Dans un premier temps, nous introduirons une notion abstraite de *modèle de normalisation*, qui est un raffinement de la notion de modèle abstrait du calcul implicite (Def 3.3.11). Nous verrons qu'un modèle de normalisation est essentiellement un modèle non-restreint (Déf 3.3.18) muni d'une fonction de *réductibilité* $\mathsf{Red} : \mathcal{T} \to \mathbf{SAT}$ associant à chaque code de type $t \in \mathcal{T}$ un ensemble saturé. En nous appuyant sur les propriétés d'une telle structure, nous établirons un théorème abstrait de normalisation forte, qui exprime que s'il existe un modèle de normalisation, alors tous les termes bien typés du calcul implicite sont fortement normalisables. Cette preuve (essentiellement syntaxique) s'effectuera avec un outillage mathématique réduit, comportant notamment la propriété de Church-Rosser pour la $\beta\eta$-réduction, ainsi que les différents résultats établis dans la section 3.3.

La seconde partie de la preuve sera consacrée à la construction d'un modèle de normalisation dans la catégorie des μ-espaces cohérents. D'un point de vue formel, la construction de ce modèle s'effectuera dans $\mathrm{ZF} + \mathrm{AI}^{\omega}$, c'est-à-dire dans la théorie des ensembles étendue par l'axiome de ω-inaccessibilité. En toute rigueur, le théorème de normalisation forte ne vaudra donc que dans cette théorie.

Cependant, nous conjecturons que la preuve de normalisation forte peut être effectuée dans ZFC (voire ZF) sans recourir aux cardinaux inaccessibles, en utilisant des méthodes voisines de celles décrites dans [48], lesquelles permettent de réduire les phénomènes d'explosion combinatoire liées à l'interprétation des produits dépendants dans les univers prédicatifs.

7.1 Une preuve abstraite de normalisation forte

7.1.1 Termes fortement normalisables

Définition 7.1.1 (Séquence de β-réductions) — Soit $M \in \Lambda_{\mathrm{CCI}}$ un terme. On appelle *séquence de β-réductions issue de M* toute suite finie ou infinie de termes $(M_i)_{i\in I}$ indicée par un intervalle non-vide de la forme $I = [0..n]$ $(n \geq 0)$ ou $I = \omega$, telle qu'on ait

$$M_0 = M \qquad \text{et} \qquad \forall\, i \quad M_i \to_\beta M_{i+1}.$$

La borne supérieure de l'intervalle I (qui vaut n si $I = [0..n]$ ou ∞ si $I = \omega$) est appelée *longueur* de la séquence de β-réductions $(M_i)_{i\in I}$.

Définition 7.1.2 (Terme fortement normalisable) — On dit qu'un terme $M \in \Lambda_{\mathrm{CCI}}$ est *fortement normalisable* si toutes les séquences de β-réductions issues de M sont finies.

L'ensemble des termes fortement normalisables est noté SN.[3]

Lemme 7.1.3 — *Si M est fortement normalisable, alors :*

1. *tous les sous-termes de M sont fortement normalisables ;*
2. *tous les termes M' tels que $M \twoheadrightarrow_\beta M'$ sont fortement normalisables.*

Lemme 7.1.4 — *Soient x une variable et $T, U, M, N_1, \ldots, N_n$ des termes. On a les équivalences suivantes :*

1. *$xN_1 \cdots N_n \in SN$ si et seulement si $N_i \in SN$ pour tout $i \in [1..n]$;*
2. *$\lambda x\,.\,M \in SN$ si et seulement si $M \in SN$*
3. *$\Pi x : T\,.\,U \in SN$ si et seulement si $T \in SN$ et $U \in SN$*
4. *$\forall x : T\,.\,U \in SN$ si et seulement si $T \in SN$ et $U \in SN$*

Définition 7.1.5 (β-réduit) — Soit $M \in \Lambda_{\mathrm{CCI}}$ un terme. On appelle *β-réduit* de M tout terme M' tel que $M \to_\beta M'$ (β-réduction en une étape).

Lemme 7.1.6 — *Un terme $M \in \Lambda_{\mathrm{CCI}}$ est fortement normalisable si et seulement si tous ses β-réduits sont des termes fortement normalisables.*

Pour tout terme $M \in \Lambda_{\mathrm{CCI}}$, on note $\varepsilon(M)$ la borne supérieure (finie ou infinie) de l'ensemble des longueurs des séquences finies de β-réductions issues de M. On a l'équivalence suivante :

Lemme 7.1.7 — *Pour qu'un terme $M \in \Lambda_{\mathrm{CCI}}$ soit fortement normalisable, il faut et il suffit que $\varepsilon(M)$ soit fini.*

Preuve. La condition est clairement suffisante. Pour montrer qu'elle est nécessaire, il suffit de considérer l'arbre formé par l'ensemble des séquences finies de β-réductions issues de M. Cet arbre qui est à branchement fini (les termes du calcul n'ayant chacun qu'un nombre fini de β-réduits) et qui ne comporte pas de branche infinie (car M est fortement normalisable par hypothèse) est fini d'après le lemme de Koenig, d'où il résulte que sa profondeur, égale à $\varepsilon(M)$, est finie. □

7.1.2 Ensembles saturés

Nous allons maintenant introduire la notion d'*ensemble saturé* relative au calcul implicite. Cette définition est essentiellement celle de Tait [59], à deux différences près :

1. Dans la définition ci-dessous, nous ne considérons pas seulement les termes du λ-calcul pur, mais tous les termes du calcul implicite. Ceci se traduit essentiellement dans le critère (SAT2), qui doit également prendre en compte les sortes et les produits dépendants (explicites ou implicites).

[3] Rappelons que Λ_{CCI} désigne l'ensemble des termes du calcul implicite, tandis que Λ désigne l'ensemble des termes du λ-calcul pur. En revanche SN (sans indice) désigne l'ensemble des termes fortement normalisables du calcul implicite (et non du λ-calcul pur), de même que **SAT** (sans indice) désignera l'ensemble des ensembles saturés du calcul implicite.

2. En plus des critères (SAT1), (SAT2) et (SAT3) figurant dans la définition de Tait, nous avons ajouté un critère supplémentaire (SAT4) qui stipule que les ensembles saturés sont stables par η-réduction de tête faible. Ce critère nous servira à traiter le cas de la règle (Ext) dans la preuve de la proposition 7.1.19.

Définition 7.1.8 (Ensemble saturé) — Un *ensemble saturé* est un ensemble de termes du calcul implicite $S \subset \Lambda_{\mathrm{CCI}}$ satisfaisant les critères suivants :

- (SAT1) $S \subset SN$
- (SAT2) si $M, N_1, \ldots, N_n \in SN$ et si M est une variable, une sorte, un produit explicite ou un produit implicite, alors $MN_1 \cdots N_n \in S$
- (SAT3) si $M\{x := N\}N_1 \cdots N_n \in S$ et $N \in SN$, alors $(\lambda x\,.\,M)NN_1 \cdots N_n \in S$.
- (SAT4) si $(\lambda x\,.\,M\ x)N_1 \cdots N_n \in S$ et $x \notin FV(M)$, alors $MN_1 \cdots N_n \in S$.

L'ensemble de tous les ensembles saturés est noté **SAT**.

Remarquons que cette définition est une définition non-typée, qui ne tient compte que du comportement calculatoire des termes indépendamment de leur signification. En particulier, le critère (SAT2) impose la présence de termes mal typés de la forme $(\Pi x : T\,.\,U)N_1 \cdots N_n$ dans tous les ensembles saturés.

Notre premier travail est de vérifier que la définition précédente a un sens, en montrant qu'il existe au moins un ensemble saturé.

Proposition 7.1.9 (Existence d'un ensemble saturé) — *L'ensemble des termes fortement normalisables SN est un ensemble saturé.*

Preuve. Les critères (SAT1) et (SAT2) sont immédiats.

(SAT3) On démontre par récurrence sur l'entier $\varepsilon(M) + \varepsilon(N) + \varepsilon(N_1) + \cdots + \varepsilon(N_n)$ que $M\{x := N\}N_1 \cdots N_n \in SN$ entraîne $(\lambda x\,.\,M)NN_1 \cdots N_n \in SN$ $(M, N, N_1, \ldots, N_n \in SN)$. Supposons $M\{x := N\}N_1 \cdots N_n \in SN$. On distingue deux cas.

1. $\varepsilon(M)+\varepsilon(N)+\varepsilon(N_1)+\cdots+\varepsilon(N_n) = 0$. Dans ce cas, les termes $M, N, N_1, \ldots, N_n$ sont en forme normale, et le terme $(\lambda x\,.\,M)NN_1 \cdots N_n$ admet pour unique β-réduit le terme $M\{x := N\}N_1 \cdots N_n$ qui est fortement normalisable par hypothèse. Par conséquent, le terme $(\lambda x\,.\,M)NN_1 \ldots N_n$ est fortement normalisable.
2. $\varepsilon(M) + \varepsilon(N) + \varepsilon(N_1) + \cdots + \varepsilon(N_n) > 0$. Dans ce cas, considérons un β-réduit M_0 du terme $(\lambda x\,.\,M)NN_1 \cdots N_n$. Le terme M_0 est de l'une des quatre formes suivantes :
   - $M_0 = M\{x := N\}N_1 \cdots N_n$, qui est fortement normalisable par hypothèse.
   - $M_0 = (\lambda x\,.\,M')NN_1 \ldots N_n$ où M' est un β-réduit de M. On a alors

     $$M\{x := N\}N_1 \cdots N_n \ \to_\beta \ M'\{x := N\}N_1 \cdots N_n,$$

     ce qui entraîne que $M'\{x := N\}N_1 \ldots N_n$ est fortement normalisable. Comme

     $$\varepsilon(M') + \varepsilon(N) + \varepsilon(N_1) + \ldots + \varepsilon(N_n) \ < \ \varepsilon(M) + \varepsilon(N) + \varepsilon(N_1) + \ldots \varepsilon(N_n),$$

     l'hypothèse de récurrence s'applique, d'où il resssort que le terme

     $$M_0 \ = \ (\lambda x\,.\,M')NN_1 \cdots N_n$$

     est fortement normalisable.

- $M_0 = (\lambda x\,.\,M)N'N_1 \ldots N_n$ où N' est un β-réduit de N. Dans ce cas nous avons
$$M\{x := N\}N_1 \cdots N_n \twoheadrightarrow_\beta M\{x := N'\}N_1 \cdots N_n,$$
ce qui prouve que $M\{x := N'\}N_1 \cdots N_n$ est fortement normalisable. Et comme
$$\varepsilon(M) + \varepsilon(N') + \varepsilon(N_1) + \ldots + \varepsilon(N_n) \;<\; \varepsilon(M) + \varepsilon(N) + \varepsilon(N_1) + \ldots \varepsilon(N_n),$$
l'hypothèse de récurrence s'applique, d'où il ressort que
$$M_0 \;=\; (\lambda x\,.\,M)N'N_1 \cdots N_n$$
est fortement normalisable.
- $M_0 = (\lambda x\,.\,M)NN'_1 \cdots N'_n$ où N'_i est un β-réduit de N_i pour un certain indice i tandis que $N'_j = N_j$ pour $j \neq i$. Ce cas se traite de la même façon que le cas précédent.

Nous venons de montrer que tous les β-réduits du terme $(\lambda x\,.\,M)NN_1 \cdots N_n$ sont fortement normalisables, ce qui entraîne que ce terme est fortement normalisable lui aussi.

(SAT4) Supposons $(\lambda x\,.\,M\ x)N_1 \cdots N_n \in SN$, avec $x \notin FV(M)$. On distingue deux cas :

1. $n = 0$. Comme $\lambda x\,.\,M\ x$ est fortement normalisable, son sous-terme M est fortement normalisable également.
2. $n > 0$. Comme $(\lambda x\,.\,M\ x)N_1N_2 \cdots N_n \to_\beta MN_1N_2 \cdots N_n$, le terme $MN_1N_2 \cdots N_n$ est fortement normalisable. □

Définition 7.1.10 (Flèche) — Soient $S, R \subset \Lambda_{\mathrm{CCI}}$ des ensembles de termes. On note $S \to R$ l'ensemble de termes défini par
$$S \to R \quad = \quad \big\{M \in \Lambda_{\mathrm{CCI}};\qquad \forall\, N \in \Lambda_{\mathrm{CCI}} \qquad N \in S \quad \Rightarrow \quad MN \in R\big\}$$

Proposition 7.1.11 (Clôture par flèche et intersection) — *L'ensemble* **SAT** *est clos par flèche et intersection non-vide arbitraire :*

1. *si* $S, T \in \mathbf{SAT}$*, alors* $S \to T \in \mathbf{SAT}$.
2. *si* $(S_i)_{i\in I} \in \mathbf{SAT}^I$ *et* $I \neq \varnothing$*, alors* $\bigcap_{i\in I} S_i \in \mathbf{SAT}$

Preuve. Le cas de l'intersection est immédiat. Notons cependant que l'hypothèse selon laquelle l'ensemble d'indices I est non-vide est cruciale pour montrer que l'intersection de la famille d'ensembles saturés $(S_i)_{i\in I}$ est encore un sous-ensemble de SN. Pour montrer que **SAT** est clos par l'opérateur flèche, considérons deux ensembles saturés S et T.

- (SAT1) Soient $M \in (S \to T)$ et x une variable quelconque. D'après (SAT2), $x \in S$ d'où $Mx \in T$ d'après la définition de $S \to T$. Par conséquent Mx est fortement normalisable (d'après (SAT1) appliqué à T) d'où il ressort que son sous-terme M l'est également.
- (SAT2) Soient $M, N_1, \ldots, N_n$ $(n \geq 0)$ des termes fortement normalisables, où M est une variable, une sorte ou un produit dépendant (explicite ou implicite). Pour tout $N \in S$, on a $MN_1 \cdots N_nN \in T$ en vertu de (SAT2), puisque $N \in SN$ d'après (SAT1). Par conséquent, $MN_1 \cdots N_n \in (S \to T)$.
- (SAT3) Supposons que $(M\{x := P\})N_1 \cdots N_n \in (S \to T)$, avec $P \in SN$. Pour tout terme $N \in S$, on a $(M\{x := P\})N_1 \cdots N_nN \in T$ (par définition de $S \to T$), d'où il ressort que $(\lambda x\,.\,M)PN_1 \cdots N_nN \in T$ d'après le critère (SAT3). La relation précédente étant vraie pour tout $N \in S$, on a donc $(\lambda x\,.\,M)PN_1 \cdots N_n \in (S \to T)$.
- (SAT4) Ce cas se traite de la même manière que pour (SAT3).

7.1.3 Modèles de normalisation

Définition 7.1.12 (Modèle de normalisation) — On appelle *modèle de normalisation du calcul implicite* toute structure de la forme

$$(\mathcal{M}, \cdot, [\![]\!], \leq, \mathcal{T}, \mathsf{El}, \mathsf{Red}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega})$$

où

- $(\mathcal{M}, \cdot, [\![]\!], \leq)$ est un λ-modèle sous-extensionnel, et $\mathcal{T}$ un sous-ensemble de $\mathcal{M}$
- El est une application de $\mathcal{T}$ dans $\mathfrak{P}(\mathcal{M}) \setminus \{\varnothing\}$ et Red une application de $\mathcal{T}$ dans $\mathbf{SAT}$
- Π et $\forall$ sont deux éléments de $\mathcal{M}$, et $(\mathsf{u}_i)_{i\in I}$ une suite d'éléments de $\mathcal{T}$

et qui satisfait les axiomes 1 à 10 donnés dans la figure 7.1.

Un modèle de normalisation est essentiellement un modèle du calcul implicite (Déf. 3.3.11) muni d'une fonction de *réductibilité* $\mathsf{Red} : \mathcal{T} \to \mathbf{SAT}$ associant à chaque code de type $t \in \mathcal{T}$ un ensemble saturé $\mathsf{Red}(t) \in \mathbf{SAT}$. Notons que la définition ci-dessus impose à la fonction de décodage El de prendre ses valeurs dans l'ensemble des parties *non-vides* de $\mathcal{M}$. Par conséquent, le modèle abstrait du calcul implicite sous-jacent est un modèle *non-restreint*, au sens de la définition 3.3.18. En pratique, un tel modèle validera donc tous les jugements du calcul implicite (Prop. 3.3.21), y compris ceux qui font usage de la règle de renforcement.

Les axiomes 1 à 10 de la figure 7.1 reprennent l'axiomatique des modèles abstraits du calcul implicite introduite au paragraphe 3.3.3 (Déf 3.3.11 et Fig 3.2), et l'étendent de manière à spécifier le comportement de la fonction de réductibilité Red. Cette extension concerne les axiomes 2, 4, 5 et 8. (La définition de $\mathsf{Red}(\Pi tu)$ donnée par l'axiome 4 sera justifiée au paragraphe 7.1.6.)

7.1.4 Propriétés des modèles de normalisation

Dans toute la suite de cette section, on suppose que $(\mathcal{M}, \cdot, [\![]\!], \leq, \mathcal{T}, \mathsf{El}, \mathsf{Red}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega})$ est un modèle de normalisation. Rappelons qu'un tel modèle de normalisation est bâti autour d'un λ-modèle extensionnel $(\mathcal{M}, \cdot, [\![]\!], \leq)$ dont la fonction d'interprétation $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$ n'est définie que sur l'ensemble des termes du λ-calcul pur. Cependant, nous avons montré au paragraphe 3.3.4 (Prop. 3.3.12) que cette fonction d'interprétation pouvait être étendue à tous les termes du calcul implicite de manière à ce que :

1. $[\![x]\!]_\rho \;=\; \rho(x)$;
2. $[\![\mathsf{Prop}]\!]_\rho \;=\; [\![\mathsf{Set}]\!]_\rho \;=\; \mathsf{u}_0$
3. $[\![\mathsf{Type}_i]\!]_\rho \;=\; \mathsf{u}_i \quad (i > 0)$
4. $[\![M\ N]\!]_\rho \;=\; [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
5. $[\![\lambda x\,.\,M]\!]_\rho \cdot a \;=\; [\![M]\!]_{\rho;x\leftarrow a}$
6. $[\![\Pi x : T\,.\,U]\!]_\rho \;=\; (\Pi \cdot [\![T]\!]_\rho) \cdot [\![\lambda x\,.\,U]\!]_\rho$
7. $[\![\forall x : T\,.\,U]\!]_\rho \;=\; (\forall \cdot [\![T]\!]_\rho) \cdot [\![\lambda x\,.\,U]\!]_\rho$
8. si $[\![M]\!]_{\rho;x\leftarrow a} = [\![M']\!]_{\rho';x\leftarrow a}$ pour tout $a \in \mathcal{M}$, alors $[\![\lambda x\,.\,M]\!]_\rho = [\![\lambda x\,.\,M']\!]_{\rho'}$.
9. $[\![\lambda f x\,.\,f\ x]\!]_\rho \;\leq\; [\![\lambda x\,.\,x]\!]_\rho$

Paramètres des modèles de normalisation

- un ensemble $\mathcal{M}$
- une opération binaire $(\cdot) : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
- une fonction $[\![]\!] : \Lambda \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$
- une relation d'ordre $(\leq) \subset \mathcal{M} \times \mathcal{M}$
- un sous-ensemble $\mathcal{T} \subset \mathcal{M}$
- deux fonctions $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{M}) \setminus \{\varnothing\}$ et $\mathsf{Red} : \mathcal{T} \to \mathbf{SAT}$
- deux constantes $\Pi, \forall \in \mathcal{M}$
- une suite de constantes $(\mathsf{u}_i)_{i\in\omega} \in \mathcal{T}^\omega$

Axiomes des λ-modèles sous-extensionnels

a. si $a \leq a'$ et $b \leq b'$, alors $a \cdot b \leq a' \cdot b'$
b. $[\![x]\!]_\rho = \rho(x)$
c. $[\![M\ N]\!]_\rho = [\![M]\!]_\rho \cdot [\![N]\!]_\rho$
d. $[\![\lambda x\,.\,M]\!]_\rho \cdot a = [\![M]\!]_{\rho;x\leftarrow a}$
e. si $[\![M]\!]_{\rho;x\leftarrow a} = [\![M']\!]_{\rho';x\leftarrow a}$ pour tout $a \in \mathcal{M}$, alors $[\![\lambda x\,.\,M]\!]_\rho = [\![\lambda x\,.\,M']\!]_{\rho'}$
f. $[\![\lambda f x\,.\,f\ x]\!] \ \leq\ [\![\lambda x\,.\,x]\!]$

Axiomes de $\mathcal{T}$, El et Red

1. si $t \in \mathcal{T}$ et $t \leq t'$, alors $t' \in \mathcal{T}$
2. si $t \in \mathcal{T}$ et $t \leq t'$, alors $\mathsf{El}(t) = \mathsf{El}(t')$ et $\mathsf{Red}(t) = \mathsf{Red}(t')$
3. si $t \in \mathcal{T}$, $a \in \mathsf{El}(t)$ et $a \leq a'$, alors $a' \in \mathsf{El}(t)$

Axiomes de Π et $\forall$

4. si $\Pi tu \in \mathcal{T}$, alors
   - $t \in \mathcal{T}$ et $ua \in \mathcal{T}$ pour tout $a \in \mathsf{El}(t)$
   - $\mathsf{El}(\Pi tu) = \{f \in \mathcal{M};\quad \forall\, a \in \mathsf{El}(t)\quad fa \in \mathsf{El}(ua)\}$
   - $\mathsf{Red}(\Pi tu) = \mathsf{Red}(t) \to \bigcap_{a\in\mathsf{El}(t)} \mathsf{Red}(ua)$
5. si $\forall tu \in \mathcal{T}$, alors
   - $t \in \mathcal{T}$ et $ua \in \mathcal{T}$ pour tout $a \in \mathsf{El}(t)$
   - $\mathsf{El}(\forall tu) = \{b \in \mathcal{M};\quad \forall\, a \in \mathsf{El}(t)\quad b \in \mathsf{El}(ua)\}$
   - $\mathsf{Red}(\forall tu) = \bigcap_{a\in\mathsf{El}(t)} \mathsf{Red}(ua)$

Axiomes des univers

6. $\mathsf{El}(\mathsf{u}_i) \subset \mathcal{T}$
7. $\mathsf{u}_i \in \mathsf{El}(\mathsf{u}_{i+1})$
8. $\mathsf{El}(\mathsf{u}_i) \subset \mathsf{El}(\mathsf{u}_{i+1})$ et $\mathsf{Red}(\mathsf{u}_i) \subset \mathsf{Red}(\mathsf{u}_{i+1})$
9. si $t \in \mathsf{El}(\mathsf{u}_i)$ et $ua \in \mathsf{El}(\mathsf{u}_i)$ pour tout $a \in \mathsf{El}(t)$, alors $\Pi tu \in \mathsf{El}(\mathsf{u}_i)$ et $\forall tu \in \mathsf{El}(\mathsf{u}_i)$
10. si $t \in \mathsf{El}(\mathsf{u}_i)$ et $ua \in \mathsf{El}(\mathsf{u}_0)$ pour tout $a \in \mathsf{El}(t)$, alors $\Pi tu \in \mathsf{El}(\mathsf{u}_0)$ et $\forall tu \in \mathsf{El}(\mathsf{u}_0)$

FIG. 7.1 – Paramètres et axiomes des modèles de normalisation

D'après la proposition 3.3.13, cette extension de la fonction d'interprétation satisfait les propriétés suivantes :

Proposition 7.1.13 (Propriétés de l'interprétation étendue) — *La fonction d'interprétation étendue* $[\![]\!] : \Lambda_{\mathrm{CCI}} \times \mathbf{Val}_{\mathcal{M}} \to \mathcal{M}$ *vérifie :*

1. *si* $\rho(x) = \rho'(x)$ *pour tout* $x \in FV(M)$, *alors* $[\![M]\!]_\rho = [\![M]\!]_{\rho'}$
2. *si* $\rho(x) \leq \rho'(x)$ *pour tout* $x \in FV(M)$, *alors* $[\![M]\!]_\rho \leq [\![M]\!]_{\rho'}$
3. $[\![M\{x := N\}]\!]_\rho = [\![M]\!]_{\rho;x \leftarrow [\![N]\!]_\rho}$
4. *si* $M \to_\beta M'$, *alors* $[\![M]\!]_\rho = [\![M']\!]_\rho$.
5. *si* $M \to_\eta M'$, *alors* $[\![M]\!]_\rho \leq [\![M']\!]_\rho$.
6. *si* $M =_{\beta\eta} M'$, *alors il existe un objet* $c \in \mathcal{M}$ *tel que* $[\![M]\!]_\rho \leq c$ *et* $[\![M']\!]_\rho \leq c$

Comme un modèle de normalisation n'est pas nécessairement un λ-modèle extensionnel, les dénotations de termes $\beta\eta$-convertibles M et M' peuvent être différentes. En revanche, nous avons montré dans le cadre des modèles abstraits du calcul implicite (Prop 3.3.14) que si ces dénotations sont des types dans le modèle (*i.e.* $[\![M]\!]_\rho \in \mathcal{T}$ et $[\![M']\!]_\rho \in \mathcal{T}$), alors leur contenu extensionnel est le même, c'est-à-dire $\mathsf{El}([\![M]\!]_\rho) = \mathsf{El}([\![M']\!]_\rho)$.

Cette propriété qui nous avait permis d'interpréter la règle de $\beta\eta$-conversion dans le cadre des modèles abstraits peut être étendue à l'information de réductibilité véhiculée par les types dans les modèles de normalisation :

Proposition 7.1.14 (Interprétation de la règle de $\beta\eta$-conversion) — *Soient* M *et* M' *deux termes* $\beta\eta$*-convertibles. Pour toute valuation* ρ *telle que* $[\![M]\!]_\rho \in \mathcal{T}$ *et* $[\![M']\!]_\rho \in \mathcal{T}$, *on a*

$$\mathsf{El}\big([\![M]\!]_\rho\big) = \mathsf{El}\big([\![M']\!]_\rho\big) \qquad \textit{et} \qquad \mathsf{Red}\big([\![M]\!]_\rho\big) = \mathsf{Red}\big([\![M']\!]_\rho\big).$$

Preuve. Si $M =_{\beta\eta} M'$, alors il existe pour toute valuation ρ un objet $c \in \mathcal{M}$ tel que $[\![M]\!]_\rho \leq c$ et $[\![M']\!]_\rho \leq c$ (point 6 de la proposition précédente). Si $[\![M]\!]_\rho \in \mathcal{T}$ et $[\![M']\!]_\rho \in \mathcal{T}$, on a $c \in \mathcal{T}$ (d'après l'axiome 1), d'où il ressort que

$$\mathsf{El}\big([\![M]\!]_\rho\big) = \mathsf{El}(c) = \mathsf{El}\big([\![M']\!]_\rho\big) \qquad \text{et} \qquad \mathsf{Red}\big([\![M]\!]_\rho\big) = \mathsf{Red}(c) = \mathsf{Red}\big([\![M']\!]_\rho\big)$$

en vertu de l'axiome 2. □

Ainsi que nous l'avons déjà signalé plus haut, un modèle de normalisation est un modèle abstrait *non-restreint* du calcul implicite, dans lequel tous les types sont habités. Par conséquent, le résultat de validité non-restreinte (Prop. 3.3.21) démontré au chapitre 3 s'applique également à $\mathcal{M}$:

Proposition 7.1.15 (Validité) — *Soient* Γ *un contexte et* M, T *des termes.*

- *Si* $\Gamma \vdash$ *alors* $[\![\Gamma]\!] \neq \varnothing$.
- *Si* $\Gamma \vdash M : T$, *alors* $[\![\Gamma]\!] \neq \varnothing$ *et pour toute valuation* $\rho \in \mathbf{Val}_{\mathcal{M}}$ *on a*

$$\rho \in [\![\Gamma]\!] \qquad \Rightarrow \qquad [\![T]\!]_\rho \in \mathcal{T} \quad \textit{et} \quad [\![M]\!]_\rho \in \mathsf{El}\big([\![T]\!]_\rho\big).$$

où $\Gamma \vdash$ *et* $\Gamma \vdash M : T$ *désignent des jugements dérivables dans le calcul implicite non-restreint.*

7.1.5 L'invariant de normalisation

Nous allons maintenant porter notre attention sur l'invariant de normalisation, qui exprime la validité de la fonction d'interprétation vis à vis de la fonction de réductibilité Red, de même que la proposition 7.1.15 exprime la validité de l'interprétation vis-à-vis de la fonction de décodage El. Dans le contexte vide, cet invariant de normalisation est que pour tout terme M de type T, le terme M appartient à l'ensemble saturé $\mathsf{Red}(\llbracket T \rrbracket)$ associé à l'interprétation de T. Afin de pouvoir généraliser cet énoncé à n'importe quel contexte, il est nécessaire d'introduire une notion de *substitution*, qui va jouer un rôle analogue à la notion de valuation.

Définition 7.1.16 (Substitution) — On appelle *substitution* toute liste finie de couples (x_i, N_i) où x_i est une variable et N_i un terme du calcul implicite, telle que $x_i \neq x_j$ dès que $i \neq j$. Les substitutions se notent $\sigma = [x_1 := N_1; \ldots; x_n : N_n]$. De plus :

- Pour toute variable x telle que $x = x_i$ pour un certain $i \in [1..n]$, on dit que $\sigma(x)$ est *défini* et on note $\sigma(x) = N_i$. L'ensemble des variables x telles que $\sigma(x)$ est défini est noté $SV(\sigma)$.
- Si σ est une substitution, x une variable et $N \in \Lambda_{\mathrm{CCI}}$ un terme, on note $\sigma; x := N$ la substitution obtenue en ajoutant le couple (x, N) à la liste σ (cette notation n'ayant un sens que si $\sigma(x)$ n'est pas défini).

Définition 7.1.17 (Application de substitution) — Soient $M \in \Lambda_{\mathrm{CCI}}$ un terme et σ une substitution. On désigne par $M[\sigma]$ le terme défini récursivement sur la structure de M par

$$
\begin{array}{lcll}
x[\sigma] & = & \begin{cases} \sigma(x) & \text{si } \sigma(x) \text{ est défini} \\ x & \text{sinon} \end{cases} & \\
s[\sigma] & = & s & (s = \mathsf{Prop},\ \mathsf{Set},\ \mathsf{Type}_i) \\
(\Pi x : T\,.\,U)[\sigma] & = & \Pi z : T\,.\,(U[\sigma; x := z]) & (z \text{ variable fraîche}) \\
(\forall x : T\,.\,U)[\sigma] & = & \forall z : T\,.\,(U[\sigma; x := z]) & (z \text{ variable fraîche}) \\
(\lambda x\,.\,M)[\sigma] & = & \lambda z\,.\,(M[\sigma; x := z]) & (z \text{ variable fraîche}) \\
(M\ N)[\sigma] & = & M[\sigma]\ N[\sigma] &
\end{array}
$$

Définition 7.1.18 (Substitution adaptée) — Soient Γ un contexte, ρ une valuation et σ une substitution. On dit que σ est *adaptée* à ρ dans Γ, ce que l'on note $\sigma \models_\Gamma \rho$, lorsqu'on a

1. $\rho \in \llbracket \Gamma \rrbracket$
2. $SV(\sigma) = DV(\Gamma)$
3. pour tout $(x : T) \in \Gamma$, $\sigma(x) \in \mathsf{Red}(\llbracket T \rrbracket_\rho)$

Proposition 7.1.19 (Invariant de normalisation) — *Soient Γ un contexte, M et T des termes. Si $\Gamma \vdash M : T$, alors pour toute valuation ρ et pour toute substitution σ on a*

$$\sigma \models_\Gamma \rho \quad \Rightarrow \quad M[\sigma] \ \in\ \mathsf{Red}(\llbracket T \rrbracket_\rho).$$

Preuve. Par récurrence sur la dérivation de $\Gamma \vdash M : T$ en distinguant les cas suivant la dernière règle appliquée. Nous noterons (HR) l'hypothèse de récurrence

$$\text{(HR)} \qquad \forall\, \rho, \sigma \qquad \sigma \models_\Gamma \rho \quad \Rightarrow \quad M[\sigma] \in \mathsf{Red}(\llbracket T \rrbracket_\rho)$$

et, dans le cas où la dernière règle appliquée comporte deux prémisses, les hypothèses de récurrence associées à chacune des prémisses seront notées (HR1) et (HR2) respectivement. Dans la disjonction de cas qui suit, nous n'écrirons que des jugements de typage valides, et nous considérerons que l'écriture $\Gamma \vdash M : T$ entraîne implicitement (d'après la proposition 7.1.15) que $[\![T]\!]_\rho \in \mathcal{T}$ et $[\![M]\!]_\rho \in \mathsf{El}([\![T]\!]_\rho)$ pour toute valuation $\rho \in [\![\Gamma]\!]$, sachant que $[\![\Gamma]\!] \neq \varnothing$.

- (Var) Résulte immédiatement de l'hypothèse selon laquelle $\sigma \models_\Gamma \rho$, qui entraîne que $x[\sigma] = \sigma(x) \in \mathsf{Red}([\![T]\!]_\rho)$.
- (Sort) Conséquence immédiate du critère (SAT2), puisque le terme figurant dans le jugement de typage est une sorte.
- (E-Prd) La conclusion de la dérivation, de la forme $\Gamma \vdash \Pi x : T \,.\, U : s_3$, est issue de deux prémisses $\Gamma \vdash T : s_1$ et $\Gamma; [x : T] \vdash U : s_2$, avec la condition de bord $(s_1, s_2, s_3) \in \mathbf{Rule}$.
  Soient ρ une valuation et σ une substitution telles que $\rho \in [\![\Gamma]\!]$ et $\sigma \models_\Gamma \rho$.
  D'après (HR1), nous avons $T[\sigma] \in \mathsf{Red}\big([\![s_1]\!]\big)$, d'où l'on tire que $T[\sigma]$ est fortement normalisable en vertu de (SAT1).
  Comme $[\![T]\!]_\rho \in \mathsf{El}([\![s_1]\!])$ et $[\![s_1]\!] = \mathsf{u}_i$ pour un certain indice $i \geq 0$, on a $[\![T]\!]_\rho \in \mathcal{T}$ d'après l'axiome 6. Soient $a \in \mathsf{El}([\![T]\!]_\rho)$ et z une variable fraîche. On vérifie immédiatement que $(\sigma; x := z) \models_{\Gamma;[x:T]} (\rho; x \leftarrow a)$, d'après (SAT2).
  D'après (HR2), nous avons $U[\sigma; x := z] \in \mathsf{Red}([\![s_2]\!])$, d'où il ressort que $U[\sigma; x := z]$ est fortement normalisable en vertu de (SAT1).
  Comme par définition, $(\Pi x : T \,.\, U)[\sigma] = \Pi z : T[\sigma] \,.\, U[\sigma; x := z]$, le terme $(\Pi x : T \,.\, U)[\sigma]$ est fortement normalisable, d'où $(\Pi x : T \,.\, U)[\sigma] \in \mathsf{Red}([\![T]\!]_\rho)$ d'après (SAT2).
- (I-Prd) Ce cas ce traite de la même manière que pour la règle (E-Prd).
- (Lam) La conclusion de la dérivation, de la forme $\Gamma \vdash \lambda x \,.\, M : \Pi x : T \,.\, U$, provient de deux prémisses $\Gamma; [x : T] \vdash M : U$ et $\Gamma \vdash \Pi x : T \,.\, U : s$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  Le contexte $\Gamma; [x : T]$ étant bien formé, il existe une sorte s telle que $\Gamma \vdash T : s$. D'après la proposition 7.1.15, on a $[\![T]\!]_\rho \in \mathsf{El}([\![s]\!])$, d'où il ressort que $[\![T]\!]_\rho \in \mathcal{T}$.
  Considérons un terme $N \in \mathsf{Red}([\![T]\!]_\rho)$ fixé. Pour tout $a \in \mathsf{El}([\![T]\!]_\rho)$, on a la relation $(\sigma; x := N) \models_{\Gamma;[x:T]} (\rho; x \leftarrow a)$, puisque $N \in \mathsf{Red}([\![T]\!]_\rho) = \mathsf{Red}\big([\![T]\!]_{\rho;x\leftarrow a}\big)$ (cette dernière égalité étant immédiate puisque $x \notin FV(T)$).
  D'après (HR1), on a donc $M[\sigma; x := N] \in \mathsf{Red}\big([\![U]\!]_{(\rho;x\leftarrow a)}\big)$ pour tout $a \in \mathsf{El}([\![T]\!]_\rho)$, d'où

$$M[\sigma; x := N] \quad \in \bigcap_{a \in \mathsf{El}([\![T]\!]_\rho)} \mathsf{Red}\big([\![U]\!]_{(\rho;x\leftarrow a)}\big).$$

  Soit z une variable fraîche. Comme $M[\sigma; x := N] = (M[\sigma; x := z])\{z := N\}$ (puisque la variable z n'apparaît ni dans M ni dans σ), la relation ci-dessus s'écrit également

$$(M[\sigma; x := z])\{z := N\} \quad \in \bigcap_{a \in \mathsf{El}([\![T]\!]_\rho)} \mathsf{Red}\big([\![U]\!]_{(\rho;x\leftarrow a)}\big).$$

  Comme par ailleurs $N \in SN$ d'après (SAT1), nous pouvons appliquer le critère (SAT3) pour obtenir

$$(\lambda z \,.\, M[\sigma; x := z])\ N \quad \in \bigcap_{a \in \mathsf{El}([\![T]\!]_\rho)} \mathsf{Red}\big([\![U]\!]_{(\rho;x\leftarrow a)}\big),$$

et cela pour tout $N \in \mathsf{Red}(\llbracket T \rrbracket_\rho)$, d'où l'on tire que

$$\begin{array}{rcll}(\lambda x\,.\,M)[\sigma] & = & \lambda z\,.\,M[\sigma; x := z] & \in \quad \mathsf{Red}(\llbracket T \rrbracket_\rho) \to \bigcap\limits_{a \in \mathsf{El}(\llbracket T \rrbracket_\rho)} \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow a)}\big) \\ & & & \in \quad \mathsf{Red}\big(\llbracket \Pi x : T\,.\,U \rrbracket_\rho\big),\end{array}$$

l'égalité des deux membres droits ci-dessus résultant de l'axiome 4 et des propriétés définitionnelles de l'interprétation étendue.

- (App) La conclusion de la dérivation, de la forme $\Gamma \vdash M\ N : U\{x := N\}$, provient de deux prémisses $\Gamma \vdash M : \Pi x : T\,.\,U$ et $\Gamma \vdash N : T$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  D'après (HR1) et (HR2), on a

$$M[\sigma] \quad \in \quad \mathsf{Red}\big(\llbracket \Pi x : T\,.\,U \rrbracket_\rho\big) \quad = \quad \mathsf{Red}(\llbracket T \rrbracket_\rho) \to \bigcap_{a \in \mathsf{El}(\llbracket T \rrbracket_\rho)} \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow a)}\big)$$

  et $N[\sigma] \in \mathsf{Red}(\llbracket T \rrbracket_\rho)$. De ces deux relations, il résulte que

$$(M\ N)[\sigma] \quad = \quad M[\sigma]\ N[\sigma] \quad \in \bigcap_{a \in \mathsf{El}(\llbracket T \rrbracket_\rho)} \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow a)}\big)$$

  Puisque $\llbracket N \rrbracket_\rho \in \mathsf{El}(\llbracket T \rrbracket_\rho)$, un argument d'inclusion immédiat nous donne

$$(M\ N)[\sigma] \quad \in \quad \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow \llbracket N \rrbracket_\rho)}\big) \quad = \quad \mathsf{Red}\big(\llbracket U\{x := N\} \rrbracket_\rho\big).$$

- (Gen) La conclusion de la dérivation, de la forme $\Gamma \vdash M : \forall x : T\,.\,U$, provient de deux prémisses $\Gamma; [x : T] \vdash M : U$ et $\Gamma \vdash \forall x : T\,.\,U : s$, avec la condition de bord $x \notin FV(M)$.
  Soient ρ une valuation, σ une substitution telle que $\sigma \models_\Gamma \rho$ et z une variable fraîche.
  Le contexte $\Gamma; [x : T]$ étant bien formé, il existe une sorte s telle que $\Gamma \vdash T : s$. D'après la proposition 7.1.15, on a $\llbracket T \rrbracket_\rho \in \mathsf{El}(\llbracket s \rrbracket)$, d'où il ressort que $\llbracket T \rrbracket_\rho \in \mathcal{T}$.
  Pour tout $a \in \mathsf{El}(\llbracket T \rrbracket_\rho)$, on a la relation $(\sigma; x := z) \models_{\Gamma; [x:T]} (\rho; x \leftarrow a)$ d'après (SAT2).
  D'après (HR1), on a donc $M[\sigma; x := z] \in \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow a)}\big)$ pour tout $a \in \mathsf{El}(\llbracket T \rrbracket_\rho)$, d'où

$$M[\sigma; x := z] \quad \in \bigcap_{a \in \mathsf{El}(\llbracket T \rrbracket_\rho)} \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow a)}\big) \quad = \quad \mathsf{Red}\big(\llbracket \forall x : T\,.\,U \rrbracket_\rho\big),$$

  l'égalité de droite découlant de l'axiome 5 et des propriétés définitionnelles de la fonction d'interprétation étendue.
  Comme par hypothèse, $x \notin FV(M)$, on a $M[\sigma] = M[\sigma; x := z] \in \mathsf{Red}\big(\llbracket \forall x : T\,.\,U \rrbracket_\rho\big)$.
- (Inst) La conclusion de la dérivation, de la forme $\Gamma \vdash M : U\{x := N\}$, provient de deux prémisses $\Gamma \vdash M : \forall x : T\,.\,U$ et $\Gamma \vdash N : T$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  D'après (HR1), nous avons

$$M[\sigma] \quad \in \quad \mathsf{Red}\big(\llbracket \forall x : T\,.\,U \rrbracket_\rho\big) \quad = \quad \bigcap_{a \in \mathsf{El}(\llbracket T \rrbracket_\rho)} \mathsf{Red}\big(\llbracket U \rrbracket_{(\rho; x \leftarrow a)}\big),$$

l'égalité de droite résultant de l'axiome 5 et des propriétés définitionnelles de l'interprétation étendue. Puisque $[\![N]\!]_\rho \in \mathsf{El}([\![T]\!]_\rho)$, un argument d'inclusion immédiat nous permet de conclure que

$$M[\sigma] \quad \in \quad \mathsf{Red}\big([\![U]\!]_{(\rho;x\leftarrow[\![N]\!]_\rho)}\big) \quad = \quad \mathsf{Red}\big([\![U\{x := N\}]\!]_\rho\big).$$

- (Conv) La conclusion de la dérivation, de la forme $\Gamma \vdash M : T'$, est issue de deux prémisses $\Gamma \vdash M : T$ et $\Gamma \vdash T' : s$, avec la condition de bord $T =_{\beta\eta} T'$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  Comme $[\![T]\!]_\rho \in \mathcal{T}$ et $[\![T']\!]_\rho \in \mathcal{T}$, il vient $\mathsf{Red}([\![T]\!]_\rho) = \mathsf{Red}\big([\![T']\!]_\rho\big)$ d'après le lemme 7.1.14, d'où l'on tire que $M[\sigma] \in \mathsf{Red}\big([\![T']\!]_\rho\big)$ d'après (HR1).
- (Cum) La conclusion de la dérivation, de la forme $\Gamma \vdash M : s_2$, est issue d'une prémisse $\Gamma \vdash M : s_1$, avec la condition de bord $s_1 \leq s_2$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  D'après (HR), on a $M[\sigma] \in \mathsf{Red}\big([\![s_1]\!]\big)$. D'après l'axiome 8, on a $\mathsf{Red}\big([\![s_1]\!]\big) \subset \mathsf{Red}\big([\![s_2]\!]\big)$ (puisque $[\![s_1]\!] = \mathsf{u}_i$ et $[\![s_2]\!] = \mathsf{u}_j$ avec $i \leq j$), d'où il résulte que $M[\sigma] \in \mathsf{Red}\big([\![s_2]\!]\big)$.
- (Ext) La conclusion de la dérivation, de la forme $\Gamma \vdash M : T$, provient d'une prémisse $\Gamma \vdash \lambda x\,.\,M\ x : T$, avec la condition de bord $x \notin FV(M)$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  D'après (HR), on a $(\lambda x\,.\,M\ x)[\sigma] \in \mathsf{Red}([\![T]\!]_\rho)$. Comme

  $$(\lambda x\,.\,M\ x)[\sigma] \quad = \quad \lambda z\,.\,M[\sigma; x := z]\ z \quad = \quad \lambda z\,.\,M[\sigma]\ z \qquad (z \text{ variable fraîche})$$

  puisque $x \notin FV(M)$, il vient $M[\sigma] \in \mathsf{Red}([\![T]\!]_\rho)$ d'après (SAT4).
- (Str) La conclusion de la dérivation, de la forme $\Gamma \vdash M : U$, provient d'une prémisse $\Gamma; [x : T] \vdash M : U$, avec les conditions de bord $x \notin FV(M)$ et $x \notin FV(U)$.
  Soient ρ une valuation et σ une substitution telles que $\sigma \models_\Gamma \rho$.
  Le contexte $\Gamma; [x : T]$ étant bien formé, il existe une sorte s telle que $\Gamma \vdash T : s$. D'après la proposition 7.1.15, on a $[\![T]\!]_\rho \in \mathsf{El}([\![s]\!])$, d'où il ressort que $[\![T]\!]_\rho \in \mathcal{T}$.
  Soient $a \in \mathsf{El}([\![T]\!]_\rho)$ et z une variable fraîche. On a alors $(\sigma; x := z) \models_{\Gamma;[x:T]} (\rho; x \leftarrow a)$ d'après (SAT2).
  En appliquant (HR), il vient $M[\sigma; x := z] \in \mathsf{Red}\big([\![U]\!]_{(\rho;x\leftarrow a)}\big)$. Des conditions de bord $x \notin FV(M)$ et $x \notin FV(U)$ on tire que $M[\sigma; x := z] = M[\sigma]$ et $[\![U]\!]_{(\rho;x\leftarrow a)} = [\![U]\!]_\rho$, d'où il résulte que $M[\sigma] \in \mathsf{El}([\![U]\!]_\rho)$. □

Corollaire 7.1.20 (Normalisation abstraite) — *Si le calcul implicite admet un modèle de normalisation, alors tous les termes bien typés du calcul implicite sont fortement normalisables.*

Preuve. Soient $\Gamma \vdash M : T$ un jugement dérivable du calcul implicite, et $\mathcal{M}$ un modèle de normalisation. D'après la proposition 7.1.15, $[\![\Gamma]\!]$ est non-vide. Considérons donc une valuation $\rho \in [\![\Gamma]\!]$ et notons σ la substitution définie par $\sigma = [x_1 := x_1; \ldots; x_n := x_n]$, où $x_1, \ldots, x_n$ sont les variables déclarées dans le contexte Γ. À l'aide du critère (SAT2), il est immédiat que $\sigma \models_\Gamma \rho$. D'après la proposition 7.1.19, on a donc

$$M \ = \ M[\sigma] \ \in \ \mathsf{Red}([\![T]\!]_\rho),$$

d'où il ressort que $M \in SN$ d'après le critère (SAT1). □

7.1.6 Modèles de normalisation et réalisabilité modifiée

Dans la preuve de normalisation abstraite effectuée ci-dessus, l'information de réductibilité est véhiculée par les produits dépendants explicites et implicites à l'aide des axiomes 4 et 5, qui définissent les contenus des ensembles saturés $\mathsf{Red}(\Pi tu)$ et $\mathsf{Red}(\forall tu)$ en fonction des ensembles saturés $\mathsf{Red}(t)$ et $\mathsf{Red}(ua)$ lorsque a décrit $\mathsf{El}(t)$. La formule définissant l'ensemble saturé $\mathsf{Red}(\forall tu)$

$$\text{(Axiome 5)} \qquad \mathsf{Red}(\forall tu) \quad = \quad \bigcap_{a \in \mathsf{El}(t)} \mathsf{Red}(ua)$$

est très naturelle, puisqu'on interprète l'intersection de types par une intersection d'ensembles saturés. En revanche, la formule définissant l'ensemble saturé $\mathsf{Red}(\Pi tu)$

$$\text{(Axiome 4)} \qquad \mathsf{Red}(\Pi tu) \quad = \quad \mathsf{Red}(t) \to \bigcap_{a \in \mathsf{El}(t)} \mathsf{Red}(ua)$$

l'est beaucoup moins, et ne peut être comprise qu'à travers la notion de *réalisabilité modifiée*[4] sur laquelle sont basées les preuves de normalisation forte présentées dans [6, 48].

Dans ce paragraphe, nous nous proposons de montrer que la fonction de réductibilité Red des modèles de normalisation du calcul implicite, lorsqu'elle est associée à la fonction de décodage El, induit une forme simplifiée de réalisabilité modifiée dans le modèle $\mathcal{M}$, que nous qualifierons ici de *réalisabilité modifiée non-dépendante*.

Pour cela, considérons un modèle de normalisation $\mathcal{M}$ et un code de type $t \in \mathcal{T}$. L'ensemble des objets de type t dans le modèle est donné par $\mathsf{El}(t)$. À l'aide de la fonction Red, on définit naturellement une notion de réalisabilité modifiée $M \models_t a$ sur $\mathsf{El}(t)$ en posant

$$M \models_t a \qquad \equiv \qquad M \in \mathsf{Red}(t)$$

pour tout $M \in \Lambda_{\mathrm{CCI}}$ et pour tout $a \in \mathsf{El}(t)$. Cette relation de réalisabilité est *non-dépendante* en ce sens que la relation $M \models_t a$ ne dépend pas de l'objet $a \in \mathsf{El}(t)$, mais uniquement du terme M. Autrement dit, la relation $\models_t$ n'est pas un sous-ensemble arbitraire du produit cartésien $\Lambda_{\mathrm{CCI}} \times \mathsf{El}(t)$, mais un produit cartésien

$$(\models_t) \quad = \quad \mathsf{Red}(t) \times \mathsf{El}(t)$$

qui exprime l'absence de dépendance entre l'argument "syntaxique" et l'argument "sémantique". En dépit de la simplicité de sa définition, cette relation de réalisabilité modifiée non-dépendante a exactement le comportement attendu sur le produit dépendant Πtu :

[4] La réalisabilité modifiée, introduite par [6], est définie sur une structure de Λ-*set* très similaire à la structure de ω-set (*cf* chapitre 3) à cette différence près que la relation de réalisabilité est exprimée vis-à-vis des λ-termes et non vis-à-vis des (codes de Gödel des) fonctions récursives partielles. Formellement, un Λ-set est donc un couple $X = (|X|, \models_X)$ formé par un ensemble quelconque $|X|$ (le support) et une relation binaire surjective $(\models_X) \subset \Lambda \times |X|$ (la relation de réalisabilité). Dans une preuve de normalisation forte, on utilise une forme particulière de Λ-set — les Λ-*sets saturés*, définis à l'aide de critères de saturation très proches de ceux que nous avons présentés au début de ce chapitre — qui permet de conserver l'invariant de normalisation dans toute la structure du modèle.

Lemme 7.1.21 (Réalisabilité sur Πtu) — *Soient $t, u \in \mathcal{M}$ des objets d'un modèle de normalisation du calcul implicite. Si Πtu est un code de type (i.e. $\Pi tu \in \mathcal{T}$), alors la relation de réalisabilité non-dépendante $M \models_{\Pi tu} f$ sur l'ensemble $\mathsf{El}(\Pi tu)$ est caractérisée par*

$$M \models_{\Pi tu} f \quad \Leftrightarrow \quad \forall\, N \in \Lambda_{\mathrm{CCI}} \quad \forall\, a \in \mathsf{El}(t) \quad N \models_t a \quad \Rightarrow \quad MN \models_{ua} fa$$

pour tout $M \in \Lambda_{\mathrm{CCI}}$ et pour tout $f \in \mathsf{El}(\Pi tu)$.

Preuve. On a en effet les équivalences suivantes :

$$\begin{aligned}
M \models_{\Pi tu} f \quad &\Leftrightarrow \quad M \in \mathsf{Red}(\Pi tu) \\
&\Leftrightarrow \quad M \in \mathsf{Red}(t) \to \bigcap_{a \in \mathsf{El}(t)} \mathsf{Red}(ua) \\
&\Leftrightarrow \quad \forall\, N \in \Lambda_{\mathrm{CCI}} \quad N \in \mathsf{Red}(t) \quad \Rightarrow \quad MN \in \bigcap_{a \in \mathsf{El}(t)} \mathsf{Red}(ua) \\
&\Leftrightarrow \quad \forall\, N \in \Lambda_{\mathrm{CCI}} \quad N \in \mathsf{Red}(t) \quad \Rightarrow \quad \forall\, a \in \mathsf{El}(t) \quad MN \in \mathsf{Red}(ua) \\
&\Leftrightarrow \quad \forall\, a \in \mathsf{El}(t) \quad \forall\, N \in \Lambda_{\mathrm{CCI}} \quad N \in \mathsf{Red}(t) \quad \Rightarrow \quad MN \in \mathsf{Red}(ua) \\
&\Leftrightarrow \quad \forall\, a \in \mathsf{El}(t) \quad \forall\, N \in \Lambda_{\mathrm{CCI}} \quad N \models_t a \quad \Rightarrow \quad MN \models_{ua} fa
\end{aligned}$$

L'argument crucial réside dans le passage entre la 4ème et la 5ème ligne, qui est justifié par le fait que les formules "$P \Rightarrow (\forall a\ Q(a))$" et "$\forall a\ (P \Rightarrow Q(a))$" sont logiquement équivalentes lorsque la variable a n'apparaît pas dans la formule P. □

Le lemme ci-dessus exprime que la réalisabilité non-dépendante définie sur l'ensemble $\mathsf{El}(\Pi tu)$ est caractérisée par la même formule que celle qui définit habituellement la relation de réalisabilité sur le produit dépendant d'une famille de Λ-sets [6, 48],[5] ce qui justifie *a posteriori* la formule de l'axiome 4.

Par ailleurs, cette remarque montre également que dans le cadre d'une preuve de normalisation forte pour le calcul implicite, nous n'avons pas besoin d'utiliser la structure de Λ-set dans toute sa généralité, et qu'il suffit simplement d'interpréter chaque type par un Λ-set saturé non-dépendant, dont la structure est donnée dans un modèle de normalisation du calcul implicite par le couple $(\mathsf{El}(t), \mathsf{Red}(t))$ pour chaque code de type $t \in \mathcal{T}$.

7.2 Le modèle concret de normalisation

Dans la section précédente, nous avons établi un théorème de normalisation abstrait dont la démonstration repose sur l'existence d'un modèle de normalisation du calcul implicite (au sens de la définition 7.1.12). Afin d'achever la preuve du théorème de normalisation forte, nous devons à présent montrer l'existence d'un modèle de normalisation forte, dont nous allons effectuer la construction dans la catégorie des μ-espaces cohérents.

[5] À cette différence près que nous utilisons ici l'application $(a, b) \mapsto a \cdot b$ du modèle $\mathcal{M}$ et non l'application de la théorie des ensembles. Rappelons que dans la catégorie des Λ-sets, la relation de réalisabilité sur le produit dépendant $\Pi(x \in X; Y_x)$ d'une famille de Λ-sets $(Y_x)_{x \in |X|}$ est définie par

$$M \models_{\Pi(x \in X; Y_x)} f \quad \equiv \quad \forall\, N \in \Lambda \quad \forall\, x \in |X| \quad N \models_X x \quad \Rightarrow \quad MN \models_{Y_x} f(x)$$

pour tout $M \in \Lambda$ et tout $f \in |\Pi(x \in X; Y_x)|$.

7.2.1 L'équation du modèle de normalisation

Au chapitre 5, nous avons construit un modèle du calcul implicite restreint à partir de la plus petite solution $\mathcal{A}$ de l'équation

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \oplus \ \mathsf{Ty}_\mu(\mathcal{A}).$$

Bien entendu, ce modèle ne constitue pas un modèle de normalisation, puisqu'il ne contient pas l'information de réductibilité nécessaire pour définir la fonction Red, mais aussi parce qu'il s'agit d'un modèle restreint, qui comporte un type vide proscrit par les axiomes des modèles de normalisation (Fig. 7.1).

Le premier problème est relativement facile à résoudre. Il suffit d'incorporer l'information de réductibilité dans les atomes typiques, en remplaçant dans l'équation ci-dessus l'espace plat $\mathsf{Ty}_\mu(\mathcal{A})$ par un nouvel espace $\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$ dont les atomes sont de la forme $\tau(X, S)$, où $X \subset \mathcal{A}$ est une base de type μ-finie et S un ensemble saturé quelconque. (La définition formelle de l'espace cohérent $\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$ sera donnée au paragraphe suivant.)

La construction d'un modèle dans lequel tous les types sont habités est en revanche beaucoup plus problématique. En effet, comme la classe des types sémantiques non-vides n'est pas stable par intersection, nous ne pouvons pas seulement nous contenter de construire l'espace $\mathcal{A}$ en prenant soin de faire disparaître toute référence à la base de type vide, c'est-à-dire en éliminant tous les atomes typiques de la forme $\tau(\mathbf{0}, S)$. Si nous procédions de cette manière naïve, nous ne serions plus en mesure de définir l'opérateur d'intersection $\forall$ dans le modèle.

Pour résoudre ce problème, l'étude des contextes fortement incohérents que nous avons effectuée au paragraphe 2.7.5 peut nous apporter de précieuses indications. En effet, nous avons montré (par la proposition 2.7.7) que dans un contexte fortement incohérent, il est possible de construire un terme qui est à la fois une fonction et un type. Dans un modèle basé sur l'équation

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \oplus \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}),$$

nous ne pourrions trouver aucune clique susceptible de jouer ce double rôle : la seule clique commune aux deux membres de la somme directe est la clique vide qui représente la fonction indéfinie, mais ne représente aucun type dans le modèle.

Pour qu'une clique puisse représenter à la fois un type et une fonction, il est donc nécessaire de modifier la relation de cohérence de manière à ce que les atomes fonctionnels puissent cohabiter avec les atomes typiques. La façon la plus naturelle d'effectuer cette modification est de remplacer la somme directe "$\oplus$" par le produit direct "&", ce qui nous amène à poser l'équation

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \& \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$$

dont la plus petite solution sera la base de notre modèle de normalisation. Cette nouvelle équation exprime que dans le modèle que nous allons construire, les cliques représenteront simultanément une fonction *et* un type.[6]

Afin de mettre un terme au problème posé par la présence du type vide, nous définirons au paragraphe 7.2.5 une clique particulière — le *diable*[7] — qui possède toutes les caractéristiques

[6] Rappelons que le produit direct "&" est le produit cartésien de la catégorie des espaces cohérents.

[7] Cette terminologie est inspirée du *démon* de la Ludique [30] qui habite tous les comportements et dont l'introduction est motivée par des raisons très similaires.

des incohérences fortes dégagées par la proposition 2.7.7. Pour définir l'ensemble $\mathcal{T}$ des types du modèle, nous ne considérerons alors que les atomes typiques formés à partir des bases de types dont la clôture supérieure contient cette clique particulière, et nous verrons que la classe de types ainsi définie possède les bonnes propriétés de clôture vis-à-vis du produit dépendant et de l'intersection.

7.2.2 Types saturés

Dans tout ce paragraphe, $\mathcal{A}$ désigne un espace cohérent quelconque.

On appelle *type saturé* de $\mathcal{A}$ tout couple (X, S) où X est une base de type de $\mathcal{A}$ et $S \in \mathbf{SAT}$ un ensemble saturé quelconque. Pour chaque type saturé (X, S), on introduit un nouvel atome $\tau(X, S)$ destiné à représenter le type saturé (X, S) comme un objet de première classe. L'ensemble de ces atomes nous permet alors de définir l'espace $\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})$:

Définition 7.2.1 (Espace cohérent des types saturés) — On appelle *espace cohérent des types saturés* de $\mathcal{A}$ l'espace cohérent noté $\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})$ dont la trame est définie par

$$\big|\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})\big| \quad = \quad \big\{\tau(X,S);\quad X \subset \mathcal{A} \text{ base de type et } \ S \in \mathbf{SAT}\big\},$$

et dont la relation de cohérence triviale est donné par

$$\tau(X_1,S_1) \frown \tau(X_2,S_2) \pmod{\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})} \qquad \equiv \qquad X_1 = X_2 \ \text{ et } \ S_1 = S_2.$$

Dans la définition ci-dessus, nous avons formé tous les couples (X, S) possibles sans imposer la moindre relation entre la base de type X et l'ensemble saturé S.[8] Il n'est en effet pas possible de déterminer à l'avance quel ensemble saturé il convient d'associer à une base de type donnée, ces objets étant de par leur nature trop différents pour qu'on puisse les relier de manière immédiate.

Proposition 7.2.2 (Covariance) — *La correspondance $\mathcal{A} \mapsto \mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})$ est un foncteur covariant non-continu dans la catégorie des plongements rigides.*

Afin de pouvoir utiliser les types saturés dans une construction de point fixe, nous utiliserons plutôt la restriction suivante :

Définition 7.2.3 (Espace cohérent des types saturés μ-finis) — Soit μ un cardinal régulier infini. On appelle *espace cohérent des types saturés μ-finis* de $\mathcal{A}$ le sous-espace cohérent de $\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})$ noté $\mathsf{Ty}^{\mathbf{SAT}}_{\mu}(\mathcal{A})$ dont la trame est définie par

$$\big|\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})\big| \quad = \quad \big\{\tau(X,S);\quad X \subset \mathcal{A} \text{ base de type } \mu\text{-finie } \ \text{et } \ S \in \mathbf{SAT}\big\}.$$

Proposition 7.2.4 (μ-continuité) — *Pour tout cardinal régulier infini μ, la correspondance $\mathcal{A} \mapsto \mathsf{Ty}^{\mathbf{SAT}}_{\mu}(\mathcal{A})$ est un foncteur covariant μ-continu dans la catégorie des plongements rigides :*

1. *Si $\mathcal{A} \Subset \mathcal{A}'$, alors $\mathsf{Ty}^{\mathbf{SAT}}_{\mu}(\mathcal{A}) \ \Subset \ \mathsf{Ty}^{\mathbf{SAT}}_{\mu}(\mathcal{A}')$.*
2. *Pour toute famille croissante d'espaces cohérents $(\mathcal{A}_i)_{i\in I}$ indicée par un ensemble μ-dirigé I, on a :*

$$\operatorname*{colim}_{i\in I}\big(\mathsf{Ty}^{\mathbf{SAT}}_{\mu}(\mathcal{A}_i)\big) \quad = \quad \mathsf{Ty}^{\mathbf{SAT}}_{\mu}\Big(\operatorname*{colim}_{i\in I}\mathcal{A}_i\Big).$$

Preuve. La preuve s'effectue de la même manière que pour la proposition 5.1.27. □

[8] Remarquons à ce sujet qu'à isomorphisme près, l'espace $\mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A})$ n'est rien d'autre que le produit tensoriel [31] $\mathsf{Ty}(\mathcal{A}) \otimes \mathbf{SAT}_{\perp}$, où $\mathbf{SAT}_{\perp}$ est l'espace cohérent plat dont les atomes sont les ensembles saturés.

7.2.3 Construction du modèle de normalisation

Dans la suite de ce chapitre, nous travaillerons dans ZFC + AI^ω. Pour tout $i > 0$, on désignera par μ_i le i-ème cardinal inaccessible, et on notera $\mu = (\sup \mu_i)^+$ le cardinal successeur de la borne supérieure de la famille $(\mu_i)_{i>0}$, c'est-à-dire le plus petit cardinal régulier supérieur à tous les cardinaux inaccessibles μ_i.

La construction de la plus petite solution de l'équation

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \& \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$$

s'effectue d'une manière très similaire à la construction que nous avons effectuée au chapitre 5. Nous adopterons en particulier la convention suivante :

Convention 7.2.5 (Codage de $\tau(X,S)$) — On suppose que pour tout espace cohérent $\mathcal{A}$, pour toute base de type $X \subset \mathcal{A}$ et pour tout ensemble saturé $S \in \mathbf{SAT}$, l'atome $\tau(X,S)$ représentant le type saturé (X,S) comme une valeur de première classe est codé de manière à ne jamais être égal à un couple (x,y) de la théorie des ensembles, c'est-à-dire :

$$\forall \mathcal{A} \qquad \forall X \subset \mathcal{A} \qquad \forall S \in \mathbf{SAT} \qquad \forall x, y \qquad\qquad \tau(X,S) \ \neq \ (x,y).$$

Grâce à cette convention, nous serons en mesure d'utiliser la définition simplifiée du produit direct (voir paragraphe 4.1.5) en posant

$$\left|[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \& \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})\right| \quad = \quad \mathcal{A}_{(\mu)} \times |\mathcal{A}| \ \cup \ \left|\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})\right|,$$

(où $\mathcal{A}_{(\mu)}$ désigne l'ensemble des cliques de $\mathcal{A}$ dont le cardinal est strictement plus petit que μ), ce qui est justifié au vu du fait que la réunion ci-dessus est disjointe d'après notre convention. Cette définition simplifiée nous permettra de construire la plus petite solution de l'équation

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \& \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}).$$

de manière à avoir de vraies inclusions rigides $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \Subset \mathcal{A}$ et $\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}) \Subset \mathcal{A}$.

La plus petite solution de l'équation récursive de domaines que nous nous proposons de résoudre est construite comme d'habitude à partir de la suite $(\mathcal{A}_x)_{x<\mu}$ définie par récurrence transfinie à l'aide des équations suivantes :

$$\begin{array}{lcll}
\mathcal{A}_0 & = & \mathsf{0} & \text{(espace cohérent nul)} \\
\mathcal{A}_{x+1} & = & [\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x] \ \& \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}_x) & \text{(avec la définition simplifiée du produit direct)} \\
\mathcal{A}_x & = & \operatorname*{colim}_{y<x} \mathcal{A}_y & \text{(si } x \text{ est un ordinal limite)}
\end{array}$$

La suite transfinie $(\mathcal{A}_x)_{x<\mu}$ étant croissante, on pose :

$$\mathcal{A} \quad = \quad \operatorname*{colim}_{x<\mu} \mathcal{A}_x.$$

Lemme 7.2.6 (Point fixe) — *L'espace $\mathcal{A}$ défini comme la colimite des espaces $\mathcal{A}_x$ $(x < \mu)$ est la plus petite solution de l'équation*

$$\mathcal{A} \quad = \quad [\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \ \& \ \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}).$$

Ainsi que nous l'avons déjà fait au chapitre 5, nous classerons les atomes de $\mathcal{A}$ par leur ordre d'apparition dans la suite transfinie $(\mathcal{A}_x)_{x<\mu}$ en introduisant une notion de rang :

Définition 7.2.7 (Rang) — Soit $\alpha \in |\mathcal{A}|$ un atome de $\mathcal{A}$. On appelle *rang de* α le plus petit ordinal x tel que $x \in |\mathcal{A}_x|$. Le rang de α est noté $\mathsf{rk}(\alpha)$. Par extension, on notera pour toute clique $a \in \mathcal{A}$ et pour toute base de type $X \subset \mathcal{A}$

$$\mathsf{rk}(a) \quad = \quad \sup_{\alpha\in a} \mathsf{rk}(\alpha) \qquad \text{et} \qquad \mathsf{rk}(X) \quad = \quad \sup_{a\in X} \mathsf{rk}(a).$$

Remarque 7.2.8 — Puisque $|\mathcal{A}|$ est la réunion (croissante) des trames des espaces $\mathcal{A}_x$ pour $x < \mu$, $\mathsf{rk}(\alpha)$ est un ordinal strictement plus petit que μ. Rappelons que le rang de α est un ordinal successeur puisque, lorsque x est un ordinal limite, chaque atome de l'espace $\mathcal{A}_x$ a déjà été introduit dans un espace $\mathcal{A}_y$ où $y < x$. En revanche, le rang d'une clique ou d'une base de type peut être n'importe quel ordinal inférieur où égal à μ.

Proposition 7.2.9 (Structure inductive de $|\mathcal{A}|$) — *Les atomes de $\mathcal{A}$ sont*

– *soit de la forme (a, β), où $a \in \mathcal{A}$, $\beta \in |\mathcal{A}|$ et $\overline{\overline{a}} < \mu$*

– *soit de la forme $\tau(X, S)$, où $S \in \mathbf{SAT}$, $X \subset \mathcal{A}$, $\overline{\overline{X}} < \mu$ et $\overline{\overline{a}} < \mu$ pour tout $a \in X$.*

De plus, on a les égalités suivantes :

$$\mathsf{rk}((a,\beta)) \quad = \quad \max\big(\mathsf{rk}(a), \mathsf{rk}(\beta)\big) + 1 \qquad \textit{et} \qquad \mathsf{rk}\big(\tau(X,S)\big) \quad = \quad \mathsf{rk}(X) + 1.$$

Lemme 7.2.10 (Cardinalité des espaces $(\mathcal{A}_x)$) — *La suite transfinie d'espaces cohérents $(\mathcal{A}_x)_{x<\mu}$ satisfait les propriétés suivantes :*

1. *pour tout $i < \omega$ et pour tout $x < \mu_i$, on a $\overline{\overline{|\mathcal{A}_x|}} < \mu_i$.*
2. *pour tout $i < \omega$, on a $\overline{\overline{|\mathcal{A}_{\mu_i}|}} \leq \mu_i$.*

Lemme 7.2.11 — *Pour tout $i < \omega$ et pour tout $x < \mu_i$, on a*

1. $\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}_x) = \mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A}_x)$ *(espace cohérent de tous les types saturés)*
2. $(\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x) = (\mathcal{A}_x \xrightarrow{q} \mathcal{A}_x)$ *(espace cohérent de toutes les fonctions quasi-stables)*
3. $\mathcal{A}_{x+1} \;=\; [\mathcal{A}_x \xrightarrow{\mu} \mathcal{A}_x] \,\&\, \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}_x) \;=\; [\mathcal{A}_x \xrightarrow{q} \mathcal{A}_x] \,\&\, \mathsf{Ty}^{\mathbf{SAT}}(\mathcal{A}_x)$.

7.2.4 Structure du λ-modèle sous-extensionnel $\mathcal{A}$

Par construction, l'espace $\mathcal{A}$ satisfait les deux inclusions rigides suivantes :

$$[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \;\Subset\; \mathcal{A} \qquad \text{et} \qquad \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A}) \;\Subset\; \mathcal{A}.$$

D'après la première de ces deux inclusions rigides, $\mathcal{A}$ est naturellement muni d'une structure de λ-modèle sous-extensionnel (Prop. 4.3.20) dont l'opérateur d'application est donné par :

$$f \cdot a \quad = \quad \big\{\beta \in \mathcal{A};\quad \exists a_0 \subset a \quad (a_0, \beta) \in f\big\}.$$

Remarquons que la formule ci-dessus ignore purement et simplement les atomes typiques $\tau(X, S)$ qui pourraient être présents dans la clique f, puisque l'application de fonction ne prend en compte que la partie fonctionnelle de f extraite par la rétraction de $\mathcal{A}$ sur $\mathcal{A} \xrightarrow{\mu} \mathcal{A}$ (*cf* paragraphe 4.3.3).

Afin d'étudier la structure de $\mathcal{A}$, on introduit les deux fonctions de projection suivantes :

Définition 7.2.12 (Projections) — On désigne par

- $\mathsf{fst} : \mathcal{A} \multimap [\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$ la rétraction associée à l'inclusion rigide $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \Subset \mathcal{A}$
- $\mathsf{snd} : \mathcal{A} \multimap \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$ la rétraction associée à l'inclusion rigide $\mathsf{Ty}_\mu^{\mathbf{SAT}} \Subset \mathcal{A}$.

Ces fonctions sont caractérisées pour toute clique $a \in \mathcal{A}$ par

$$\mathsf{fst}(a) \quad = \quad a \ \cap \ \left|\mathcal{A} \xrightarrow{\mu} \mathcal{A}\right| \qquad \text{et} \qquad \mathsf{snd}(a) \quad = \quad a \ \cap \ \left|\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})\right|.$$

Lemme 7.2.13 — *Pour toute clique $a \in \mathcal{A}$ on a*

$$\mathsf{fst}(a) \cap \mathsf{snd}(a) \quad = \quad \varnothing \qquad \textit{et} \qquad \mathsf{fst}(a) \cup \mathsf{snd}(a) \quad = \quad a.$$

Intuitivement, $\mathsf{fst}(a)$ désigne la partie fonctionnelle de a tandis que $\mathsf{snd}(a)$ désigne sa partie typique. Concernant cette dernière, on a

- Soit $\mathsf{snd}(a) = \varnothing$, auquel cas a est une clique purement fonctionnelle, qui ne représente aucun type dans le modèle.
- Soit $\mathsf{snd}(a) = \{\tau(X, S)\}$ (puisque l'espace $\mathsf{Ty}_\mu^{\mathbf{SAT}}(a)$ est plat), ce qui signifie que a représente le type saturé (X, S).

Autrement dit, une clique $a \in \mathcal{A}$ est composée d'atomes fonctionnels et/ou typiques, mais lorsqu'elle contient un atome typique $\tau(X, S)$, cet atome est unique, quel que soit par ailleurs le contenu fonctionnel de la clique a.

7.2.5 Types bien formées

Dans le modèle que nous venons de construire, il subsiste encore des atomes typiques $\tau(X, S)$ formés à partir de la base de type vide $X = \mathbf{0}$. Afin d'être en mesure de définir la classe $\mathcal{T}$ des cliques représentant les types dans le modèle — lesquels doivent être tous non-vides — nous devons d'abord définir une clique particulière, le *diable*, qui nous servira d'habitant canonique :

Définition 7.2.14 (Diable) — Soit $(\delta_i)_{i\in\omega}$ la suite d'atomes de $\mathcal{A}$ définie par récurrence sur $i \in \omega$ par

$$\delta_0 \ = \ \tau(\mathbf{1}, SN) \qquad \text{et} \qquad \delta_{i+1} \ = \ (\varnothing, \delta_i).$$

On appelle *diable* et on note d l'ensemble défini par $d \ = \ \{\delta_i; \quad i \in \omega\}$.

Lemme 7.2.15 — *Le diable est une clique de $\mathcal{A}$ dont le rang est $\mathsf{rk}(d) = \omega$.*

Preuve. Considérons la famille croissante d'ensembles $(d_i)_{i\in\omega}$ définie par $d_i = \{\delta_0; \ldots; \delta_i\}$, et montrons par récurrence sur i que d_i est une clique de $\mathcal{A}$ (c'est-à-dire un ensemble d'atomes deux à deux cohérents entre eux). Ce résultat est immédiat pour $d_0 = \{\delta_0\}$, qui n'a qu'un seul atome. Supposons à présent que d_i est une clique. La fonction constante $\mathbf{K}d_i$ est une fonction μ-stable dont la trace est donnée par

$$\mathsf{Tr}(\mathbf{K}d_i) \quad = \quad \{(\varnothing, \delta_0); \ldots; (\varnothing, \delta_i)\} \quad = \quad d_{i+1} \setminus \{\delta_0\}.$$

Puisque $\delta_0 = \tau(\mathbf{1}, SN)$ est cohérent avec n'importe quel atome fonctionnel, l'ensemble $d_{i+1} = \mathsf{Tr}(\mathbf{K}d_i) \cup \{\delta_0\}$ est lui aussi une clique. Par conséquent, $d = \bigcup d_i$ est une clique. Il est par ailleurs clair que $\mathsf{rk}(\delta_i) = i + 1$ pour tout $i \in \omega$, d'où il ressort que $\mathsf{rk}(d) = \omega$. □

Remarque 7.2.16 — D'après la définition du diable, on a

$$\mathsf{fst}(d) \;=\; \mathbf{K}d \qquad \text{et} \qquad \mathsf{snd}(d) \;=\; \{\tau(\mathbf{1}, SN)\},$$

ce qui signifie que

- en tant que fonction, le diable se comporte comme une fonction constante retournant le diable lui-même lorsqu'elle est appliquée à n'importe quelle clique du modèle.
- en tant que type saturé, le diable représente le type de tous les objets du modèle $\mathcal{A}$ (donné par la base de type $\mathbf{1} = \{\varnothing\}$), et annoté par l'ensemble saturé SN.[9]

La proposition suivante établit que la classe des bases de types de $\mathcal{A}$ dont la clôture supérieure contient le diable est close par les opérateurs $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$.

Proposition 7.2.17 — *Soient X une base de type μ-finie de $\mathcal{A}$ et $(Y_a)_{a\in\mathcal{A}}$ une famille de bases de types de $\mathcal{A}$ indicée par X. Si $d \in \uparrow Y_a$ pour tout $a \in X$, alors :*

$$d \in \uparrow\Pi(a{\in}X; Y_a) \qquad \textit{et} \qquad d \in \uparrow\forall(a{\in}X; Y_a).$$

Preuve. Cette proposition est immédiate dans le cas de l'intersection puisque $\uparrow\forall(a{\in}X; Y_a) = \bigcap \uparrow Y_a$. Dans le cas du produit dépendant, il suffit de remarquer que puisque $d \in \uparrow Y_a$ pour tout $a \in X$, on a $\mathbf{K}d \in \uparrow\Pi(a{\in}X; Y_a)$. Comme par ailleurs $\mathbf{K}d \subset d$, on a alors $d \in \uparrow\Pi(a{\in}X; Y_a)$. □

Remarque 7.2.18 — Dans l'énoncé ci-dessus, nous avons dû supposer que X est μ-fini de manière à avoir l'inclusion $\Pi(a{\in}X; Y_a) \subset [\mathcal{A} \xrightarrow{\mu} \mathcal{A}]$ (d'après la proposition 5.1.21), ce qui nous permet de considérer $\Pi(a{\in}X; Y_a)$ comme une base de type de $\mathcal{A}$ à travers l'inclusion rigide $[\mathcal{A} \xrightarrow{\mu} \mathcal{A}] \Subset \mathcal{A}$. Remarquons que ce changement d'espace n'est pas anodin. Dans l'espace $\mathcal{A} \xrightarrow{\mu} \mathcal{A}$, la base de type $\Pi(a{\in}X; Y_a)$ et sa clôture supérieure ne contiennent que des cliques purement fonctionnelles. Dans $\mathcal{A}$ en revanche, la clôture supérieure de $\Pi(a{\in}X; Y_a)$ contient des cliques "mixtes" (d'après la relation de cohérence sur le produit direct), ce qui explique la présence du diable qui n'est pourtant pas une clique purement fonctionnelle.

Définition 7.2.19 (Type bien formé) — Soit $a \in \mathcal{A}$ une clique. On dit que a est un *type bien formé* (ou encore un *code de type*) s'il existe une base de type $X \subset \mathcal{A}$ et un ensemble saturé S tels que $\mathsf{snd}(a) = \{\tau(X, S)\}$ et $d \in \uparrow X$. L'ensemble des types bien formés de $\mathcal{A}$ est noté $\mathcal{T}$.

Lemme 7.2.20 — *$\mathcal{T}$ est un type sémantique de $\mathcal{A}$, dont la base de type est formée par tous les singletons de la forme $\{\tau(X, S)\}$, où X décrit l'ensemble des bases de types μ-finies de $\mathcal{A}$ telles que $d \in \uparrow X$, et où S décrit l'ensemble des ensembles saturés.*

Preuve. Il suffit de remarquer que l'ensemble $Y \subset \mathcal{A}$ défini par

$$Y \quad = \quad \Big\{\{\tau(X, S)\};\quad X \subset \mathcal{A} \text{ base de type } \mu\text{-finie},\ d \in \uparrow X,\ S \in \mathbf{SAT}\Big\}$$

est une base de type de $\mathcal{A}$, dont la clôture supérieure est précisément l'ensemble $\mathcal{T}$. □

Nous sommes à présent en mesure de définir la fonction de décodage $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{A}) \setminus \{\varnothing\}$ ainsi que la fonction de réductibilité $\mathsf{Red} : \mathcal{T} \to \mathbf{SAT}$:

[9] On pourra vérifier que le choix de l'ensemble saturé SN est ici purement arbitraire, et que tout autre ensemble saturé aurait pu faire l'affaire. En revanche, le choix de la base de type $\mathbf{1}$ est crucial, puisque cette base de type est dans le modèle la seule base de type qui puisse nous assurer que le diable soit son propre type.

Définition 7.2.21 (Fonctions de décodage et de réductibilité) — Pour tout type bien formé $t \in \mathcal{T}$, on note

$$\mathsf{El}(t) \;=\; \uparrow X \quad \text{et} \quad \mathsf{Red}(t) \;=\; S, \qquad \text{où} \quad \{\tau(X,S)\} \;=\; \mathsf{snd}(t).$$

(Remarquons que pour tout $t \in \mathcal{T}$, $\mathsf{El}(t)$ est habité par le diable d.)

Lemme 7.2.22 (Validité des axiomes 1, 2 et 3) — *Le sous-ensemble $\mathcal{T} \subset \mathcal{A}$ muni des fonctions $\mathsf{El} : \mathcal{T} \to \mathfrak{P}(\mathcal{A}) \setminus \{\varnothing\}$ et $\mathsf{Red} : \mathcal{T} \to \mathbf{SAT}$ satisfont les axiomes 1, 2 et 3 des modèles de normalisation.*

Preuve. Les axiomes 1 et 3 sont immédiats, car $\mathcal{T}$ est un type sémantique de $\mathcal{A}$, de même que $\mathsf{El}(t)$ pour tout $t \in \mathcal{T}$. Pour montrer l'axiome 2, supposons $t, t' \in \mathcal{T}$ tels que $t \subset t'$. Soient X, S, X' et S' tels que $\mathsf{snd}(t) = \{\tau(X,S)\}$ et $\mathsf{snd}(t') = \{\tau(X',S')\}$. Comme $t \subset t'$, on a $\mathsf{snd}(t) \subset \mathsf{snd}(t')$, d'où $X = X'$ et $S = S'$. □

7.2.6 Opérateurs de produit dépendant et d'intersection

Définition 7.2.23 (Opérateurs de produit dépendant et d'intersection) — On définit deux applications $\Pi, \forall : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ en posant

$$\begin{aligned} \Pi(t,f) \;&=\; \Big\{\tau\Big(\Pi(a{\in}X; Y_a),\; S \to \bigcap_{a\in X} R_a\Big)\Big\} \\ \forall(t,u) \;&=\; \Big\{\tau\Big(\forall(a{\in}X; Y_a),\; \bigcap_{a\in X} R_a\Big)\Big\} \end{aligned}$$

s'il existe X, S, $(Y_a)_{a\in X}$ et $(R_a)_{a\in X}$ tels que

1. $\mathsf{snd}(t) = \{\tau(X,S)\}$
2. $\mathsf{snd}(u \cdot a) = \{\tau(Y_a, R_a)\}$ pour tout $a \in X$
3. X est une base de type (μ-finie) de $\mathcal{A}$ telle que $d \in \uparrow X$
4. Y_a est une base de type (μ-finie) de $\mathcal{A}$ telle que $d \in \uparrow Y_a$ pour tout $a \in X$
5. $\Pi(a{\in}X; Y_a)$ (resp. $\forall(a{\in}X; Y_a)$) est une base de type μ-finie de $\mathcal{A}$.

et

$$\Pi(t,u) \quad = \quad \forall(t,u) \quad = \quad \varnothing$$

dans tous les autres cas.

Remarque 7.2.24 — Les conditions 1 et 3 ci-dessus expriment que t est un type bien formé (*i.e.* $t \in \mathcal{T}$). De même, les conditions 2 et 4 expriment que $u \cdot a \in \mathcal{T}$ pour tout $a \in X$, ce qui entraîne par ailleurs que $u \cdot a \in \mathcal{T}$ pour tout $a \in \mathsf{El}(t) = \uparrow X$, puisque

$$\forall\, a \in \uparrow X \qquad \mathsf{snd}(u \cdot a) \quad = \quad \mathsf{snd}(u \cdot \lfloor a \rfloor_X) \quad = \quad \big\{\tau\big(Y_{\lfloor a\rfloor_X}, R_{\lfloor a\rfloor_X}\big)\big\}$$

d'après un argument de monotonie immédiat. La condition 5 impose la μ-finitude de la base de type $\Pi(a{\in}X; Y_a)$ (resp. $\forall(a{\in}X; Y_a)$) qui n'est pas automatique puisque μ n'est pas un cardinal inaccessible. Lorsque les conditions 1 à 5 sont vérifiés, on a alors $d \in \uparrow\Pi(a{\in}X; Y_a)$ et $d \in \uparrow\Pi(a{\in}X; Y_a)$ d'après le lemme 7.2.17, d'où il ressort que $\Pi tu \in \mathcal{T}$ et $\forall tu \in \mathcal{T}$.

Proposition 7.2.25 (μ-stabilité de Π et $\forall$) — *Les applications* $\Pi : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ *et* $\forall : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ *sont des fonctions* μ*-stables.*

Preuve. La preuve se conduit de la même manière que la preuve de la proposition 5.1.29, à cette différence près qu'ici, Π et $\forall$ sont des fonctions prenant leurs deux arguments dans $\mathcal{A}$ et retournant leur résultat dans $\mathcal{A}$. Cependant, les deux fonctions Π et $\forall$ ne consultent leurs arguments qu'à travers les rétractions de $\mathcal{A}$ sur $\mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$ (pour le premier argument) et sur $\mathcal{A} \xrightarrow{\mu} \mathsf{Ty}_\mu^{\mathbf{SAT}}(\mathcal{A})$ (pour le second), ce qui nous place dans un cadre très analogue à celui de la proposition 5.1.29. □

Comme Π et $\forall$ sont des fonctions μ-stables, ces deux fonctions peuvent elles-mêmes être vues comme des cliques de $\mathcal{A} \,\&\, \mathcal{A} \xrightarrow{\mu} \mathcal{A}$. On utilisera par la suite les versions curryfiées correspondantes

$$\Pi, \forall \quad : \quad \mathcal{A} \xrightarrow{\mu} \mathcal{A} \xrightarrow{\mu} \mathcal{A} \quad \Subset \quad \mathcal{A},$$

considérées comme des cliques de $\mathcal{A}$ à travers l'inclusion rigide ci-dessus.

Proposition 7.2.26 (Validité des axiomes 4 et 5) — *Les constantes* $\Pi, \forall \in \mathcal{A}$ *satisfont les axiomes 4 et 5 des modèles de normalisation.*

Preuve. Il suffit de remarquer que Πtu et $\forall tu$ ne sont des cliques définies que lorsqu'il existe des bases de types X et Y_a ($a \in X$) ainsi que des ensembles saturés S et R_a ($a \in X$) qui satisfont les conditions 1–5 de la définition 7.2.23. De ces conditions découlent immédiatement les égalités recherchées. □

7.2.7 La hiérarchie d'univers

Définition 7.2.27 (Univers) — Pour tout $i > 0$ on considère l'ensemble $\mathsf{U}_i \subset \mathcal{A}$ défini par

$$\mathsf{U}_0 \;=\; \Big\{\{\tau(\mathbf{1}, S)\};\quad S \in \mathbf{SAT}\Big\}$$

$$\mathsf{U}_i \;=\; \Big\{\{\tau(X, S)\};\quad X \subset \mathcal{A}\ \mu\text{-finie},\quad S \in \mathbf{SAT},\quad d \in \uparrow X \quad \text{et} \quad \mathsf{rk}(X) < \mu_i\Big\} \qquad (i > 0)$$

Lemme 7.2.28 — *Pour tout* $i \in \omega$, U_i *est une base de type* μ*-finie de* $\mathcal{A}$ *telle que* $d \in \uparrow\mathsf{U}_i$.

Preuve. Chacun des ensembles $\mathsf{U}_i \subset \mathcal{A}$ ne contient que des singletons typiques, et forme de ce fait une base de type de $\mathcal{A}$. Pour U_0, on remarque que

$$\overline{\overline{\mathsf{U}_0}} \quad = \quad \overline{\overline{\mathbf{SAT}}} \quad = \quad 2^{\aleph_0} \quad < \quad \mu_1 \quad < \quad \mu,$$

ce qui prouve que U_0 est une base de type μ-finie. Supposons $i > 0$. On a alors d'après le lemme 7.2.10 (pour l'inégalité centrale)

$$\overline{\overline{\mathsf{U}_i}} \quad \leq \quad \overline{\overline{|\mathcal{A}_{\mu_i}|}} \quad < \quad \mu_{i+1} \quad < \quad \mu,$$

ce qui prouve que U_i est une base de type μ-finie. Pour tout $i \in \omega$, on a d'après la définition ci-dessus $\{\tau(\mathbf{1}, SN)\} \in \mathsf{U}_i$, d'où il ressort que $d \in \uparrow\mathsf{U}_i$ puisque $\{\tau(\mathbf{1}, SN)\} = \{\delta_0\} \subset d$. □

Comme chacune des bases de types $\mathsf{U}_i \subset \mathcal{A}$ est μ-finie et contient le diable d dans sa clôture supérieure, les cliques $\{\tau(\mathsf{U}_i, SN)\}$ constituent des types bien formés, ce qui justifie la définition suivante :

Définition 7.2.29 (Constantes d'univers) — Pour tout $i \in \omega$, on note $\mathsf{u}_i = \{\tau(\mathsf{U}_i, SN)\}$.

D'après la remarque précédente, on a $\mathsf{u}_i \in \mathcal{T}$ pour tout $i \in \omega$. Par ailleurs, le contenu extensionnel de chaque code de type u_i est donné par

$$\begin{array}{rcl} \mathsf{El}(\mathsf{u}_0) & = & \uparrow\mathsf{U}_0 \quad = \quad \Big\{a \in \mathcal{A};\quad \exists S \in \mathbf{SAT} \quad \tau(\mathbf{1}, S) \in a\Big\} \\ \mathsf{El}(\mathsf{u}_i) & = & \uparrow\mathsf{U}_i \quad = \quad \Big\{a \in \mathcal{A};\quad \exists X \subset \mathcal{A}\ \mu\text{-finie} \quad \exists S \in \mathbf{SAT} \\ & & \qquad\qquad \tau(X, S) \in a, \quad d \in \uparrow X \quad \text{et} \quad \mathsf{rk}(X) < \mu_i\Big\} \qquad (i > 0) \end{array}$$

Proposition 7.2.30 (Validité des axiomes 6, 7 et 8) — *La famille de constantes $(\mathsf{u}_i)_{i\in\omega}$ satisfont les axiomes 6, 7 et 8 des modèles de normalisation.*

Preuve. (Axiome 6) Soient $i \in \omega$ et $t \in \mathsf{El}(t) = \uparrow\mathsf{U}_i$. D'après la caractérisation des éléments de $\uparrow\mathsf{U}_i$ figurant ci-dessus, on a $\mathsf{snd}(t) = \{\tau(X, S)\}$ (avec $X = \mathbf{1}$ si $i = 0$) où X est une base de type μ-finie telle que $d \in \uparrow X$ (car $\uparrow X = \uparrow\mathbf{1} = \mathcal{A}$ si $i = 0$, et par hypothèse si $i > 0$). Par conséquent, t est un type bien formé, d'où $t \in \mathcal{T}$.

(Axiome 7) Considérons la suite croissante d'espaces cohérents $(\mathcal{A}_x)_{x<\mu}$ introduite au paragraphe 7.2.3 dont la colimite est $\mathcal{A}$. Pour tout $S \in \mathbf{SAT}$ on a $\mathsf{rk}(\tau(\mathbf{1}, S)) = 1$, d'où l'on tire que $\mathsf{rk}(\mathsf{U}_0) = 1 < \mu_1$. Comme par ailleurs $d \in \uparrow\mathsf{U}_0$, on a $\mathsf{u}_0 = \{\tau(\mathsf{U}_0, SN)\} \in \mathsf{U}_1$ par définition de U_1, d'où il ressort que $\mathsf{u}_0 \in \mathsf{El}(\mathsf{u}_1) = \uparrow\mathsf{U}_1$. Considérons à présent un entier $i > 0$. La base de type U_i ne contient par définition que des cliques de rang $< \mu_i$. Par conséquent, on a $\mathsf{rk}(\mathsf{U}_i) \leq \mu_i$ (par passage à la borne supérieure), d'où $\mathsf{rk}(\tau(\mathsf{U}_i, SN)) \leq \mu_i + 1 < \mu_{i+1}$. Comme $d \in \uparrow\mathsf{U}_i$, on a $\mathsf{u}_i = \{\tau(\mathsf{U}_i, SN)\} \in \mathsf{U}_{i+1} \subset \mathsf{El}(\mathsf{u}_{i+1})$.

(Axiome 8) D'après la définition des bases de types U_i, on a clairement $\mathsf{U}_i \subset \mathsf{U}_{i+1}$ pour tout $i \in \omega$, d'où $\mathsf{El}(\mathsf{u}_i) = \uparrow\mathsf{U}_i \subset \uparrow\mathsf{U}_{i+1} = \mathsf{El}(\mathsf{u}_{i+1})$. L'autre inclusion demandée est ici une égalité puisque $\mathsf{Red}(\mathsf{u}_i) = SN = \mathsf{Red}(\mathsf{u}_{i+1})$. □

Proposition 7.2.31 (Validité des axiomes 9 et 10) — *Les constantes Π, $\forall$ et $(\mathsf{u}_i)_{i\in\omega}$ satisfont les axiomes 9 et 10 des modèles de normalisation.*

Preuve. (Axiome 10) Soit $i \in \omega$. Considérons des cliques $t, u \in \mathcal{A}$ tels que $t \in \mathsf{El}(\mathsf{u}_i)$ et $ua \in \mathsf{El}(\mathsf{u}_0)$ pour tout $a \in \mathsf{El}(t)$.
Comme $t \in \mathsf{El}(\mathsf{u}_i) \subset \mathcal{T}$, il existe une base de type μ-finie $X \subset \mathcal{A}$ et un ensemble saturé $S \in \mathbf{SAT}$ tels que $\mathsf{snd}(t) = \{\tau(X, S)\}$ et $d \in \uparrow X$.
Comme $ua \in \mathsf{El}(\mathsf{u}_0)$ pour tout $a \in \mathsf{El}(t) = \uparrow X$, il existe une famille $(R_a)_{a\in X}$ d'ensembles saturés tels que $\mathsf{snd}(ua) = \{\tau(\mathbf{1}, R_a)\}$ pour tout $a \in X$.
D'après la définition des cliques Π et $\forall$ et en vertu du lemme 5.1.23 nous avons

$$\begin{array}{rcl} \Pi tu & = & \big\{\tau\big(\Pi(\_{\in}X; \mathbf{1}),\ S \to \bigcap R_a\big)\big\} \quad = \quad \big\{\tau\big(\mathbf{1},\ S \to \bigcap R_a\big)\big\} \\ \forall tu & = & \big\{\tau\big(\forall(\_{\in}X; \mathbf{1}),\ \bigcap R_a\big)\big\} \qquad\quad = \quad \big\{\tau\big(\mathbf{1},\ \bigcap R_a\big)\big\}, \end{array}$$

d'où il ressort que $\Pi tu \in \mathsf{U}_0 \subset \mathsf{El}(\mathsf{u}_0)$ et $\forall tu \in \mathsf{U}_0 \subset \mathsf{El}(\mathsf{u}_0)$.

(Axiome 9) Le cas $i = 0$ a déjà été traité ci-dessus. Supposons $i > 0$.
Comme $t \in \mathsf{El}(\mathsf{u}_i) \subset \mathcal{T}$, il existe une base de type μ-finie $X \subset \mathcal{A}$ et un ensemble saturé $S \in \mathbf{SAT}$ tels que $\mathsf{snd}(t) = \{\tau(X, S)\}$, $d \in \uparrow X$ et $\mathsf{rk}(X) < \mu_i$.

Comme $ua \in \mathsf{El}(\mathsf{u}_i)$ pour tout $a \in \mathsf{El}(t) = \uparrow X$, il existe une famille $(Y_a)_{a\in X}$ de bases de types μ-finies et une famille $(R_a)_{a\in X}$ d'ensembles saturés telles que $\mathsf{snd}(ua) = \{\tau(Y_a, R_a)\}$, $d \in \uparrow Y_a$ et $\mathsf{rk}(Y_a) < \mu_i$ pour tout $a \in X$.
Posons alors $x = \mathsf{rk}(X) < \mu_i$. On remarque alors (d'après le lemme 7.2.10) que

$$\overline{\overline{X}} \quad \leq \quad \overline{\overline{\mathcal{A}_x}} \quad \leq \quad 2^{\overline{|\mathcal{A}_x|}} \quad < \quad \mu_i$$

d'où il ressort que l'ordinal z défini par

$$z \quad = \quad \max\Big(x,\ \sup_{a\in X} \mathsf{rk}(Y_a)\Big)$$

est strictement plus petit que μ_i, qui est inaccessible. Par conséquent, X et Y_a ($a \in X$) sont des bases de types de $\mathcal{A}_z$, d'où il découle que $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$ sont des bases de types de $\mathcal{A}_z$ et de $[\mathcal{A}_z \xrightarrow{q} \mathcal{A}_z] \Subset \mathcal{A}_{z+1}$ (Lemme 7.2.11) respectivement, ce qui prouve que leur rang est strictement plus petit que μ_i.
Par conséquent, les bases de types $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$ sont μ-finies, d'où il ressort d'après la définition de Π et $\forall$ que

$$\begin{aligned} \Pi tu &= \big\{\tau\big(\Pi(a{\in}X; Y_a),\ S \to \bigcap R_a\big)\big\} \\ \forall tu &= \big\{\tau\big(\forall(a{\in}X; Y_a),\ \bigcap R_a\big)\big\}. \end{aligned}$$

D'après la proposition 7.2.17, le diable appartient à la clôture supérieure des bases de types $\forall(a{\in}X; Y_a)$ et $\Pi(a{\in}X; Y_a)$, dont nous avons montré qu'elles étaient de rang strictement plus petit que μ_i, d'où il ressort que $\Pi tu \in \mathsf{U}_i \subset \mathsf{El}(\mathsf{u}_i)$ et $\forall tu \in \mathsf{U}_i \subset \mathsf{El}(\mathsf{u}_i)$. □

D'après les lemmes 7.2.22, 7.2.26, 7.2.30 et 7.2.31, nous pouvons donc conclure que :

Proposition 7.2.32 (Modèle de normalisation) — *La structure*

$$\big(\mathcal{A}, \cdot, [\![]\!], \subset, \mathcal{T}, \mathsf{El}, \mathsf{Red}, \Pi, \forall, (\mathsf{u}_i)_{i\in\omega}\big)$$

est un modèle de normalisation du calcul implicite.

Ce qui achève la démonstration du théorème de normalisation forte.

Troisième partie

Types et ensembles

Chapitre 8

Le Paradoxe de Russell dans les systèmes U et U^-

Dans ce chapitre, nous présentons une nouvelle preuve d'incohérence des systèmes U et U^- basée sur une traduction du paradoxe de Russell. Initialement, l'incohérence du système U a été établie en 1971 par J.-Y Girard dans sa thèse d'État [28] à partir d'une traduction du paradoxe de Burali-Forti.[1] Le paradoxe de Girard, qui est le premier paradoxe découvert en théorie des types, se transpose de manière triviale dans le système de types muni d'un type de tous les types (Type : Type) dont il permet de montrer l'incohérence.[2] En 1991, T. Coquand [18] a affiné le résultat en montrant l'incohérence d'un sous-système du système U (le système U^-) à partir d'une traduction d'un résultat de Reynolds [55] montrant que le polymorphisme du système F n'a pas de modèle ensembliste. Depuis, les systèmes U et U^- ont fait l'objet de nombreuses études, notamment pour y construire des combinateurs bouclants (*looping combinators*) [34, 19], voire des combinateurs de point fixe [35].[3]

L'originalité du paradoxe que nous allons présenter ne vient pas du fait qu'il s'agit d'une traduction du paradoxe de Russell plutôt que du paradoxe de Burali-Forti (dont la preuve n'est en réalité pas très éloignée), mais du fait que ce paradoxe repose sur une traduction complète du fragment intuitionniste de la théorie des ensembles de Frege [23]. Autrement dit, la construction que nous allons effectuer est une véritable construction de modèle, au travers de laquelle nous serons en mesure de transformer n'importe quel paradoxe de la théorie des ensembles — pourvu qu'il soit formalisé en logique intuitionniste — en un paradoxe de la théorie des types.

Pour cela, nous allons nous appuyer sur les travaux de P. Aczel autour de la théorie des hyper-ensembles [4] et de l'axiome d'anti-fondation (*cf* paragraphe 8.3.1), en proposant un

[1] Rappelons que le paradoxe de Burali-Forti est historiquement le premier paradoxe découvert dans la théorie des ensembles introduite par G. Cantor puis formalisée par G. Frege dans [23]. Pour établir ce paradoxe, on considère l'ensemble O de tous les ordinaux, dont on vérifie immédiatement qu'il est lui-même un ordinal. De ceci, on tire la relation $O \in O$ — c'est-à-dire $O < O$ — qui entre en contradiction avec le fait que la relation d'ordre sur les ordinaux est bien fondée.

[2] Il suffit pour cela de remplacer dans la preuve du paradoxe toutes les occurrences des sortes Prop, Type et Kind du système U par l'unique sorte Type du système Type : Type.

[3] La plus petite preuve de la proposition fause "$\bot$" (construite dans le système U^-) reste à ce jour celle qui a été présentée par A.J. Hurkens dans [35]. Récemment, H. Geuvers a remarqué qu'en effaçant toutes les annotations de types sous les λ-abstractions dans le terme de preuve correspondant, on obtient un véritable combinateur de point fixe dans l'équivalent "*domain-free*" [12] du système U^-.

schéma d'interprétation dans lequel les ensembles sont représentés par des graphes pointés, et où l'égalité extensionnelle de la théorie des ensembles est interprétée par l'existence d'une bissimulation entre deux graphes pointés.

Contrairement à l'interprétation standard de la théorie des ensembles en théorie des types — dans laquelle les ensembles sont représentés par des arbres bien fondés [1, 2, 3, 64, 5] (*cf* paragraphe 8.2.4) — le codage que nous utilisons ici est particulièrement économique d'un point de vue théorique, puisqu'il n'utilise ni sommes dépendantes, ni types inductifs primitifs. Pour cette raison, le paradigme "ensemble = graphe pointé" est un outil très intéressant pour mesurer la puissance théorique de petites théories des types (*i.e.* sans types inductifs) pourvu que celles-ci soient suffisamment expressives (ce qui est le cas de nombreux systèmes de types purs imprédicatifs).

La traduction de la théorie des ensembles incohérente de Frege effectuée dans ce chapitre en est un premier exemple, et nous verrons au chapitre suivant que les mêmes idées s'adaptent sans difficulté au cadre de théories (supposées) cohérentes, ce qui nous permettra de démontrer la cohérence relative de la théorie des ensembles de Zermelo par rapport à celle de la théorie des types de Church avec trois univers (et sans entiers primitifs).

8.1 Une présentation des systèmes U et U^-

8.1.1 Les systèmes de types purs λU et λU^-

Définition 8.1.1 (λU) — On appelle λU (ou encore *système U*) le PTS dont la hiérarchie des sortes est définie par

$$\mathcal{S}_U \;=\; \big\{\mathsf{Prop};\ \mathsf{Type};\ \mathsf{Kind}\big\} \qquad \text{et} \qquad \mathbf{Axiom}_U \;=\; \big\{(\mathsf{Prop} : \mathsf{Type});\ (\mathsf{Type} : \mathsf{Kind})\big\},$$

et dont les règles de formation de produits dépendants sont données par l'ensemble

$$\mathbf{Rule}_U \quad = \quad \big\{(\mathsf{Prop}, \mathsf{Prop});\ (\mathsf{Type}, \mathsf{Prop});\ (\mathsf{Kind}, \mathsf{Prop});\ (\mathsf{Type}, \mathsf{Type});\ (\mathsf{Kind}, \mathsf{Type})\big\}.$$

Dans l'écriture ci-dessus, nous avons utilisé la notation simplifiée de [26] en désignant par le couple (s_1, s_2) chaque triplet de sortes (s_1, s_2, s_2) dont les deux dernières composantes sont les mêmes. Intuitivement, le système U est une extension de $F\omega$ dans laquelle on a introduit une nouvelle sorte Kind telle que $\mathsf{Type} : \mathsf{Kind}$ ainsi que deux nouvelles règles de formation de produits dépendants $(\mathsf{Kind}, \mathsf{Type})$ et $(\mathsf{Kind}, \mathsf{Prop})$. Dans ce qui suit, nous travaillerons également avec la restriction suivante du système U :

Définition 8.1.2 (λU^-) — On appelle λU^- (ou encore *système* U^-) le système de types purs basé sur les mêmes sortes et les mêmes axiomes que le système U, et dont les règles de formation de produits dépendants sont données par

$$\begin{aligned} \mathbf{Rule}_{U^-} \;&=\; \mathbf{Rule}_U \setminus \{(\mathsf{Kind}, \mathsf{Prop})\} \\ &=\; \big\{(\mathsf{Prop}, \mathsf{Prop});\ (\mathsf{Type}, \mathsf{Prop});\ (\mathsf{Type}, \mathsf{Type});\ (\mathsf{Kind}, \mathsf{Type})\big\}. \end{aligned}$$

Les systèmes de types purs λU et λU^- sont des PTS fonctionnels [26], d'où il ressort que :

Proposition 8.1.3 (Unicité du type) — *Dans un contexte donné, le type d'un terme du système U (resp. du système U^-) est unique à β-conversion près :*

$$\Gamma \vdash M : T \quad et \quad \Gamma \vdash M : T' \qquad \Rightarrow \qquad T =_\beta T'.$$

Une autre caractéristique des PTS fonctionnels est que dans ces systèmes, la règle de renforcement est admissible [26] :

Proposition 8.1.4 (Renforcement) — *Si $\Gamma_1; [x_0 : T_0]; \Gamma_2 \vdash M : T$ est un jugement dérivable dans le système U (resp. le système U^-) et si x_0 n'apparaît ni dans le contexte Γ_2, ni dans les termes M et T, alors le jugement $\Gamma_1; \Gamma_2 \vdash M : T$ est dérivable dans le système U (resp. le système U^-).*

Le système U^- est clairement un sous-système du système U, en ce sens que chaque dérivation de typage construite dans le système U^- est également une dérivation de typage bien formée dans le système U. Pour cette raison, nous nous contenterons d'étudier les propriétés du système U dans la suite de cette section, en précisant à chaque fois les restrictions qu'il convient d'effectuer pour obtenir le résultat analogue dans le cadre du système U^-.

8.1.2 Stratification des termes

Dans un PTS, on appelle *sorte maximale* (*top-sort* en anglais) toute sorte s telle que $(s : s') \notin \mathbf{Axiom}$ quelle que soit la sorte s'. Un résultat classique de la théorie des PTS est que pour tout jugement dérivable $\Gamma \vdash M : T$, soit T est une sorte maximale (qui est par définition mal typée), soit il existe une sorte s telle que le jugement $\Gamma \vdash T : s$ est dérivable.

Dans le système U, la seule sorte maximale est la sorte Kind. Par conséquent, si $\Gamma \vdash M : T$ est un jugement dérivable dans le système U et si $T \not\equiv \mathsf{Kind}$, alors une et une seule des trois assertions suivantes est satisfaite :

- $\Gamma \vdash T : \mathsf{Kind}$, auquel cas[4] on dit que M est un *genre* dans le contexte Γ ;
- $\Gamma \vdash T : \mathsf{Type}$, auquel cas on dit que M est un *constructeur* dans le contexte Γ ;
- $\Gamma \vdash T : \mathsf{Prop}$, auquel cas on dit que M est un *terme de preuve* dans le contexte Γ.

Notons que ces trois cas sont disjoints en raison de la propriété d'unicité du type modulo β-conversion. Si $\Gamma = [x_1 : T_1; \ldots; x_n : T_n]$ est un contexte bien formé, il existe pour chaque déclaration $(x_i : T_i)$ figurant dans ce contexte une unique sorte s_i telle que $\Gamma \vdash T_i : s_i$. Lorsque $s_i \equiv \mathsf{Prop}$ (resp. $s_i \equiv \mathsf{Type}$, $s_i \equiv \mathsf{Kind}$), on dit que x_i est une *variable de terme de preuve* (resp. *variable de constructeur*, *variable de genre*).

Dans la suite de ce paragraphe, on notera ξ, ξ' les variables de termes de preuves, x, y, z les variables de constructeurs et α, β les variables de genre. De même, on utilisera les lettres t, u pour désigner les termes de preuves, les lettres M, N, ϕ, ψ pour désigner les constructeurs (les lettres ϕ et ψ étant réservées aux constructeurs de type Prop) ainsi que les lettres A, B pour désigner les genres.

[4] Nous montrerons un peu plus loin que dans ce cas, on a même $T \equiv \mathsf{Type}$, puisque Type est le seul objet de type Kind dans le système U (et dans le système U^-).

Proposition 8.1.5 (Stratification) — *Dans le système U, les jugements de typage dérivables sont de l'une des quatre formes suivantes*

(Type)	$\Gamma \vdash \mathsf{Type} : \mathsf{Kind}$	
(**Genre**)	$\Gamma \vdash A : \mathsf{Type}$	
(**Constructeur**)	$\Gamma \vdash M : A$	(*avec* $\Gamma \vdash A : \mathsf{Type}$)
(**Terme de preuve**)	$\Gamma \vdash t : \phi$	(*avec* $\Gamma \vdash \phi : \mathsf{Prop}$)

où A, M, ϕ et t sont construits à partir des règles syntaxiques indiquées par la figure 8.1.

Genres	A, B	::=	α | Prop		
		|	$A \to B$		(Type, Type)
		|	$\Pi\alpha \,.\, A$		(Kind, Type)
Constructeurs	M, N, ϕ, ψ	::=	x		
		|	$\phi \Rightarrow \psi$		(Prop, Prop)
		|	$\forall x : A \,.\, \phi$		(Type, Prop)
		|	$\overline{\forall}\alpha \,.\, \phi^{(*)}$		(Kind, Prop)
		|	$\lambda x : A \,.\, M$	| $M\ N$	(Type, Type)
		|	$\Lambda\alpha \,.\, M$	| $M\ A$	(Kind, Type)
Termes de preuves	t, u	::=	x		
		|	$\lambda\xi^{\phi}.\, t$	| $t\ u$	(Prop, Prop)
		|	$\lambda x : A \,.\, t$	| $t\ M$	(Type, Prop)
		|	$\Lambda\alpha \,.\, t^{(*)}$	| $t\ A^{(*)}$	(Kind, Prop)

Les constructions surmontées d'une astérisque $^{(*)}$ sont spécifiques au système U et n'existent pas dans le système U^-. Par ailleurs, nous avons indiqué à droite de chaque produit dépendant, chaque abstraction et chaque application la règle de formation de produit dépendant qui permet d'introduire cette construction.

FIG. 8.1 – Présentation stratifiée des systèmes U et U^-

Dans la figure 8.1 nous avons utilisé des symboles différents (Π, $\to$, $\forall$, $\Rightarrow$ et $\overline{\forall}$) pour représenter les différentes formes de produits dépendants et non-dépendants du système U. Le choix de ces notations est dicté par l'intuition selon laquelle Prop désigne la sorte des *propositions*, tandis que Type désigne la sorte des *types de données* (dont Prop fait partie d'après l'axiome $\mathsf{Prop} : \mathsf{Type}$). Les cinq règles de formation de produits dépendants données par l'ensemble $\mathbf{Rule}_U$ ont la signification suivante :

- La règle (Prop, Prop) ne permet de définir que des produits non-dépendants de la forme $\Pi\_ : \phi \,.\, \psi$, où ϕ et ψ sont des propositions. Ce produit non-dépendant $\Pi\_ : \phi \,.\, \psi$ qui est lui-même une proposition correspond à l'implication logique et est noté $\phi \Rightarrow \psi$.
- La règle (Type, Prop) permet de définir des produits dépendants de la forme $\Pi x : A \,.\, \phi$, où ϕ est une proposition dépendant (éventuellement) d'une variable de constructeur $x : A$. Ce produit dépendant correspond à la quantification universelle (d'ordre supérieur) de la proposition ϕ par la variable $x : A$ et est noté $\forall x : A \,.\, \phi$.

- La règle $(\mathsf{Kind}, \mathsf{Prop})$ permet de définir des produits dépendants de la forme $\Pi\alpha : \mathsf{Type} \,.\, \phi$, où ϕ est une proposition dépendant (éventuellement) d'une variable de genre $\alpha : \mathsf{Type}$. Ce produit dépendant correspond à la quantification universelle de la proposition ϕ par la variable $\alpha : \mathsf{Type}$ et est noté $\overline{\forall}\alpha \,.\, \phi$. Cette quantification universelle sur tous les types est spécifique au système U et n'existe pas dans le système U^-.
- La règle $(\mathsf{Type}, \mathsf{Type})$ ne permet de définir que des produits non-dépendants de la forme $\Pi\_ : A \,.\, B$, où A et B sont deux genres (types de données). Le produit non-dépendant $\Pi\_ : A \,.\, B$ est lui-même un type de données correspondant au type des fonctions de A dans B, qui est noté $A \to B$.
- Enfin, la règle $(\mathsf{Kind}, \mathsf{Type})$ permet de former des produits dépendants de la forme $\Pi\alpha : \mathsf{Type} \,.\, A$, où A est un genre dépendant de la variable de genre $\alpha : \mathsf{Type}$. Ce produit dépendant permet d'introduire au niveau des genres (types de données) le polymorphisme du système F, et est noté $\Pi\alpha \,.\, A$.

8.1.3 La logique des systèmes U et U^-

De la théorie des types simples de Church au système U^- D'un point de vue logique, le système U^- correspond à la théorie des types simples de Church (dans sa version minimale en logique intuitionniste) dans laquelle on a incorporé le polymorphisme du système F au niveau des types comme au niveau des termes. En effet :

- Les genres de λU^- correspondent aux types du système F auxquels on a ajouté une constante Prop représentant le type des propositions :

$$\textbf{Types (genres)} \qquad A, B \quad := \quad \alpha \quad | \quad \mathsf{Prop} \quad | \quad A \to B \quad | \quad \Pi\alpha \,.\, A$$

- Les constructeurs de λU^- correspondent aux termes du système F auxquels on a ajouté deux constructions spécifiques $\phi \Rightarrow \psi$ et $\forall x : A \,.\, \phi$ pour représenter l'implication et la quantification universelle :

$$\begin{array}{lrcl} \textbf{Termes (constructeurs)} & M, N, \phi, \psi & ::= & x \\ & & | & \lambda x : A \,.\, M \quad | \quad M\ N \\ & & | & \Lambda\alpha \,.\, M \quad | \quad M\ A \\ & & | & \phi \Rightarrow \psi \quad | \quad \forall x : A \,.\, \phi \end{array}$$

- Enfin, les termes de preuves de λU^- correspondent aux règles de déduction du calcul des prédicats intuitionniste d'ordre supérieur, basé sur l'implication et la quantification universelle :

$$\begin{array}{lrcll} \textbf{Termes de preuves} & t, u & ::= & \xi & & \text{(axiome)} \\ & & | & \lambda\xi^{\phi} . t \quad | \quad t\ u & & (\Rightarrow\text{-intro}, \Rightarrow\text{-élim}) \\ & & | & \lambda x : A \,.\, t \quad | \quad t\ M & & (\forall\text{-intro}, \forall\text{-élim}) \end{array}$$

Du système U^- au système U Une limitation du système U^- est que s'il est possible d'y définir des fonctions polymorphes, il n'est pas possible d'exprimer leurs propriétés d'une manière qui rende compte du polymorphisme. Par exemple, si id désigne le terme défini par

$$\mathsf{id} \quad : \quad \Pi\alpha \,.\, \alpha \to \alpha \quad := \quad \Lambda\alpha \,.\, \lambda x : \alpha \,.\, x$$

et si A désigne un type quelconque, il est possible d'exprimer le fait que $(\mathsf{id}\ A\ x)$ est égal à x pour tout x de type A

$$\forall x : A\,.\ \ (\mathsf{id}\ A\ x)\ =_A\ x$$

(en anticipant sur la définition du codage imprédicatif de l'égalité de Leibniz qui sera donné au paragraphe suivant), mais il n'est pas possible d'exprimer par une formule du langage que cette propriété est vraie *pour tout type* A.

C'est pour cette raison que le système U dans sa version complète comporte en plus des règles du système U^- une règle de formation de produits dépendants $(\mathsf{Kind}, \mathsf{Prop})$, qui ajoute aux constructeurs (termes) une construction $\overline{\forall}\alpha\,.\,\phi$ permettant de quantifier universellement une proposition sur tous les genres (types de données)

$$\textbf{Termes} \qquad M, N, \phi, \psi \quad ::= \quad \cdots \quad | \quad \overline{\forall}\alpha\,.\,\phi$$

et qui se traduit au niveau des termes de preuves par l'apparition des schémas d'introduction et d'élimination correspondants :

$$\textbf{Termes de preuves} \qquad t, u \quad ::= \quad \cdots \quad | \quad \Lambda\alpha\,.\,t \quad | \quad t\ A \qquad (\overline{\forall}\text{-intro},\ \overline{\forall}\text{-élim})$$

Inférence et vérification de type Ainsi que nous l'avons déjà mentionné ci-dessus, les genres du système U sont les types du système F auxquels on a ajouté une constante de type Prop, qui se comporte comme une variable de type particulière dont toutes les occurrences sont libres. De même, les constructeurs du système U sont les termes du système F auxquels on a ajouté trois constructions supplémentaires $\phi \Rightarrow \psi$ (règle $(\mathsf{Prop}, \mathsf{Prop})$), $\forall x : A\,.\,\phi$ (règle $(\mathsf{Type}, \mathsf{Prop})$) et $\overline{\forall}\alpha\,.\,\phi$ (règle $(\mathsf{Kind}, \mathsf{Prop})$).

De manière équivalente, on peut considérer que les constructions $\phi \Rightarrow \psi$, $\forall x : A\,.\,\phi$ et $\overline{\forall}\alpha\,.\,\phi$ peuvent être définies à partir de trois constantes primitives[5]

$$\begin{aligned}
(\Rightarrow) \quad &: \quad \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop} \\
(\forall) \quad &: \quad \Pi\alpha\,.\,(\alpha \to \mathsf{Prop}) \to \mathsf{Prop} \\
(\overline{\forall}) \quad &: \quad (\Pi\alpha\,.\,\mathsf{Prop}) \to \mathsf{Prop}
\end{aligned}$$

en posant

$$\begin{aligned}
\phi \Rightarrow \psi \quad &\equiv \quad (\Rightarrow)\ \phi\ \psi, \\
\forall x : A\,.\,\phi \quad &\equiv \quad (\forall)\ A\ (\lambda x : A\,.\,\phi) \\
\overline{\forall}\alpha\,.\,\phi \quad &\equiv \quad (\overline{\forall})\ (\Lambda\alpha\,.\,\phi).
\end{aligned}$$

On vérifie aisément que le passage de l'un à l'autre de ces deux points de vue constitue un isomorphisme vis-à-vis de la β-réduction (les constantes $(\Rightarrow)$, $(\forall)$ et $(\overline{\forall})$ étant inertes), ce qui suggère une traduction immédiate des genres et des constructeurs du système U dans les types et les termes du système F, d'où il ressort que :

Proposition 8.1.6 (Normalisation forte des constructeurs) — *Dans les systèmes U et U^-, tous les constructeurs sont des termes fortement normalisables.*

[5]En s'inspirant de la présentation habituelle de la théorie des types simples de Church, dans laquelle l'implication et la quantification universelle sont définies à l'aide de constantes primitives $(\Rightarrow)$ et $(\forall^A)$, de types respectifs $\mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$ et $(A \to \mathsf{Prop}) \to \mathsf{Prop}$ (où A décrit l'ensemble des types simples).

Dans ces deux systèmes, les types (c'est-à-dire les termes figurant à droite des jugements de typage) sont soit des constructeurs de type Prop (qui sont fortement normalisables d'après la proposition ci-dessus), soit des genres (qui ne contiennent aucun β-radical), soit les sortes Type ou Kind. Par conséquent, le test d'égalité intensionnelle entre deux types du système U est décidable, d'où il ressort de manière immédiate que :

Corollaire 8.1.7 (Décidabilité du typage) — *Dans les systèmes U et U^-, l'inférence de type et la vérification de type sont décidables.*

Notons que la propriété de normalisation forte évoquée ci-dessus n'est valable que pour les constructeurs (et les genres) du système U. Comme les systèmes U et U^- sont incohérents, il existe dans ces deux systèmes des termes de preuves qui n'ont pas de forme normale de tête d'après le lemme suivant :

Lemme 8.1.8 — *Dans le système U, les termes de preuves clos dont le type est la proposition $\bot \equiv \forall a : \mathsf{Prop} \,.\, a$ n'admettent pas de forme normale de tête.*

Preuve. Par une disjonction de cas immédiate sur la forme d'un terme de preuve clos de type $\bot \equiv \forall a : \mathsf{Prop} \,.\, a$ en forme normale de tête. □

Les sortes imprédicatives Prop et Type Ainsi que nous l'avons déjà mentionné dans l'introduction de ce chapitre, les systèmes U et U^- sont tous les deux des systèmes incohérents dans lesquels il est possible de prouver n'importe quelle proposition. L'incohérence vient de ce que le système U^- (et le système U qui le contient) comporte deux sortes imprédicatives Prop et Type hiérarchisées par l'axiome $\mathsf{Prop} : \mathsf{Type}$.

- La règle $(\mathsf{Kind}, \mathsf{Type})$, en conjonction avec l'axiome $\mathsf{Type} : \mathsf{Kind}$, permet de former le produit dépendant imprédicatif $\Pi\alpha \,.\, A$ ($\equiv \Pi\alpha : \mathsf{Type} \,.\, A$) qui introduit le polymorphisme du système F au niveau des types de données (genres) et des termes (constructeurs). Nous parlerons ici d'*imprédicativité fonctionnelle*, car c'est cette forme d'imprédicativité qui s'exprime dans le langage des termes ainsi que dans leurs types.
- La règle $(\mathsf{Type}, \mathsf{Prop})$, en conjonction avec l'axiome $\mathsf{Prop} : \mathsf{Type}$, permet de former la quantification universelle imprédicative $\forall a : \mathsf{Prop} \,.\, \phi$ ($\equiv \Pi a : \mathsf{Prop} \,.\, \phi$), où ϕ est une proposition dépendant d'une variable propositionnelle $a : \mathsf{Prop}$. Cette forme d'imprédicativité est appelée *imprédicativité propositionnelle*, car c'est elle qui s'exprime dans le langage des propositions et dans le langage des preuves qui lui correspond. Notons que cette forme d'imprédicativité se rencontre déjà dans la théorie des types simples de Church, qui est une théorie cohérente (et dont le système de preuves est isomorphe au système de types purs $F\omega$).

La conjonction des axiomes $\mathsf{Type} : \mathsf{Kind}$, $\mathsf{Prop} : \mathsf{Type}$ et des deux règles de formation de produits dépendants $(\mathsf{Kind}, \mathsf{Type})$ et $(\mathsf{Type}, \mathsf{Prop})$ est cruciale pour établir l'incohérence du système U^-, puisque si l'on retire au système U^- l'un de ces deux axiomes ou l'une de ces deux règles, on obtient un système cohérent. En effet :

- Dans le système U^- privé de l'axiome $\mathsf{Type} : \mathsf{Kind}$, il n'est plus possible de former le moindre type dans la sorte Kind. Par conséquent, la règle $(\mathsf{Kind}, \mathsf{Type})$ n'a plus d'effet, et les seuls genres, constructeurs et termes de preuves qu'il est possible de former sont ceux du système $F\omega$, qui est cohérent et dont tous les termes sont fortement normalisables.

- Dans le système U^- privé de l'axiome $\mathsf{Prop} : \mathsf{Type}$ (qu'il est alors nécessaire de remplacer par l'axiome $\mathsf{Prop} : \mathsf{Kind}$), il est possible d'identifier les sortes Prop et Type, et de traduire de manière immédiate tous les termes de ce système dans le système F, en envoyant Kind sur Type, Type sur Prop et Prop sur lui-même. Là encore, le système obtenu est cohérent et tous ses termes sont fortement normalisables.
- Si l'on retire au système U^- la règle de formation de produits dépendants $(\mathsf{Kind}, \mathsf{Type})$, on obtient un sous-système du Calcul des Constructions avec deux univers (et plus précisément de la théorie des types de Church avec deux univers), qui est un système cohérent et dont tous les termes sont fortement normalisables.
- Enfin, si l'on retire la règle $(\mathsf{Type}, \mathsf{Prop})$ (tout en conservant la règle $(\mathsf{Kind}, \mathsf{Type})$), on obtient un système plus "exotique" dans lequel il n'est pas possible de former le moindre terme de type Prop dans le contexte vide en raison de la disparition de la quantification universelle $\forall x : A \,.\, \phi$. Intuitivement, la partie logique de ce système correspond à la logique minimale intuitionniste et sa partie fonctionnelle, qui est indépendante, correspond au système F muni de la constante de type Prop. Là encore, il est facile de vérifier qu'un tel système est cohérent.

8.1.4 Connecteurs, quantificateurs et égalité de Leibniz

Connecteurs logiques dans le système U^- Le système U^- contient de manière immédiate la logique intuitionniste minimale d'ordre supérieur de la théorie des types simples de Church. Bien que dans cette logique, l'implication $\phi \Rightarrow \psi$ et la quantification universelle $\forall x : A \,.\, \phi$ soient les seules constructions primitives, il est possible de recourir aux codages impédicatifs standards pour définir les autres connecteurs de la logique intuitionniste, qui sont donnés par

$$
\begin{array}{lcll}
\top & \equiv & \forall a : \mathsf{Prop} \,.\, a \Rightarrow a & \\
\bot & \equiv & \forall a : \mathsf{Prop} \,.\, a & \\
\neg\phi & \equiv & \phi \Rightarrow \bot & \\
\phi \wedge \psi & \equiv & \forall a : \mathsf{Prop} \,.\, (\phi \Rightarrow \psi \Rightarrow a) \Rightarrow a & (a \notin FV(\phi) \cup FV(\psi)) \\
\phi \vee \psi & \equiv & \forall a : \mathsf{Prop} \,.\, (\phi \Rightarrow a) \Rightarrow (\psi \Rightarrow a) \Rightarrow a, & (a \notin FV(\phi) \cup FV(\psi))
\end{array}
$$

où ϕ et ψ désignent des propositions arbitraires formées dans un même contexte. Ces connecteurs satisfont les schémas d'introduction et d'élimination correspondants, en ce sens que les propositions suivantes

$$
\begin{array}{lll}
(\top\text{-intro}) & & \top \\
(\bot\text{-élim}) & \forall a : \mathsf{Prop} \,. & \bot \Rightarrow a \\
(\wedge\text{-intro}) & \forall a, b : \mathsf{Prop} \,. & a \Rightarrow b \Rightarrow a \wedge b \\
(\wedge\text{-élim}_1) & \forall a, b : \mathsf{Prop} \,. & a \wedge b \Rightarrow a \\
(\wedge\text{-élim}_2) & \forall a, b : \mathsf{Prop} \,. & a \wedge b \Rightarrow b \\
(\vee\text{-intro}_1) & \forall a, b : \mathsf{Prop} \,. & a \Rightarrow a \vee b \\
(\vee\text{-intro}_2) & \forall a, b : \mathsf{Prop} \,. & b \Rightarrow a \vee b \\
(\vee\text{-elim}) & \forall a, b, c : \mathsf{Prop} \,. & (a \Rightarrow c) \Rightarrow (b \Rightarrow c) \Rightarrow a \vee b \Rightarrow c
\end{array}
$$

sont prouvables de manière immédiate dans le système U^-.

Quantificateur existentiel Soit A un type de données (*i.e.* un genre) du système U^-. La quantification existentielle intuitionniste $\exists x : A \,.\, \phi$ est définie par

$$\exists x : A \,.\, \phi \quad \equiv \quad \forall a : \mathsf{Prop} \,.\, (\forall x : A \,.\, \phi \Rightarrow a) \Rightarrow a \qquad (a \notin FV(\phi))$$

pour toute proposition ϕ dépendant (éventuellement) d'une variable x de type A. Il est alors immédiat que les schémas d'introduction et d'élimination donnés par

$$\begin{array}{ll} (\exists\text{-intro}) & \forall \phi : A \to \mathsf{Prop} \,.\, \forall x_0 : A \,. \quad (\phi\ x_0) \ \Rightarrow\ \exists x : A \,.\, (\phi\ x) \\ (\exists\text{-élim}) & \forall \phi : A \to \mathsf{Prop} \,.\, \forall c : \mathsf{Prop} \,. \quad (\forall x : A \,.\, (\phi\ x) \Rightarrow c) \ \Rightarrow\ (\exists x : A \,.\, (\phi\ x)) \ \Rightarrow\ c \end{array}$$

sont prouvables dans le système U^-.

Égalité de Leibniz Soit A un type (genre) du système U^-. Pour tous termes M_1 et M_2 de type A, on note $M_1 =_A M_2$ la proposition définie par

$$M_1 =_A M_2 \quad \equiv \quad \forall \phi : A \to \mathsf{Prop} \,.\, (\phi\ M_1) \Rightarrow (\phi\ M_2) \qquad (\phi \notin FV(M_{1,2}))$$

Cette définition valide de manière immédiate les schémas d'introduction et d'élimination de l'égalité de Leibniz :

$$\begin{array}{ll} (=_A\text{-intro}) & \forall x : A \,. \qquad\qquad\qquad\quad x =_A x \\ (=_A\text{-élim}) & \forall \phi : A \to \mathsf{Prop} \,.\, \forall x, y : A \,. \quad (\phi\ x) \ \Rightarrow\ x =_A y \ \Rightarrow\ (\phi\ y). \end{array}$$

Constructions spécifiques au système U Toutes les constructions que nous venons de définir demeurent bien entendu valables dans le système U. Comme dans ce système on dispose en outre d'une construction spécifique $\overline{\forall}\alpha \,.\, \phi$ permettant de quantifier universellement une formule ϕ sur tous les types de données α : Type, il est également possible de définir la quantification existentielle correspondante $\overline{\exists}\alpha \,.\, \phi$ en posant

$$\overline{\exists}\alpha \,.\, \phi \quad \equiv \quad \forall a : \mathsf{Prop} \,.\, (\overline{\forall}\alpha \,.\, \phi \Rightarrow a) \Rightarrow a \qquad (a \notin FV(\phi))$$

pour toute proposition ϕ dépendant (éventuellement) de la variable de type α : Type. Il est alors immédiat que les schémas d'introduction et d'élimination

$$\begin{array}{ll} (\overline{\exists}\text{-intro}) & \forall \phi : (\Pi\alpha \,.\, \mathsf{Prop}) \,.\, \overline{\forall}\alpha_0 \,. \qquad (\phi\ \alpha_0) \ \Rightarrow\ \overline{\exists}\alpha \,.\, (\phi\ \alpha) \\ (\overline{\exists}\text{-élim}) & \forall \phi : (\Pi\alpha \,.\, \mathsf{Prop}) \,.\, \forall c : \mathsf{Prop} \,. \quad (\overline{\forall}\alpha \,.\, (\phi\ \alpha) \Rightarrow c) \ \Rightarrow\ (\overline{\exists}\alpha \,.\, (\phi\ \alpha)) \ \Rightarrow\ c \end{array}$$

sont prouvables dans le système U. De même, il est possible de définir une égalité de Leibniz $A = B$ sur les types en posant

$$A = B \quad \equiv \quad \forall \phi : (\Pi\alpha \,.\, \mathsf{Prop}) \,.\, (\phi\ A) \Rightarrow (\phi\ B), \qquad (\phi \notin FV(A) \cup FV(B))$$

dont on vérifie de manière immédiate qu'elle satisfait les schémas d'introduction et d'élimination donnés par les propositions :

$$\begin{array}{ll} (=\text{-intro}) & \overline{\forall}\alpha \,. \qquad\qquad\qquad\quad \alpha = \alpha \\ (=\text{-élim}) & \forall \phi : (\Pi\alpha \,.\, \mathsf{Prop}) \,.\, \overline{\forall}\alpha, \beta \,. \quad (\phi\ \alpha) \ \Rightarrow\ \alpha = \beta \ \Rightarrow\ (\phi\ \beta). \end{array}$$

8.2 Représentation des ensembles en théorie des types

Dans cette section, nous abandonnons (temporairement) le cadre des systèmes U et U^- pour nous intéresser aux différentes possibilités permettant de représenter les ensembles en théorie des types. Pour cela, nous nous plaçons ici dans un cadre informel qui nous permettra d'envisager la représentation des ensembles dans des formalismes tels que le langage Caml [41] ou le Calcul des Constructions Inductives [63, 53].

8.2.1 La représentation naïve

En théorie des types, les ensembles d'entiers, de réels, etc. sont généralement représentés par des prédicats définis sur le type correspondant (*i.e.* le type des entiers, des réels, etc.). Une telle représentation correspond à la conception naïve de la théorie des ensembles, selon laquelle les objets mathématiques sont soit des objets élémentaires — tels que les nombres entiers ou les nombres réels par exemple — soit des ensembles d'objets de même type, c'est-à-dire des ensembles d'entiers, des ensembles de réels, ou même des ensembles d'ensembles d'entiers, des ensembles d'ensembles de réels, etc.

En théorie des ensembles, la seule notion primitive est la notion d'ensemble. Par conséquent, tout ensemble est un ensemble d'ensembles, dont les éléments sont encore des ensembles d'ensembles, et ainsi de suite. Si l'on désire représenter les ensembles (au sens de la théorie du même nom) en théorie des types, il est important d'abandonner le paradigme "ensemble = prédicat" et de passer à une représentation dans laquelle la relation d'appartenance est conçue dès le départ de manière récursive.

Par exemple, si on considère la représentation usuelle des entiers naturels en théorie des ensembles (due à von Neumann) dans laquelle chaque entier est l'ensemble des entiers strictement plus petits que lui, il est important de comprendre que la valeur de l'entier 4 n'est pas donnée par l'égalité

$$4 \quad = \quad \{0;\ 1;\ 2;\ 3\},$$

mais plutôt par l'égalité

$$4 \quad = \quad \left\{\{\};\ \{\{\}\};\ \left\{\{\};\ \{\{\}\}\right\};\ \left\{\{\};\ \{\{\}\};\ \left\{\{\};\ \{\{\}\}\right\}\right\}\right\}$$

(obtenue en dépliant récursivement les définitions de 0, 1, 2 et 3), qui illustre bien mieux la structure intime de cet ensemble.

8.2.2 Une définition incorrecte

Puisqu'en théorie des types, les ensembles sur X sont représentés par des prédicats de type $X \to \mathsf{Prop}$, il est naturel de chercher à représenter les ensembles "sur eux-mêmes" par les objets d'un type set qui serait caractérisé par l'égalité

$$\mathsf{set} \quad = \quad (\mathsf{set} \to \mathsf{Prop}),$$

c'est-à-dire par les objets d'un type qui serait égal au type de ses prédicats. Bien entendu, l'égalité ci-dessus ne doit pas être prise au pied de la lettre — c'est-à-dire dans un sens

intensionnel qui donnerait lieu à une définition manifestement mal fondée — mais plutôt dans le sens d'une égalité à isomorphisme près.

Dans un système de types inductifs, on peut envisager de définir set comme un type inductif dont les objets sont définis à partir d'un unique constructeur

$$\mathsf{fold} \quad : \quad (\mathsf{set} \to \mathsf{Prop}) \to \mathsf{set}$$

transformant n'importe quel prédicat sur set en un objet de type set. Avec une telle définition, chaque objet de type set est de la forme (fold f) où f est un prédicat sur set, et la relation d'appartenance entre deux ensembles x et y de type set peut être définie très simplement par :

$$x \in y \qquad \equiv \qquad \textbf{Cases } y \textbf{ of } (\mathsf{fold}\ f) \Rightarrow (f\ x) \textbf{ end}$$

(en adoptant la syntaxe du système Coq-V7.0). Cette définition pose néanmoins un léger problème, puisqu'elle permet manifestement de définir un ensemble

$$R \quad \equiv \quad \mathsf{fold}\big(\lambda x : \mathsf{set}\,.\, x \notin x\big)$$

dont on montre aisément qu'il satisfait la relation

$$R \in R \qquad \Leftrightarrow \qquad R \notin R,$$

d'où l'on tire une incohérence manifeste.

Bien entendu, l'incohérence résulte du fait que la condition de stricte positivité [53] n'est pas satisfaite par le type du constructeur fold (dont le type de l'argument comporte une occurrence négative de la constante set) ; et c'est précisément pour cette raison que la définition du type inductif set que nous venons de donner est rejetée systématiquement par les systèmes de traitement de démonstrations basés sur la théorie des types, tels que les systèmes Alf [44], LEGO [43] et Coq [11].[6]

8.2.3 Représentation des ensembles héréditairement finis

Ensembles héréditairement finis en ML Dans le paragraphe 8.2.1 nous avons pu représenter *in extenso* le contenu de l'entier de von Neumann 4 parce que celui-ci est un ensemble bien fondé héréditairement fini (*cf* paragraphe 3.4.3). En ML, les ensembles finis d'objets de type `t` (où `t` désigne un type quelconque) peuvent être représentés par des listes d'objets de type `t`, c'est-à-dire par le type `t list` (en adoptant ici la syntaxe de Caml). En reprenant l'idée du paragraphe précédent, il est naturel de représenter le type des ensembles héréditairement finis en posant la définition de type récursive[7]

```
type set = set list.
```

[6] On remarquera que l'évaluation (infinie) du terme $R \in R$ est très similaire à celle du terme $(\lambda x.xx)(\lambda x.xx)$ du λ-calcul pur, à cette différence près que le premier terme ne cesse d'empiler négation sur négation là où le second se contente de reproduire sans cesse sa propre structure. On retrouvera le même phénomène pour la preuve du paradoxe que nous construirons au paragraphe 8.4.4.

[7] Cette définition est acceptée en Objective Caml (version 3.00) avec l'option de compilation `-rectypes`.

qui permet de représenter par exemple les entiers de von Neumann par les listes suivantes :

$$
\begin{array}{rcl}
0 & \equiv & \texttt{[]} \\
1 & \equiv & \texttt{[[]]} \\
2 & \equiv & \texttt{[[] ; []]} \\
3 & \equiv & \texttt{[[] ; []] ; [[] ; []]]} \\
4 & \equiv & \texttt{[[] ; []] ; [[] ; []]] ; [[] ; []] ; [[] ; []]]]} \\
 & \ldots &
\end{array}
$$

Ensembles héréditairement finis en théorie des types Contrairement à la définition récursive du paragraphe précédent, la définition ci-dessus correspond à un type inductif légal en théorie des types, et que l'on peut définir par exemple dans le système Coq en écrivant :

$$
\begin{array}{l}
\textbf{Inductive}\ \mathsf{set}\ :\ \mathsf{Type}_1\ := \\
|\ \ \mathsf{nil}\ \ :\ \mathsf{set} \\
|\ \ \mathsf{cons}\ :\ \mathsf{set} \rightarrow \mathsf{set} \rightarrow \mathsf{set}.
\end{array}
$$

Cette définition correspond à la définition habituelle du type des listes, à cette différence près qu'on a ici identifié le type des listes (qu'on est en train de définir) avec le type de ses éléments. De manière équivalente, ce type de données est un type d'arbres binaires dans lequel on a assigné aux deux arguments du constructeur cons des rôles dissymétriques, puisque le terme ($\mathsf{cons}\ x\ y$) représente l'ensemble $\{x\} \cup y$ et non la paire $\{x;\ y\}$.

Le problème de l'égalité extensionnelle Le codage des ensembles héréditairement finis par des listes récursives (ou arbres binaires) pose d'emblée le problème de la définition de l'égalité extensionnelle.

Dans le cas non-récursif — c'est-à-dire lorsque les types `t` et `t list` ne sont pas reliés par l'équation `t = t list` — un même ensemble (par exemple l'ensemble $\{a; b; c\}$) peut être représenté par des listes différentes (par exemple, les listes `[a ; b ; c]` et `[b ; a ; c ; a]`), qui diffèrent par le nombre d'occurrences d'un même élément et/ou l'ordre dans lequel ces éléments sont rangés. Pour tester l'égalité des ensembles représentés par deux listes, il suffit donc de vérifier que chaque élément de chacune des deux listes apparaît également dans l'autre liste, et vice-versa :

```
let equal a b =
  List.for_all (fun x -> List.exists (fun y -> x=y) b) a &&
  List.for_all (fun y -> List.exists (fun x -> x=y) a) b ;;
```

Dans le cas des ensembles héréditairement finis, le type de base (`set`) est égal au type de ses listes (`set list`), et un même élément peut apparaître dans les deux listes à comparer sous deux représentations différentes, et cela de manière récursive. Par conséquent, l'égalité entre ensembles héréditairement finis est maintenant une fonction récursive, dont le code Caml est donné par :[8]

```
let rec equal a b =
  List.for_all (fun x -> List.exists (fun y -> equal x y) b) a &&
  List.for_all (fun y -> List.exists (fun x -> equal x y) a) b ;;
```

[8] La complexité de cet algorithme naïf est au pire en $O(2^p \cdot n^{2p})$, où n désigne le branchement maximal des deux ensembles à comparer, et où p désigne le minimum des profondeurs de chacun de ces deux ensembles.

La fonction testant l'appartenance s'écrit alors :

```
let element a b = List.exists (fun y -> equal a y) b ;;
```

(On pourra vérifier que ces deux fonctions écrites en Caml se traduisent de manière très naturelle en théorie des types, et que la fonction testant l'égalité de deux ensembles héréditairement finis est définie par une récursion bien fondée.)

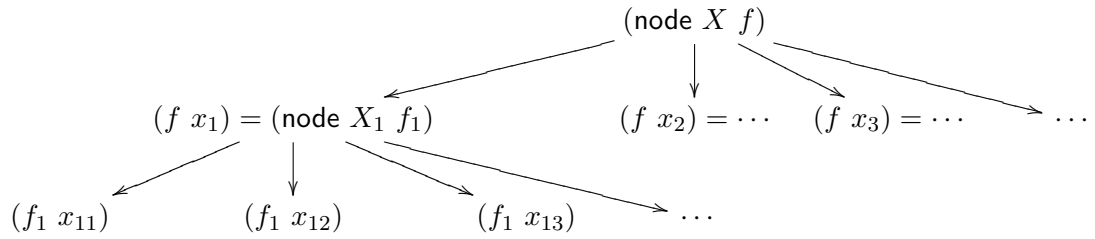
8.2.4 Représentation des ensembles par des arbres bien fondés

Le codage des ensembles par des listes récursives ne permet de représenter que des ensembles héréditairement finis. Une manière standard de représenter les ensembles infinis en théorie des types [64, 5] est de remplacer le type des listes récursives par un type d'arbres à branchement infini suffisamment grand. Dans le Calcul des Constructions Inductives par exemple, le type des ensembles peut être défini par

$$\begin{array}{l}\textbf{Inductive}\ \ \mathsf{set}\ :\ \mathsf{Type}_2\ :=\\ |\quad \mathsf{node}\quad :\quad \Pi X : \mathsf{Type}_1\,.\,(X \to \mathsf{set}) \to \mathsf{set}.\end{array}$$

(en adoptant une indexation explicite des univers prédicatifs pour faire ressortir les différences de niveau).

Intuitivement, cette définition inductive construit un type d'arbres dont les éléments sont de la forme $(\mathsf{node}\ X\ f)$, où X est un type quelconque (de niveau Type_1) et où $f : X \to \mathsf{set}$ représente la famille $(f(x))_{x\in X}$ des sous-arbres directs de l'arbre $(\mathsf{node}\ X\ f)$. Il est important que dans le terme $(\mathsf{node}\ X\ f)$ figure non seulement la fonction f, mais aussi le type X sur lequel cette fonction est définie, puisque celui-ci peut également varier en fonction du branchement de l'ensemble qu'on est en train de construire :



Remarquons également que la sorte du type set (ici Type_2) se situe un niveau au-dessus de la sorte des types définissant le branchement des nœuds des arbres (ici, la sorte Type_1). Il est clair que si ces sortes étaient au même niveau (ce qui serait refusé par le système Coq), il serait possible de construire un arbre représentant l'ensemble paradoxal de Russell.[9]

L'égalité extensionnelle En Coq, la relation d'égalité extensionnelle entre deux ensembles se définit par une récurrence bien fondée sur ses arguments (le choix de l'argument récursif

[9] En effet, si on avait $\mathsf{set} : \mathsf{Type}_1$, on pourrait définir dans la sorte Type_1 le type somme $X \equiv \Sigma x : \mathsf{set}\,.\,x \notin x$ (en supposant définie la relation d'appartenance $(\in) : \mathsf{set} \to \mathsf{set} \to \mathsf{Prop}$), et définir ensuite l'ensemble paradoxal de Russell en posant $R \equiv (\mathsf{node}\ X\ (\lambda p : X\,.\,\pi_1(p)))$, où $\pi_1(p)$ désigne la première projection de la paire dépendante $p : \Sigma x : \mathsf{set}\,.\,x \notin x$.

est ici arbitraire) :

$$\begin{array}{l}\textbf{Fixpoint}\ \ \mathsf{equal}\ \ [a : \mathsf{set};\ b : \mathsf{set}]\ :\ \mathsf{Prop}\ :=\\ \quad \textbf{Cases}\ a\ \textbf{of}\ (\mathsf{node}\ X\ f)\ \mapsto\\ \quad\quad \textbf{Cases}\ b\ \textbf{of}\ (\mathsf{node}\ Y\ g)\ \mapsto\\ \quad\quad\quad \big(\forall x : X\,.\,\exists y : Y\,.\,(\mathsf{equal}\ (f\ x)\ (g\ y))\big)\quad \wedge\\ \quad\quad\quad \big(\forall y : Y\,.\,\exists x : X\,.\,(\mathsf{equal}\ (f\ x)\ (g\ y))\big)\\ \quad\quad \textbf{end}\\ \quad \textbf{end}.\end{array}$$

Notons que ce programme est très similaire à la fonction `equal` dont le code ML a été donné dans le paragraphe précédent, à cette différence près que la valeur de retour n'est plus un booléen, mais une proposition exprimant l'égalité des deux ensembles a et b :

$$\mathsf{equal}\quad :\quad \mathsf{set} \to \mathsf{set} \to \mathsf{Prop}.$$

Bien entendu, il n'est plus possible d'exprimer l'égalité par un booléen, puisque la définition de la proposition ($\mathsf{equal}\ a\ b$) utilise des quantifications universelles et existentielles sur chaque type de branchement X figurant dans un sous-arbre de a ou de b, qui peut être un type infini. (Il est par ailleurs clair que puisque l'égalité entre ensembles infinis est indécidable, cette égalité ne peut pas être calculée par une fonction de type $\mathsf{set} \to \mathsf{bool}$ définissable dans le système.)

De manière analogue au paragraphe précédent, il est possible de définir en Coq la relation d'appartenance sur le type set

$$\mathsf{element}\quad :\quad \mathsf{set} \to \mathsf{set} \to \mathsf{Prop}$$

en posant :

$$\begin{array}{l}\textbf{Definition}\ \ \mathsf{element}\ \ [a : \mathsf{set};\ b : \mathsf{set}]\ :\ \mathsf{Prop}\ :=\\ \quad \textbf{Cases}\ b\ \textbf{of}\ (\mathsf{node}\ Y\ g)\ \mapsto\\ \quad\quad \exists y : Y\,.\,(\mathsf{equal}\ a\ (g\ y))\\ \quad \textbf{end}.\end{array}$$

On vérifie aisément que la relation d'appartenance ainsi définie est extensionnelle, en ce sens que la proposition

$$\forall x, y : \mathsf{set}\,.\qquad \big(\forall z : \mathsf{set}\,.\quad (\mathsf{element}\ z\ x)\ \Leftrightarrow\ (\mathsf{element}\ z\ y)\big)\quad \Rightarrow\quad (\mathsf{equal}\ x\ y)$$

est prouvable dans le système. Plus généralement, il est possible de montrer que les relations `equal` et `element` satisfont tous les axiomes et schémas d'axiomes de la théorie des ensembles constructive CZF [1, 2, 3].

8.3 Représentation des ensembles par des graphes pointés

Un des inconvénients de la représentation des ensembles par des arbres bien fondés est qu'elle repose sur un mécanisme de définitions inductives primitives qui n'existe pas dans les systèmes de types purs. Dans ce qui suit, nous adopterons une autre approche qui consiste à représenter un ensemble non pas par un arbre, mais par un graphe pointé, c'est-à-dire par un triplet (X, A, a) dont les composantes sont :

– un type X quelconque ;
– une relation binaire $A : X \to X \to \mathsf{Prop}$;
– un objet a de type X.

Ici, X est le type des sommets du graphe, $A(x, y)$ est la relation définissant les arcs du graphe, et $a : X$ est un sommet particulier, appelé *racine* du graphe pointé (X, A, a).

La structure de graphe pointé est plus générale que la structure d'arbre bien fondé, puisque s'il est possible de représenter n'importe quel arbre bien fondé par un graphe, la plupart des graphes n'ont pas de structure arborescente et peuvent même comporter des cycles. Du point de vue de la théorie des ensembles modélisée, ce changement de représentation permet ainsi de passer d'une théorie dans laquelle tous les ensembles sont bien fondés (*cf* paragraphe 3.4.3) à une théorie des ensembles plus générale dans laquelle on autorise également l'existence d'ensembles mal fondés.

8.3.1 Graphes pointés et hyper-ensembles

La représentation des ensembles par les graphes pointés est à la base de la théorie des *hyper-ensembles* [4], qui est une extension de la théorie des ensembles de Zermelo-Fränkel[10] dans laquelle on a ajouté un *axiome d'anti-fondation* qui peut s'énoncer ainsi :

Axiome 8.3.1 (Anti-fondation) — *Pour tout graphe $G = (I, A)$ dont I est l'ensemble des sommets et $A \subset (I \times I)$ l'ensemble des arcs, il existe une unique famille d'ensembles $(x_i)_{i \in I}$ étiquetant les sommets du graphe G telle qu'on ait*

$$x_i \quad = \quad \{x_j; \quad (j, i) \in A\}$$

pour tout sommet $i \in I$.

Dans ce qui suit, on dira qu'une famille d'ensembles $(x_i)_{i \in I}$ est une *réification* du graphe $G = (I, A)$ si on a l'égalité

$$x_i \quad = \quad \{x_j; \quad (j, i) \in A\}$$

pour tout $i \in I$. L'axiome d'anti-fondation exprime non seulement que tout graphe G peut être réifié en une famille d'ensembles indicée par les sommets du graphe G, mais il exprime également que cette réification est unique.

Pertinence de l'axiome d'anti-fondation Il est important de remarquer que lorsque le graphe $G = (I, A)$ est bien fondé (c'est-à-dire lorsque tous ses sommets sont accessibles par la relation $(j, i) \in A$), il n'est pas nécessaire de recourir à l'axiome d'anti-fondation pour démontrer l'existence et l'unicité de la réification de G, qui est dans ce cas une conséquence des axiomes standards de la théorie des ensembles de Zermelo-Fränkel. (L'existence et l'unicité se prouvent tous les deux par une récurrence bien fondée sur l'accessibilité des sommets du graphe G.)

En revanche, lorsque le graphe G n'est pas bien fondé — ce qui est le cas lorsque G contient des cycles par exemple — il n'est pas possible de démontrer ou de réfuter l'existence (resp. l'unicité) de la réification du graphe G. Autrement dit, les énoncés

[10] Il est utile de rappeler ici que la théorie de Zermelo-Fränkel [39] ne comprend pas d'axiome de la fondation.

– "G admet une réification" (existence) et
– "toutes les réifications de G sont égales" (unicité)
sont des énoncés indécidables dans ZF dès lors que le graphe G n'est pas bien fondé.

Pour résoudre ce problème d'indécidabilité, on peut alors adopter l'une des deux stratégies suivantes, qui s'excluent mutuellement :

- La première stratégie consiste à interdire toute possibilité de réifier un graphe qui ne serait pas bien fondé, puisqu'une telle réification entraînerait manifestement l'existence d'ensembles mal fondés dans la théorie. Pour cela, on ajoute à la théorie un *axiome de la fondation* (*cf* paragraphe 3.4.3), qui exprime que tous les ensembles sont bien fondés, et dont une conséquence immédiate est que les graphes mal fondés n'admettent pas de réification (ces deux énoncés sont même équivalents).
- La seconde stratégie consiste au contraire à étendre le résultat d'existence et d'unicité de la réification (qui est un théorème de ZF dans le cas des graphes bien fondés) à tous les graphes en posant l'axiome d'anti-fondation, qui permet ainsi de réifier n'importe quel graphe, même mal fondé.

Il est important de comprendre que l'axiome d'anti-fondation fait bien plus que réfuter le principe de bonne fondation : il donne à la fois un procédé permettant de construire des ensembles mal fondés (par réification de graphe), mais il permet également de montrer des égalités entre ensembles mal fondés (en utilisant la propriété d'unicité de la réification) qu'il serait impossible d'établir en utilisant seulement l'axiome d'extensionnalité.

Afin d'illustrer ce point, nous allons maintenant donner quelques exemples d'utilisation de l'axiome d'anti-fondation en théorie des hyper-ensembles.

8.3.2 Existence et unicité de l'atome Ω

Un cas typique d'utilisation de l'axiome d'anti-fondation est le suivant. En théorie des ensembles, on appelle *atome* tout ensemble x qui est son propre singleton, c'est-à-dire tel que $x = \{x\}$. Dans ZF, il n'est pas possible de prouver ou de réfuter l'existence des atomes, à moins d'ajouter l'axiome de la fondation qui en réfute clairement l'existence. Par ailleurs, la relation d'égalité entre atomes est problématique, puisque si $x = \{x\}$ et $y = \{y\}$, l'axiome d'extensionnalité ne nous donne que des équivalences de la forme

$$x = y \quad \Leftrightarrow \quad \{x\} = \{y\} \quad \Leftrightarrow \quad x = y \quad \Leftrightarrow \quad \{x\} = \{y\} \quad \Leftrightarrow \quad \cdots$$

qui ne permettent pas de conclure quoi que ce soit.

Dans la théorie des hyper-ensembles en revanche, il est possible de démontrer non seulement qu'il existe un atome, mais encore que cet atome est unique. Pour cela, on considère un graphe $G = (I, A)$ à un seul sommet et un seul arc défini par

$$G \quad = \quad i_0 \bullet \circlearrowleft \qquad\qquad \begin{array}{rcl} I & = & \{i_0\} \\ A & = & \{(i_0, i_0)\} \end{array}$$

où i_0 est un objet arbitraire de la théorie. D'après l'axiome d'anti-fondation, le graphe G admet une unique réification, qui est une famille d'ensembles $(x_i)_{i \in \{i_0\}}$ dont l'unique élément x_{i_0} satisfait par définition

$$x_{i_0} \quad = \quad \{x_j; \quad (j, i_0) \in A\} \quad = \quad \{x_{i_0}\}.$$

Réciproquement, considérons un atome quelconque $y = \{y\}$. Il est facile de vérifier que la famille $(y_i)_{i\in\{i_0\}}$ dont l'unique élément y_{i_0} est défini par $y_{i_0} = y$ est une réification du graphe G, puisque par hypothèse $y_{i_0} = \{y_{i_0}\}$. Comme d'après l'axiome d'anti-fondation, la réification du graphe G est unique, on a donc $y = y_{i_0} = x_{i_0}$, ce qui montre l'unicité de l'atome, que l'on note généralement Ω.

Un graphe cyclique à deux sommets Considérons à présent deux ensembles x_1 et x_2 tels que $x_1 = \{x_2\}$ et $x_2 = \{x_1\}$. L'axiome d'anti-fondation permet de montrer l'existence d'une telle paire d'ensembles, simplement en réifiant le graphe à deux sommets

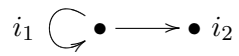
et en posant $x_1 = x_{i_1}$ et $x_2 = x_{i_2}$, où $(x_i)_{i\in\{i_1;i_2\}}$ est la réification du graphe ci-dessus.

On peut alors se demander ce qui distingue les objets x_1 et x_2, puisque ces deux ensembles sont définis par des équations parfaitement symétriques. La réponse est simple : puisque rien ne les distingue, ces deux ensembles sont forcément égaux !

Pour établir cette égalité, il suffit simplement d'échanger les rôles de x_1 et de x_2 en posant $y_{i_1} = x_2$ et $y_{i_2} = x_1$, et de remarquer que la famille $(y_i)_{i\in\{i_1;i_2\}}$ ainsi définie est une réification du graphe à deux sommets et deux arcs dessiné ci-dessus. D'après l'unicité de la réification, on a donc $x_1 = x_2 = \{x_1\} = \{x_2\}$, d'où il ressort clairement que $x_1 = x_2 = \Omega$.

Notons que nous aurions pu montrer que les deux ensembles x_1 et x_2 sont tous les deux égaux à Ω de manière directe, simplement en vérifiant que la famille constante $(z_i)_{i\in\{i_1;i_2\}}$ définie par $z_{i_1} = z_{i_2} = \Omega$ est une réification du même graphe.

Un ensemble mal fondé différent de Ω Pour terminer ce paragraphe consacré aux exemples, nous allons montrer l'existence d'un ensemble mal fondé différent de Ω. Cet ensemble est défini par réification du graphe suivant



Dans ce graphe ainsi que dans tous les graphes que nous dessinerons par la suite, on fait pointer les flèches des ensembles vers leurs éléments. Ainsi, une flèche allant d'un sommet i vers un sommet j représente l'arc $A(j, i)$, et signifie que $x_j \in x_i$ (où $(x_i)_{i\in I}$ désigne la réification du graphe en question).

Soit $(x_i)_{i\in\{i_1;i_2\}}$ la réification du graphe ci-dessus. Par définition, on a $x_{i_1} = \{x_{i_1}; x_{i_2}\}$ et $x_{i_2} = \varnothing$. Si on pose $x = x_{i_1}$, on a $x = \{x; \varnothing\}$, ce qui montre que l'ensemble x est mal fondé. Il est alors clair que les ensembles $x = \{x; \varnothing\}$ et $\Omega = \{\Omega\}$ sont distincts, puisque l'ensemble vide $\varnothing$ est un élément du premier mais pas du second.

8.3.3 Égalité et bissimulation

Ainsi que nous l'avons vu dans les paragraphes précédents, l'axiome d'anti-fondation associe à chaque graphe $G = (I, A)$ une famille d'ensembles $(x_i)_{i\in I}$ indicée par l'ensemble des sommets de G. Dans ce qui suit, on notera $\mathsf{res}(I, A)$ cette famille d'ensembles qui est caractérisée par

$$\mathsf{res}(I, A)_i \quad = \quad \big\{\mathsf{res}(I, A)_j; \quad (j, i) \in A\big\}$$

pour tout sommet $i \in I$. Cette correspondance entre graphes et familles d'ensembles s'étend naturellement aux graphes pointés en posant

$$\mathsf{res}(I, A, i_0) \quad = \quad \mathsf{res}(I, A)_{i_0}$$

pour tout graphe pointé (I, A, i_0).[11] Intuitivement, la racine du graphe pointé (I, A, i_0) sert à sélectionner un élément particulier de la famille obtenue par réification du graphe (I, A) sous-jacent.

Un résultat central de la théorie des hyper-ensembles [4] est la caractérisation de la relation d'égalité extensionnelle (définie sur la collection des ensembles) en termes de bissimulation entre graphes pointés :

Définition 8.3.2 (Bissimulation) — Soient (I, A, i_0) et (J, B, j_0) deux graphes pointés. On appelle *bissimulation* de (I, A, i_0) dans (J, B, j_0) toute relation $R \subset (I \times J)$ reliant les sommets de (I, A) à ceux de (J, B) qui satisfait les trois conditions suivantes :

1. pour tous sommets $i, i' \in I$ et pour tout sommet $j \in J$ tels que $A(i', i)$ et $R(i, j)$, il existe un sommet $j' \in J$ tel que $R(i', j')$ et $B(j', j)$;
2. pour tous sommets $j, j' \in J$ et pour tout sommet $i \in I$ tels que $B(j', j)$ et $R(i, j)$, il existe un sommet $i' \in I$ tel que $R(i', j')$ et $A(i', i)$;
3. $R(i_0, j_0)$.

Lorsqu'il existe une bissimulation du graphe pointé (I, A, i_0) dans le graphe pointé (J, B, j_0), on dit que les graphes pointés (I, A, i_0) et (J, B, j_0) sont *bissimilaires*.

Notons que la relation de bissimilarité — qui est une relation d'équivalence sur la collection des graphes pointés — est plus générale que la notion d'isomorphisme : tout isomorphisme de graphes pointés constitue manifestement une bissimulation, mais la réciproque est fausse, ainsi que l'illustre la figure 8.2.

Théorème 8.3.3 (Caractérisation de la relation d'égalité) — *Pour tous graphes pointés (I, A, i) et (J, B, j) on a l'équivalence*

$$\mathsf{res}(I, A, i) = \mathsf{res}(J, B, j) \qquad \Leftrightarrow \qquad (I, A, i) \text{ et } (J, B, j) \text{ bissimilaires.}$$

Cette caractérisation de l'égalité en termes de graphes pointés induit naturellement une caractérisation de la relation d'appartenance, que nous avons illustrée dans la figure 8.3 :

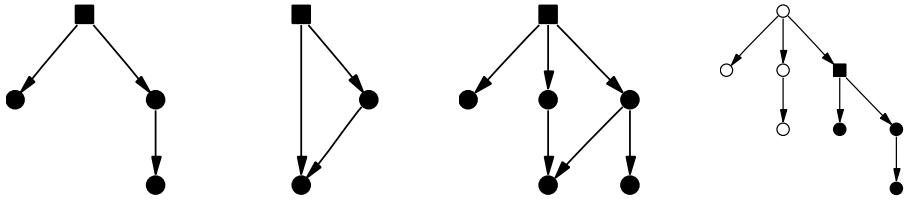
Corollaire 8.3.4 (Caractérisation de la relation d'appartenance) — *Soient (I, A, i) et (J, B, j) des graphes pointés quelconques. On a l'équivalence :*

$$\mathsf{res}(I, A, i) \in \mathsf{res}(J, B, j) \qquad \Leftrightarrow \qquad \exists\, j' \in J \quad B(j', j) \quad \text{et} \quad (I, A, i),\ (J, B, j') \text{ bissimilaires.}$$

Il est important de comprendre que ces deux résultats sont très généraux, puisque tout ensemble peut être obtenu par réification d'un graphe pointé particulier. En effet, si x est un ensemble quelconque, on considère le graphe pointé (I, A, i_0) défini par

[11] En théorie des ensembles, un graphe pointé est un triplet (I, A, i_0) où I est un ensemble arbitraire, $A \subset (I \times I)$ une relation binaire sur I, et $i_0 \in I$ un sommet distingué du graphe sous-jacent $G = (I, A)$.

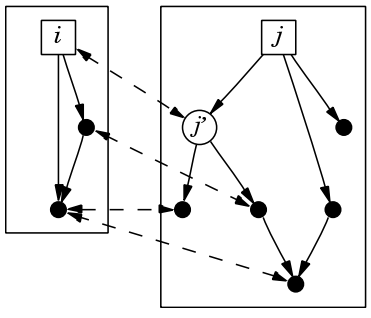
Ci dessous, on a représenté quatre graphes pointés bissimilaires mais deux-à-deux non-isomorphes. Dans chacun de ces quatre graphes, la racine est représentée par un carré et les arcs $A(j,i)$ sont représentés par des flèches allant du sommet i vers le sommet j. Comme ces graphes pointés sont bien fondés, il n'est pas nécessaire de recourir à l'axiome d'anti-fondation pour montrer qu'ils sont réifiés par l'entier de von Neumann 2.



On remarquera que le graphe pointé de droite comporte des sommets inaccessibles (représentés en blanc sur la figure), c'est-à-dire des sommets qu'il n'est pas possible d'atteindre à partir de la racine par une suite finie d'arcs consécutifs en suivant le sens des flèches. Dans le cas général, il est toujours possible d'effacer les sommets inaccessibles d'un graphe pointé sans pour autant changer son réifié.

Le graphe pointé de droite est en réalité obtenu à partir de la représentation arborescente de l'entier de von Neumann 3, dans laquelle on a décalé d'un cran vers le bas la racine de l'arbre, rendant ainsi inaccessibles tous les sommets au dessus de la nouvelle racine. Par ce mécanisme, on passe donc naturellement d'un graphe pointé représentant l'ensemble $3 = \{0;1;2\}$ à un graphe pointé représentant un de ses éléments, ici l'ensemble $2 = \{\varnothing;\{\varnothing\}\}$. Dans le cas général, ce mécanisme consistant à déplacer la racine un sommet plus bas permet d'interpréter la relation d'appartenance en termes de graphes pointés (voir Fig 8.3).

FIG. 8.2 – Quatre graphes pointés représentant l'entier de von Neumann 2



Dans la figure ci-contre, on a représenté deux graphes pointés (I,A,i) (graphe pointé de gauche) et (J,B,j) (graphe pointé de droite) dont les réifications sont respectivement les entiers de von Neumann 2 et 3.

Les flèches (bi-directionnelles) dessinées en tiretés représentent la relation de bissimulation entre les graphes pointés (I,A,i) et (J,B,j'), où j' est un sommet du second graphe tel que $(j',j) \in B$.

FIG. 8.3 – La relation $2 \in 3$ en termes de graphes pointés

1. $I \;=\; \mathrm{Cl}(\{x\}) \;=\; \{x\} \cup \mathrm{Cl}(x)$,[12]
2. $A \;=\; \{(j,i) \in I \times I; \quad j \in i\}$,
3. $i_0 \;=\; x$,

et on vérifie immédiatement que x est le réifié du graphe pointé (I, A, i_0), simplement en remarquant que la famille $(x_i)_{i\in I}$ définie par $x_i = i$ pour tout $i \in I$ est une réification du graphe $G = (I, A)$.

Par ailleurs, ces deux résultats suggèrent une traduction de la théorie des hyper-ensembles (avec son axiome d'anti-fondation) dans la théorie des ensembles standard (*i.e.* ZF sans axiome de la fondation ni axiome d'anti-fondation) simplement en interprétant :

- les hyper-ensembles par des graphes pointés construits dans ZF, en relativisant chacune des quantifications à la collection des graphes pointés ;
- la relation d'égalité par l'existence d'une bissimulation entre deux graphes pointés ;
- la relation d'appartenance par la combinaison d'un décalage de la racine (voir Fig. 8.3) et d'une bissimulation.

On vérifie alors que par ce procédé, chaque formule prouvable dans la théorie des hyper-ensembles se traduit en une formule prouvable dans la théorie des ensembles de Zermelo-Fränkel,[13] d'où il ressort que :

Théorème 8.3.5 (Équiconsistance) — *La théorie des hyper-ensembles et la théorie des ensembles de Zermelo-Fränkel sont équiconsistantes.*

Nous allons voir dans la suite de ce chapitre ainsi que dans le chapitre suivant que cette construction de modèle (définie à l'origine en théorie des ensembles) se transpose très naturellement en théorie des types.

8.4 La théorie des ensembles de Frege dans le système U

Dans cette section, nous nous proposons d'utiliser la représentation des ensembles par les graphes pointés pour construire une théorie des ensembles incohérente dans le système U. Le système formel que nous allons modéliser est la version intuitionniste de la théorie des ensembles de Frege [23], qui se distingue des théories des ensembles habituelles Z et ZF (supposées cohérentes) par la présence d'un schéma d'axiomes de compréhension non-restreint

$$\exists\, y \quad \forall\, x \qquad x \in y \quad \Leftrightarrow \quad \phi(x)$$

permettant de construire l'ensemble des objets x tels que $\phi(x)$ pour n'importe quelle formule $\phi(x)$ dépendant éventuellement de la variable x. Bien entendu, cette théorie des ensembles (dans sa version intuitionniste comme dans sa version classique) est incohérente, et il nous suffira de traduire le paradoxe de Russell pour obtenir une preuve originale de l'énoncé

$$(\bot) \qquad \forall a : \mathsf{Prop}\,.\, a$$

dans le système U. L'axiomatique complète de la théorie des ensembles intuitionniste de Frege (basée sur le calcul des prédicats en logique intuitionniste du premier ordre) est récapitulée par la figure 8.4.

[12] Où $\mathrm{Cl}(x)$ désigne la clôture transitive de l'ensemble x qui a été définie au paragraphe 3.4.3.

[13] Il suffit en fait de vérifier que la traduction de chacun des axiomes de ZF + Anti-Fondation est une formule prouvable dans ZF, puisque les connecteurs logiques (et leurs règles de déduction) sont traduits par eux-mêmes.

Axiomes d'égalité

(Identité)	$\forall\, x$	$x = x$
(Symétrie)	$\forall\, x, y$	$x = y \;\Rightarrow\; y = x$
(Transitivité)	$\forall\, x, y, z$	$x = y \;\Rightarrow\; y = z \;\Rightarrow\; x = z$
(Compatibilité à gauche)	$\forall\, x, y, z$	$x = y \;\Rightarrow\; y \in z \;\Rightarrow\; x \in z$
(Compatibilité à droite)	$\forall\, x, y, z$	$x \in y \;\Rightarrow\; y = z \;\Rightarrow\; x \in z$

Axiome d'extensionnalité

(Extensionnalité)	$\forall\, x, y$	$(\forall\, z \quad z \in x \;\Leftrightarrow\; z \in y) \;\Rightarrow\; x = y$

Schéma de compréhension non-restreint

(Compréhension)	$\forall\, x_1, \ldots, x_n \quad \exists y \quad \forall x$	$x \in y \;\Leftrightarrow\; \phi$

où ϕ est une formule telle que $FV(\phi) \subset \{x_1; \ldots; x_n; x\}$

FIG. 8.4 – Axiomatique de la théorie des ensembles intuitionniste de Frege

8.4.1 Les grandes lignes de la preuve

Techniquement, la démonstration que nous allons effectuer suit de très près la preuve du paradoxe de Girard telle qu'elle est présentée dans [17].

Nous allons dans un premier temps introduire deux relations binaires EQV et ELT sur les graphes pointés sous la forme de fonctions à six paramètres (puisque chaque graphe pointé en comporte trois) dont nous allons établir les principales propriétés.

Dans un second temps, nous utiliserons le polymorphisme fonctionnel du système U pour construire un type de données U permettant de représenter tous les graphes pointés, muni d'une injection i associant à chaque graphe pointé (X, A, a) un objet $i(X, A, a)$ de type U. Nous montrerons alors que les relations EQV et ELT définies sur la classe des graphes pointés sont représentées par deux relations binaires eqv et elt sur le type U.

En utilisant le fait que le type U se trouve au même niveau que les types de base X de chacun des graphes pointés (X, A, a) qu'il permet de représenter, nous montrerons que le graphe (U, elt) a une structure de graphe universel permetant de dériver le schéma d'axiomes de compréhension non-restreint. Nous serons alors en mesure de prouver l'incohérence du système U par une traduction immédiate du paradoxe de Russell.

Enfin, nous montrerons au paragraphe 8.4.5 comment il est possible de transformer cette preuve d'incohérence du système U en une preuve d'incohérence du système U^-, simplement en ne considérant dès le départ que des graphes pointés basés sur le type U représentant la collection des ensembles. Cette idée qui nous a été sugérée par Hugo Herbelin a une portée plus générale, puisqu'elle s'applique également au paradoxe de Girard (formulé à l'origine dans le système U) qu'elle permet de transporter dans le système U^- exactement de la même façon.

Il est important de préciser que toutes les démonstrations que nous allons présenter ont été complètement formalisées sur machine (dans le système U comme dans le système U^-). La

plupart du développement a été formalisé dans le système Coq (pour les parties de la preuve n'utilisant pas le polymorphisme fonctionnel du système U) et les termes de preuves correspondants ont ensuite été transférés dans un vérificateur indépendant basé sur les systèmes U et U^-, puis complétés par les termes de preuves spécifiques écrits à la main.

8.4.2 Graphes pointés dans le système U

Dans le système U, on appelle *graphe pointé* tout triplet (X, A, a) où X est un genre (*i.e.* $X : \mathsf{Type}$), A un terme de type $X{\to}X{\to}\mathsf{Prop}$ et a un terme de type X (dans un contexte donné). Notons que dans cette définition, X, A et a sont des objets du système U, tandis que le triplet (X, A, a) n'est qu'une notation commode pour regrouper ces trois objets, qui ne constitue pas un objet du formalisme.

Soient M un terme et ϕ une proposition, dépendant de la variable de type X et des variables de termes A et a de types $X{\to}X{\to}\mathsf{Prop}$ et X respectivement. On note $\lambda(X, A, a).\, M$, $\forall(X, A, a).\, \phi$ et $\exists(X, A, a).\, \phi$ les termes définis par :

$$\begin{aligned}
\lambda(X, A, a).\, M &\equiv \Lambda X\,.\,\lambda A : X{\to}X{\to}\mathsf{Prop}\,.\,\lambda a : X\,.\,M \\
\forall(X, A, a).\, \phi &\equiv \overline{\forall} X\,.\,\forall A : X{\to}X{\to}\mathsf{Prop}\,.\,\forall a : X\,.\,\phi \\
\exists(X, A, a).\, \phi &\equiv \overline{\exists} X\,.\,\exists A : X{\to}X{\to}\mathsf{Prop}\,.\,\exists a : X\,.\,\phi
\end{aligned}$$

Afin d'alléger l'écriture des termes, les applications multiples seront notées

$$M(N_1, \ldots, N_n) \quad \equiv \quad M\ N_1\ \cdots\ N_n$$

et cette notation sera utilisée indifféremment lorsque $N_1, \ldots, N_n$ désignent des arguments de termes et/ou des arguments de types. On remarquera que la construction $\lambda(X, A, a).\, M$ est définissable dans le système U^-, mais que les constructions $\forall(X, A, a).\, \phi$ et $\exists(X, A, a).\, \phi$ ne peuvent être définies que dans le système U, puisqu'elles utilisent toutes les deux la quantification $\overline{\forall}$ induite par la règle de formation $(\mathsf{Kind}, \mathsf{Prop})$.

La relation exprimant la bissimilarité de deux graphes pointés (X, A, a) et (Y, B, b) peut être représentée dans le système U par la fonction suivante, qui prend six arguments :

$$\begin{aligned}
&\mathsf{EQV} \quad : \quad \Pi X\,.\,(X{\to}X{\to}\mathsf{Prop}) \to X \to \Pi Y\,.\,(Y{\to}Y{\to}\mathsf{Prop}) \to Y \to \mathsf{Prop} \quad := \\
&\quad \lambda(X, A, a)\,.\,\lambda(Y, B, b)\,.\quad \exists R : (X{\to}Y{\to}\mathsf{Prop})\,. \\
&\qquad \big(\forall x, x' : X\,.\,\forall y : Y\,.\quad A(x', x) \ \Rightarrow\ R(x, y) \ \Rightarrow\ \exists y' : Y\,.\,R(x', y') \ \wedge\ B(y', y)\big) \quad \wedge \\
&\qquad \big(\forall y, y' : Y\,.\,\forall x : X\,.\quad B(y', y) \ \Rightarrow\ R(x, y) \ \Rightarrow\ \exists x' : X\,.\,R(x', y') \ \wedge\ A(x', x)\big) \quad \wedge \\
&\qquad R(a, b)
\end{aligned}$$

Dans la suite de ce paragraphe, on notera

$$(X, A, a) \approx (Y, B, b) \quad \equiv \quad \mathsf{EQV}(X, A, a, Y, B, b).$$

Proposition 8.4.1 (Équivalence) — *La relation de bissimilarité entre les graphes pointés est une relation d'équivalence :*

$$\forall (X,A,a)\,. \qquad (X,A,a) \approx (X,A,a)$$

$$\forall (X,A,a),(Y,B,b)\,. \quad (X,A,a) \approx (Y,B,b) \;\Rightarrow\; (Y,B,b) \approx (X,A,a)$$

$$\forall (X,A,a),(Y,B,b),(Z,C,c)\,. \quad (X,A,a) \approx (Y,B,b) \;\Rightarrow\; (Y,B,b) \approx (Z,C,c) \;\Rightarrow\; (X,A,a) \approx (Z,C,c)$$

Preuve. (Réflexivité) La relation d'égalité sur X définie par $R \equiv \lambda x,y : X\,.\,x =_X y$ est une bissimulation de (X,A,a) dans lui-même.

(Symétrie) Si R est une bissimulation de (X,A,a) dans (Y,B,b), alors la relation symétrique $R^{-1} \equiv \lambda y : Y\,.\,\lambda x : X\,.\,R(x,y)$ est une bissimulation de (Y,B,b) dans (X,A,a).

(Transitivité) Si R est une bissimulation de (X,A,a) dans (Y,B,b) et S une bissimulation de (Y,B,b) dans (Z,C,c), alors la relation composée $S \circ R \equiv \lambda x : X\,.\,\lambda z : Z\,.\,\exists y : Y\,.\,R(x,y) \wedge S(y,z)$ est une bissimulation de (X,A,a) dans (Z,C,c). □

Pour montrer les lemmes de compatibilité de la relation d'appartenance vis-à-vis de la relation de bissimilarité, nous aurons besoin du lemme suivant :

Lemme 8.4.2 — *Si (X,A,a) et (Y,B,b) sont des graphes pointés bissimilaires, et si $a' : X$ est un sommet du premier graphe tel que $A(a',a)$, alors il existe un sommet $b' : Y$ du second graphe tel que $B(b',b)$ et tel que les graphes pointés (X,A,a') et (Y,B,b') sont bissimilaires :*

$$\begin{array}{l}\forall (X,A,a),(Y,B,b).\quad (X,A,a) \approx (Y,B,b) \quad\Rightarrow\\ \quad \forall a' : X\,.\quad A(a',a) \quad\Rightarrow\quad \exists b' : Y\,.\quad B(b',b) \quad\wedge\quad (X,A,a') \approx (Y,B,b').\end{array}$$

Preuve. Soit $R : X \to Y \to \mathsf{Prop}$ une bissimulation de (X,A,a) dans (Y,B,b). Si $a' : X$ est un sommet tel que $A(a',a)$, il existe un sommet $b' : Y$ tel que $R(a',b')$ et $B(b',b)$ (puisque R est une bissimulation de (X,A,a) dans (Y,B,b) et $R(a,b)$). Il est alors immédiat que R est également une bissimulation de (X,A,a') dans (Y,B,b'). □

La relation d'appartenance entre graphes pointés est exprimée dans le système U par le terme suivant :

$$\begin{array}{l}\mathsf{ELT} \quad : \quad \Pi X\,.\,(X \to X \to \mathsf{Prop}) \to X \to \Pi Y\,.\,(Y \to Y \to \mathsf{Prop}) \to Y \to \mathsf{Prop} \quad :=\\ \quad \lambda (X,A,a)\,.\,\lambda (Y,B,b)\,.\quad \exists b' : Y\,.\quad B(b',b) \quad\wedge\quad (X,A,a) \approx (Y,B,b').\end{array}$$

Dans ce qui suit, la relation $\mathsf{ELT}(X,A,a,Y,B,b)$ sera notée $(X,A,a) \in (Y,B,b)$.

Lemme 8.4.3 (Compatibilité) — *La relation d'appartenance sur les graphes pointés est compatible à gauche et à droite par rapport à la relation de bissimilarité :*

$$\begin{array}{l}\forall (X,A,a),(Y,B,b),(Z,C,c).\\ \quad (X,A,a) \approx (Y,B,b) \quad\Rightarrow\quad (Y,B,b) \in (Z,C,c) \quad\Rightarrow\quad (X,A,a) \in (Z,C,c)\\ \forall (X,A,a),(Y,B,b),(Z,C,c).\\ \quad (X,A,a) \in (Y,B,b) \quad\Rightarrow\quad (Y,B,b) \approx (Z,C,c) \quad\Rightarrow\quad (X,A,a) \in (Z,C,c)\end{array}$$

Preuve. La compatibilité à gauche résulte de la transitivité de la relation de bissimilarité, et la compatibilité à droite est une conséquence immédiate du lemme 8.4.2. □

Nous terminerons ce paragraphe en énonçant la traduction de l'axiome d'extensionnalité en termes de graphes pointés. On remarquera que cette propriété n'est pas utilisée par la preuve du paradoxe de Russell que nous présenterons au paragraphe 8.4.4.

Lemme 8.4.4 (Extensionnalité) — *Si deux graphes pointés ont les mêmes éléments au sens de la relation* ELT*, alors ces graphes pointés sont bissimilaires :*

$$\begin{array}{l}\forall(X,A,a).\ \forall(Y,B,b)\\ \quad\big(\forall(Z,C,c).\qquad (Z,C,c)\in(X,A,a)\quad\Leftrightarrow\quad (Z,C,c)\in(Y,B,b)\big)\\ \qquad\Rightarrow\qquad (X,A,a)\approx(Y,B,b).\end{array}$$

Preuve. Soient (X,A,a) et (Y,B,b) deux graphes pointés tels que $(Z,C,c)\in(X,A,a)$ si et seulement si $(Z,C,c)\in(Y,B,b)$ pour tout graphe pointé (Z,C,c). On vérifie que la relation $R : X\to Y\to\mathsf{Prop}$ définie par

$$R\quad\equiv\quad\lambda x:X\,.\,\lambda y:Y\,.\quad(X,A,x)\approx(Y,B,y)\quad\vee\quad(x=_X a\ \wedge\ y=_Y b)$$

est une bissimulation de (X,A,a) dans (Y,B,b). □

8.4.3 Le type universel des graphes pointés

Nous allons maintenant définir le type U des graphes pointés, en utilisant un codage imprédicatif (au sens faible) de la somme dépendante $\Sigma X:\mathsf{Type}\,.\,(X\to X\to\mathsf{Prop})\times X$:

$$\mathsf{U}\quad:\quad\mathsf{Type}\quad:=\quad\big(\Pi X\,.\,(X\to X\to\mathsf{Prop})\to X\to\mathsf{Prop}\big)\to\mathsf{Prop}.$$

La fonction i qui envoie chaque graphe pointé (X,A,a) dans le type U est définie par

$$\begin{array}{rl}i\quad:\quad & \Pi X\,.\,(X\to X\to\mathsf{Prop})\to X\to\mathsf{U}\quad:=\\ & \lambda(X,A,a).\quad\lambda f:(\Pi X\,.\,(X\to X\to\mathsf{Prop})\to X\to\mathsf{Prop})\,.\quad f(X,A,a)\end{array}$$

Ici, la règle $(\mathsf{Kind},\mathsf{Type})$ qui autorise la formation du produit dépendant imprédicatif fonctionnel joue un rôle crucial, car c'est elle qui permet de construire le type U au même niveau que les types de base des graphes pointés (X,A,a). Dans le Calcul des Constructions avec univers par exemple, il serait possible de construire un type U analogue — en remplaçant la sorte Type par la sorte Type_1 dans la définition ci-dessus — mais le type U serait alors formé non pas dans la sorte Type_1, mais dans la sorte Type_2.

Dans le système U (comme dans le système U^-), la règle $(\mathsf{Kind},\mathsf{Type})$ permet de confondre le niveau des types de base des graphes pointés avec le niveau du type universel U censé les contenir tous, et c'est précisément cette particularité qui va nous permettre de dériver le schéma de compréhension non-restreint en construisant des graphes pointés basés sur le type U lui-même.

Lemme 8.4.5 (Injectivité) — *La fonction i est injective, en ce sens que si deux graphes pointés (X,A,a) et (Y,B,b) ont la même image par i dans le type U (au sens de l'égalité de Leibniz définie au paragraphe 8.1.4), alors les graphes pointés (X,A,a) et (Y,B,b) sont bissimilaires :*

$$\forall(X,A,a),(Y,B,b).\quad i(X,A,a)=_{\mathsf{U}} i(Y,B,b)\quad\Rightarrow\quad(X,A,a)\approx(Y,B,b).$$

Preuve. Considérons deux graphes pointés (X, A, a) et (Y, B, b), une preuve d'égalité

$$\xi \quad : \quad i(X, A, a) =_{\mathsf{U}} i(Y, B, b)$$

ainsi que le prédicat $P : \mathsf{U} \to \mathsf{Prop}$ défini par

$$P \quad := \quad \lambda u : \mathsf{U} \,.\quad u \left(\lambda(Z, C, c).\ (X, A, a) \approx (Z, C, c)\right).$$

On remarque que les propositions $P(i(X, A, a))$ et $(X, A, a) \approx (X, A, a)$ sont convertibles, de même que les propositions $P(i(Y, B, b))$ et $(X, A, a) \approx (Y, B, b)$. Comme la relation de bissimilarité est réflexive, la proposition $(X, A, a) \approx (X, A, a)$ est prouvable par un terme de preuve p, et on vérifie alors de manière immédiate que

$$\xi\ P\ p \quad : \quad (X, A, a) \approx (Y, B, b). \qquad \square$$

Il est important de noter que la fonction i n'est pas surjective, et nous construirons un peu plus loin un objet de type U qui n'est l'image d'aucun graphe pointé par la fonction i. Pour cette raison, nous serons fréquemment amenés à relativiser les quantifications existentielle et universelle sur le type U aux objets satisfaisant le prédicat $\mathsf{set} : \mathsf{U} \to \mathsf{Prop}$ défini par

$$\mathsf{set} \quad : \quad \mathsf{U} \to \mathsf{Prop} \quad := \quad \lambda u : \mathsf{U} \,.\quad \exists(X, A, a).\quad u =_{\mathsf{U}} i(X, A, a),$$

et qui exprime que l'objet $u : \mathsf{U}$ auquel ce prédicat est appliqué est l'image d'un graphe pointé. Dans ce qui suit, on appellera *ensemble* tout objet $u : \mathsf{U}$ tel que $\mathsf{set}(u)$.

Un exemple d'objet de type U en dehors de la classe définie par le prédicat set est le terme $\mathsf{out} : \mathsf{U}$ défini par :

$$\mathsf{out} \quad : \quad \mathsf{U} \quad := \quad \lambda f : (\Pi X \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}) \,.\, \bot \,.$$

Lemme 8.4.6 — *Le terme* $\mathsf{out} : \mathsf{U}$ *n'est pas un ensemble :* $\neg\mathsf{set}(\mathsf{out})$.

Preuve. Pour démontrer la proposition $\neg\mathsf{set}(\mathsf{out})$, il suffit de prouver que pour tout graphe pointé (X, A, a), on a l'implication $i(X, A, a) =_{\mathsf{U}} \mathsf{out} \Rightarrow \bot$. On considère le prédicat P défini par

$$P \quad : \quad \mathsf{U} \to \mathsf{Prop} \quad := \quad \lambda u : \mathsf{U} \,.\quad u \ (\lambda(X, A, a).\top) \,,$$

et on remarque que pour tout graphe pointé (X, A, a) les termes $P(i(X, A, a))$ et $\top$ sont convertibles, de même que les termes $P(\mathsf{out})$ et $\bot$. Soit id une preuve de la proposition $\top$ (par exemple le terme $\lambda a : \mathsf{Prop} \,.\, \lambda \xi^a . \xi$). On vérifie aisément que

$$\lambda \xi^{i(X,A,a) =_{\mathsf{U}} \mathsf{out}} . \xi\ P\ \mathsf{id} \quad : \quad i(X, A, a) =_{\mathsf{U}} \mathsf{out} \quad \Rightarrow \quad \bot \,. \qquad \square$$

Comme tous les graphes pointés s'injectent dans le type U par la fonction i, il est possible d'exprimer les relations de bissimilarité et d'appartenance sur les objets de type U par les

termes suivants

$$\begin{array}{l}\mathsf{eqv} \quad : \quad \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda u, v : \mathsf{U}\,. \quad \exists (X, A, a), (Y, B, b). \\ \qquad u =_{\mathsf{U}} i(X, A, a) \quad \wedge \quad v =_{\mathsf{U}} i(Y, B, b) \quad \wedge \quad (X, A, a) \approx (Y, B, b)\end{array}$$

$$\begin{array}{l}\mathsf{elt} \quad : \quad \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda u, v : \mathsf{U}\,. \quad \exists (X, A, a), (Y, B, b). \\ \qquad u =_{\mathsf{U}} i(X, A, a) \quad \wedge \quad v =_{\mathsf{U}} i(Y, B, b) \quad \wedge \quad (X, A, a) \in (Y, B, b).\end{array}$$

qui définissent deux relations binaires $\mathsf{eqv}(u, v)$ et $\mathsf{elt}(u, v)$ sur U, qu'on notera par la suite $u \approx v$ et $u \in v$ respectivement.

Lemme 8.4.7 (Domaine des relations eqv et elt) — *Les relations binaires $u \approx v$ et $u \in v$ sont définies sur la classe des ensembles, c'est-à-dire :*

$$\begin{array}{llll}\forall u, v : \mathsf{U}\,. & u \approx v \;\Rightarrow\; \mathsf{set}(u) & \forall u, v : \mathsf{U}\,. & u \in v \;\Rightarrow\; \mathsf{set}(u) \\ \forall u, v : \mathsf{U}\,. & u \approx v \;\Rightarrow\; \mathsf{set}(v) & \forall u, v : \mathsf{U}\,. & u \in v \;\Rightarrow\; \mathsf{set}(v)\end{array}$$

Preuve. Résulte immédiatement de la définition des relations $u \approx v$ et $u \in v$. □

Contrairement à la relation EQV (définie sur la classe des graphes pointés), la relation $u \approx v$ n'est pas une relation d'équivalence, mais une relation d'équivalence partielle dont le domaine est la classe des ensembles :

Lemme 8.4.8 (Relation d'équivalence partielle) — *La relation binaire $u \approx v$ est une relation d'équivalence partielle définie sur la classe des objets $u : \mathsf{U}$ tels que $\mathsf{set}(u)$:*

$$\begin{array}{lll}\forall u : \mathsf{U}\,. & u \approx u \;\Leftrightarrow\; \mathsf{set}(u) \\ \forall u, v : \mathsf{U}\,. & u \approx v \;\Rightarrow\; v \approx u \\ \forall u, v, w : \mathsf{U}\,. & u \approx v \;\Rightarrow\; v \approx w \;\Rightarrow\; u \approx w\end{array}$$

Preuve. Ces trois propositions résultent de la réflexivité, de la symétrie et de la transitivité de la relation $(X, A, a) \approx (Y, B, b)$ qui ont été établies par le lemme 8.4.1. La preuve de transitivité utilise l'injectivité de la fonction i (lemme 8.4.5). □

On remarquera que pour exprimer la symétrie et la transitivité de la relation $u \approx v$, il n'est pas nécessaire de relativiser les quantifications sur u, v et w par le prédicat set, puisque les propositions $\mathsf{set}(u)$, $\mathsf{set}(v)$ et $\mathsf{set}(w)$ sont des conséquences immédiates des hypothèses (d'après le lemme 8.4.7). Par ailleurs, la relation elt est compatible vis-à-vis de la relation eqv :

Lemme 8.4.9 (Compatibilité) — *La relation d'appartenance $u \in v$ est compatible à gauche et à droite vis-à-vis de la relation d'équivalence partielle $u \approx v$:*

$$\begin{array}{lll}\forall u, v, w : \mathsf{U}\,. & u \approx v \;\Rightarrow\; v \in w \;\Rightarrow\; u \in w \\ \forall u, v, w : \mathsf{U}\,. & u \in v \;\Rightarrow\; v \approx w \;\Rightarrow\; u \in w\end{array}$$

Preuve. Ces deux propositions résultent immédiatement des lemmes 8.4.3 (compatibilité de la relation ELT) et 8.4.5 (injectivité de la fonction i). □

Dans le type U, la traduction de l'axiome d'extensionnalité est également prouvable. (On remarquera la nécessité de relativiser les quantifications sur u et v à l'aide du prédicat set.)

Lemme 8.4.10 (Extensionnalité) — *Si deux ensembles u et v ont les mêmes éléments (au sens de la relation elt), alors $u \approx v$:*

$$\begin{array}{l}\forall u : \mathsf{U} \,.\, \mathsf{set}(u) \Rightarrow \forall v : \mathsf{U} \,.\, \mathsf{set}(v) \Rightarrow \\ \quad \big(\forall w : \mathsf{U} \,.\quad w \in u \;\Leftrightarrow\; w \in v\big) \quad\Rightarrow\quad u \approx v.\end{array}$$

Preuve. Conséquence immédiate du lemme 8.4.4. □

Dans les lignes qui précèdent, nous avons transporté dans le type U les relations EQV et ELT définies à l'origine sur la classe des graphes pointés. Comme le type U se situe au même niveau que les types de base des graphes pointés, nous pouvons à présent utiliser ce type de données particulier pour former de nouveaux graphes pointés.

Le lemme suivant exprime que le graphe $(\mathsf{U}, \mathsf{elt})$ est un graphe universel vis-à-vis de la relation de bissimilarité, puisque tout graphe pointé (X, A, a) peut être représenté par un graphe pointé basé sur le graphe universel $(\mathsf{U}, \mathsf{elt})$, et dont la racine est précisément l'image $i(X, A, a)$ du graphe pointé (X, A, a) par l'injection i :

Lemme 8.4.11 (Plongement) — *Tout graphe pointé (X, A, a) est bissimilaire au graphe pointé $\big(\mathsf{U}, \mathsf{elt}, i(X, A, a)\big)$:*

$$\forall (X, A, a). \qquad (X, A, a) \approx \big(\mathsf{U}, \mathsf{elt}, i(X, A, a)\big).$$

Preuve. Il suffit de vérifier que la relation $R : X \to \mathsf{U} \to \mathsf{Prop}$ définie par

$$R \qquad \equiv \qquad \lambda x : X \,.\, \lambda u : \mathsf{U} \,.\quad i(X, A, x) \approx u$$

est une bissimulation du graphe pointé (X, A, a) dans $\big(\mathsf{U}, \mathsf{elt}, i(X, A, a)\big)$. □

8.4.4 Le schéma de compréhension non-restreint

Nous sommes à présent en mesure d'associer à tout prédicat $P : \mathsf{U} \to \mathsf{Prop}$ un ensemble noté $\mathsf{fold}(P)$ dont les éléments sont caractérisés par

$$\forall u : \mathsf{U} \,. \qquad u \in \mathsf{fold}(P) \quad\Leftrightarrow\quad P(u).$$

En toute rigueur, la fonction $\mathsf{fold} : (\mathsf{U} \to \mathsf{Prop}) \to \mathsf{U}$ que nous allons définir ne pourra pas satisfaire l'équivalence ci-dessus pour n'importe quel prédicat P, mais seulement pour les prédicats *compatibles* vis-à-vis de la relation d'équivalence $u \approx v$,[14] c'est-à-dire les termes $P : \mathsf{U} \to \mathsf{Prop}$ tels que les propositions $P(u)$ et $u \approx v$ entraînent $P(v)$ quels que soient les objets u et v de type U. Cette condition de compatibilité est testée dans le système U par le terme compat de type $(\mathsf{U} \to \mathsf{Prop}) \to \mathsf{Prop}$ défini par

$$\mathsf{compat} \quad := \quad \lambda P : (\mathsf{U} \to \mathsf{Prop}) \,.\quad \forall u, v : \mathsf{U} \,.\quad P(u) \;\Rightarrow\; u \approx v \;\Rightarrow\; P(v),$$

et il est facile de vérifier que toute proposition $\phi : \mathsf{Prop}$ construite uniquement à partir

[14] En effet, le membre gauche $u \in \mathsf{fold}(P)$ de l'équivalence est lui-même un prédicat compatible (par rapport à la variable $u : \mathsf{U}$), indépendamment de la définition du terme $\mathsf{fold}(P)$.

– des relations binaires $u \approx v$ et $u \in v$;
– des connecteurs logiques $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$;
– des quantifications relativisées $\forall u : \mathsf{U} \,.\, \mathsf{set}(\mathsf{U}) \Rightarrow \psi$ et $\exists u : \mathsf{U} \,.\, \mathsf{set}(u) \wedge \psi$
satisfait l'hypothèse de compatibilité $\mathsf{compat}(\lambda u : \mathsf{U} \,.\, \phi)$ lorsqu'elle est abstraite par rapport à n'importe laquelle de ses variables libres.

La construction de l'ensemble $\mathsf{fold}(P)$ s'effectue à l'aide d'une fonction FOLD définie par

$$\begin{array}{l} \mathsf{FOLD} \quad : \quad (\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \qquad \lambda P : (\mathsf{U}{\to}\mathsf{Prop}) \,.\quad \lambda u, v : \mathsf{U} \,.\quad u \in v \quad \vee \quad (\mathsf{set}(u) \ \wedge \ P(u) \ \wedge \ v =_{\mathsf{U}} \mathsf{out}) \,, \end{array}$$

et qui à chaque prédicat $P : \mathsf{U} \to \mathsf{Prop}$ associe une relation binaire $\mathsf{FOLD}(P) : \mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop}$ caractérisée par

$$\mathsf{FOLD}(P)(u, v) \qquad \Leftrightarrow \qquad u \in v \quad \vee \quad (\mathsf{set}(u) \ \wedge \ P(u) \ \wedge \ v =_{\mathsf{U}} \mathsf{out})$$

pour tous $u, v : \mathsf{U}$.

Intuitivement, la relation $\mathsf{FOLD}(P)$ définit un graphe $(\mathsf{U}, \mathsf{FOLD}(P))$ qui étend le graphe universel $(\mathsf{U}, \mathsf{elt})$ en lui ajoutant un arc reliant u à l'objet out (qui n'est pas un ensemble) pour chaque ensemble u tel que $P(u)$. Comme out est connecté (par la relation $\mathsf{FOLD}(P)$) à tous les ensembles u tels que $P(u)$, et à eux seulement, le graphe pointé $(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out})$ représente l'ensemble de tous les ensembles u tels que $P(u)$, ainsi que nous allons le démontrer.

Lemme 8.4.12 (Propriétés de $\mathsf{FOLD}(P)$) — *Pour tout prédicat $P : \mathsf{U} \to \mathsf{Prop}$, la relation $\mathsf{FOLD}(P)$ satisfait les équivalences suivantes :*

$$\begin{array}{lll} \forall u, v : \mathsf{U} \,. & \mathsf{FOLD}(P)(u, v) \ \wedge \ \mathsf{set}(v) & \Leftrightarrow \quad u \in v \\ \forall u : \mathsf{U} \,. & \mathsf{FOLD}(P)(u, \mathsf{out}) & \Leftrightarrow \quad \mathsf{set}(u) \ \wedge \ P(u) \end{array}$$

Preuve. Résulte immédiatement de la définition de la relation $\mathsf{FOLD}(P)$ et du fait que out n'est pas un ensemble (*i.e.* $\neg\mathsf{set}(\mathsf{out})$). □

Lemme 8.4.13 — *Pour tout prédicat $P : \mathsf{U} \to \mathsf{Prop}$ et pour tout objet $u : \mathsf{U}$ tel que $\mathsf{set}(u)$, les graphes pointés $(\mathsf{U}, \mathsf{elt}, u)$ et $\big(\mathsf{U}, \mathsf{FOLD}(P), u\big)$ sont bissimilaires.*

Preuve. Soient $P : \mathsf{U} \to \mathsf{Prop}$ et $u : \mathsf{U}$ tels que $\mathsf{set}(u)$. En utilisant le lemme précédent, on montre aisément que la relation R définie par

$$R \quad : \quad \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \quad \lambda v, w : \mathsf{U} \,.\quad \mathsf{set}(v) \ \wedge \ v =_{\mathsf{U}} w$$

(c'est-à-dire la relation d'égalité restreinte aux ensembles) est une bissimulation du graphe pointé $(\mathsf{U}, \mathsf{elt}, u)$ dans le graphe pointé $\big(\mathsf{U}, \mathsf{FOLD}(P), u\big)$. □

Posons à présent

$$\mathsf{fold} \quad : \quad (\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \quad := \quad \lambda P : (\mathsf{U}{\to}\mathsf{Prop}) \,.\ \ i(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}).$$

Il s'agit à présent de montrer que pour tout prédicat compatible $P : \mathsf{U} \to \mathsf{Prop}$, l'ensemble $\mathsf{fold}(P)$ satisfait les propriétés requises, c'est-à-dire :

1. si $P(u)$, alors $u \in \mathsf{fold}(P)$ (Introduction) ;
2. si $u \in \mathsf{fold}(P)$, alors $P(u)$ (Élimination)

pour tout objet $u : \mathsf{U}$ tel que $\mathsf{set}(u)$. On remarquera que l'hypothèse de compatibilité du prédicat P n'est utile que pour établir la propriété d'élimination, mais qu'elle n'est pas nécessaire pour démontrer la propriété d'introduction.

Lemme 8.4.14 (Introduction) — *Pour tout prédicat P et pour tout ensemble u tel que $P(u)$, on a $u \in \mathsf{fold}(P)$:*

$$\forall P : \mathsf{U} \to \mathsf{Prop}\,.\quad \forall u : \mathsf{U}\,.\quad \mathsf{set}(u) \quad\Rightarrow\quad P(u) \quad\Rightarrow\quad u \in \mathsf{fold}(P).$$

Preuve. Soient $P : \mathsf{U} \to \mathsf{Prop}$ et $u : \mathsf{U}$ tels que $\mathsf{set}(u)$ et $P(u)$. Comme u est un ensemble, il existe un graphe pointé (X, A, a) tel que $u =_{\mathsf{U}} i(X, A, a)$. Cette égalité permet d'établir les équivalences de bissimulation

$$(X, A, a) \quad\approx\quad (\mathsf{U}, \mathsf{elt}, u) \quad\approx\quad \big(\mathsf{U}, \mathsf{FOLD}(P), u\big),$$

qui résultent respectivement des lemmes 8.4.11 (puisque $u =_{\mathsf{U}} i(X, A, a)$) et 8.4.13 (puisque u est un ensemble). Comme u satisfait P par hypothèse, on a $\mathsf{FOLD}(P)(u, \mathsf{out})$, d'où il ressort que le graphe pointé (X, A, a) appartient au graphe pointé $\big(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}\big)$ au sens de la relation binaire ELT définie sur la collection des graphes pointés. En injectant (X, A, a) et $\big(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}\big)$ dans le type U par la fonction i, nous obtenons alors

$$u \quad =_{\mathsf{U}} \quad i(X, A, a) \quad\in\quad i\big(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}\big) \quad\equiv\quad \mathsf{fold}(P)\,. \qquad \square$$

Lemme 8.4.15 (Élimination) — *Pour tout prédicat compatible P et pour tout ensemble u tel que $u \in \mathsf{fold}(P)$, on a $P(u)$:*

$$\forall P : \mathsf{U} \to \mathsf{Prop}\,.\quad \mathsf{compat}(P) \quad\Rightarrow\quad \forall u : \mathsf{U}\,.\quad u \in \mathsf{fold}(P) \quad\Rightarrow\quad P(u).$$

Preuve. Soient $P : \mathsf{U} \to \mathsf{Prop}$ un prédicat compatible, et $u : \mathsf{U}$ tel que $u \in \mathsf{fold}(P)$. D'après les définitions des termes elt et $\mathsf{FOLD}(P)$, il existe des graphes pointé (X, A, a) et (Z, C, c) tels que

$$u =_{\mathsf{U}} i(X, A, a), \qquad \mathsf{fold}(P) =_{\mathsf{U}} i(Z, C, c) \qquad \text{et} \qquad (X, A, a) \in (Z, C, c).$$

Comme par ailleurs

$$i\big(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}\big) \quad\equiv\quad \mathsf{fold}(P) \quad=_{\mathsf{U}} \quad i(Z, C, c),$$

les graphes pointés $\big(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}\big)$ et (Z, C, c) sont bissimilaires (lemme 8.4.5), d'où il ressort que $(X, A, a) \in \big(\mathsf{U}, \mathsf{FOLD}(P), \mathsf{out}\big)$ d'après le lemme 8.4.3. Par définition du terme ELT, il existe un objet $v : \mathsf{U}$ tel que

$$(X, A, a) \approx \big(\mathsf{U}, \mathsf{FOLD}(P), v\big) \qquad \text{et} \qquad \mathsf{FOLD}(P)(v, \mathsf{out}).$$

D'après la seconde de ces deux relations (lemme 8.4.12), v est un ensemble tel que $P(v)$, et il existe un graphe pointé (Y, B, b) tel que $v =_{\mathsf{U}} i(Y, B, b)$. D'après les lemmes 8.4.13 et 8.4.11 on a les équivalences de bissimulation

$$(X, A, a) \quad\approx\quad (\mathsf{U}, \mathsf{FOLD}(P), v) \quad\approx\quad (\mathsf{U}, \mathsf{elt}, v) \quad\approx\quad (Y, B, b),$$

lesquelles entraînent que $u \approx v$, puisque $u =_{\mathsf{U}} i(X, A, a)$ et $v =_{\mathsf{U}} i(Y, B, b)$. Nous avons donc montré l'existence d'un objet $v : \mathsf{U}$ tel que $P(v)$ et $u \approx v$, ce qui entraîne aussitôt que $P(u)$ puisque P est un prédicat compatible. □

Les propriétés de la fonction fold étant établies, il est maintenant facile de dériver le paradoxe de Russell. Pour cela, on considère le prédicat $\lambda u : \mathsf{U} \,.\, \neg u \in u$, dont on montre qu'il est compatible en utilisant le lemme 8.4.9. On pose alors

$$R \quad : \quad \mathsf{U} \quad := \quad \mathsf{fold}\big(\lambda u : \mathsf{U} \,.\, \neg u \in u\big)$$

En utilisant les lemmes 8.4.14 et 8.4.15, on construit deux termes de preuve

$$p \quad : \quad \neg R \in R \quad \Rightarrow \quad R \in R$$

$$q \quad : \quad R \in R \quad \Rightarrow \quad \neg R \in R,$$

lesquels permettent de dériver le paradoxe :

Proposition 8.4.16 (Incohérence) — *Dans le système U, toutes les propositions sont prouvables :*

$(\bot)$ $$\forall a : \mathsf{Prop} \,.\, a.$$

Terme de preuve. $(\lambda \xi^{R \in R} \,.\, q\ \xi\ \xi)\ (p\ (\lambda \xi^{R \in R} \,.\, q\ \xi\ \xi)) \quad : \quad \bot.$ □

On notera que le terme de preuve ci-dessus a essentiellement la structure du λ-terme bouclant $\Delta\Delta \equiv (\lambda x \,.\, xx)(\lambda x \,.\, xx)$ à ceci près qu'ici, on a dû insérer deux "coercitions" p et q qui sont en fait les preuves des parties directe et réciproque du schéma de compréhension spécialisé sur la formule $\phi(x) \equiv x \notin x$ et sur l'ensemble $x \equiv R$. D'après le lemme 8.1.8, le terme de preuve de la proposition précédente n'a pas de forme normale de tête et, en pratique, le calcul de ses β-réduits successifs conduit à former des termes de plus en plus gros.[15]

8.4.5 Une reformulation du paradoxe dans le système U^-

Un examen attentif du paradoxe que nous venons de présenter montre que toute la preuve d'incohérence se joue sur des graphes pointés basés sur le type U, en ce sens que les seuls graphes pointés (X, A, a) dont le type de base X est utilisé de manière explicite dans la preuve sont ceux pour lesquels on a $X \equiv \mathsf{U}$.[16]

À partir de cette constation, il est possible de réécrire toute la démonstration en ne considérant dès le départ que des graphes pointés basés sur le type U et représentés par des couples (A, a) où $A : \mathsf{U} \to \mathsf{U} \to \mathsf{Prop}$ est une relation binaire sur U et a un objet quelconque de type U. L'avantage de cette méthode est qu'elle supprime la nécessité de recourir à la quantification universelle $\overline{\forall}\alpha \,.\, \phi$ induite par la règle de formation $(\mathsf{Kind}, \mathsf{Prop})$, et permet ainsi de reformuler le paradoxe dans le système U^-. Notons que cette reformulation du paradoxe est rendue possible par le fait que le type

$$\mathsf{U} \quad \equiv \quad (\Pi X \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}) \to \mathsf{Prop}$$

que nous avons défini au paragraphe 8.4.3 est un terme bien formé dans le système U^-.

[15] À l'heure actuelle, nous ne savons pas s'il est possible de modifier la formulation du paradoxe de manière à obtenir un terme de preuve susceptible de se β-réduire sur lui-même.

[16] Il s'agit typiquement des graphes pointés de la forme $(\mathsf{U}, \mathsf{elt}, u)$ ou $(\mathsf{U}, \mathsf{FOLD}(P), u)$.

Les fonctions monomorphes EQV', ELT' et i' Dans ce qui suit, nous ne considérerons que des graphes pointés (A, a) basés sur le type U. On utilisera les abbréviations suivantes

$$\begin{array}{lcl} \lambda(A,a)\,.\,M & \equiv & \lambda A:(\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop})\,.\,\lambda a:\mathsf{U}\,.\,M \\ \forall(A,a)\,.\,\phi & \equiv & \forall A:(\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop})\,.\,\forall a:\mathsf{U}\,.\,\phi \\ \exists(A,a)\,.\,\phi & \equiv & \exists A:(\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop})\,.\,\exists a:\mathsf{U}\,.\,\phi, \end{array}$$

qui sont bien définies dans le système U^- puisqu'elles n'utilisent pas la quantification de type $\overline{\forall}\alpha\,.\,\phi$ spécifique au système U.

Dans ce cadre, la relation de bissimilarité $(A, a) \approx (B, b)$, la relation d'appartenance $(A, a) \in (B, b)$ et la fonction d'injection envoyant chaque graphe pointé (A, a) dans le type U sont exprimées par des termes ELT', ELT' et i' définis dans le système U^- par :

$$\begin{array}{l} \mathsf{EQV}' \quad : \quad (\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \to (\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda(A,a)\,.\,\lambda(B,b)\,.\quad \exists R:(\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop})\,. \\ \qquad \big(\forall x,x',y:\mathsf{U}\,.\quad A(x',x)\ \Rightarrow\ R(x,y)\ \Rightarrow\ \exists y':\mathsf{U}\,.\,R(x',y')\ \wedge\ B(y',y)\big) \quad \wedge \\ \qquad \big(\forall y,y',x:\mathsf{U}\,.\quad B(y',y)\ \Rightarrow\ R(x,y)\ \Rightarrow\ \exists x':\mathsf{U}\,.\,R(x',y')\ \wedge\ A(x',x)\big) \quad \wedge \\ \qquad R(a,b) \end{array}$$

$$\begin{array}{l} \mathsf{ELT}' \quad : \quad (\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \to (\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda(A,a)\,.\,\lambda(B,b)\,.\quad \exists b':\mathsf{U}\,.\quad \mathsf{EQV}'(A,a,B,b')\ \wedge\ B(b',b)\,. \end{array}$$

$$\begin{array}{l} i' \quad : \quad (\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop}) \to \mathsf{U} \to \mathsf{U} \quad := \\ \quad \lambda(A,a).\quad \lambda f:(\Pi X\,.\,(X{\to}X{\to}\mathsf{Prop}) \to X \to \mathsf{Prop})\,.\quad f(\mathsf{U},A,a). \end{array}$$

Notons que les fonctions monomorphes EQV', ELT' et i' sont caractérisées en fonction de leurs équivalents polymorphes EQV, ELT et i (définis au paragraphe 8.4.3) par les égalités intensionnelles suivantes

$$\begin{array}{lcll} \mathsf{EQV}' & =_\beta & \lambda(A,a)\,.\,\lambda(B,b)\,. & \mathsf{EQV}(\mathsf{U},A,a,\mathsf{U},B,b) \\ \mathsf{ELT}' & =_\beta & \lambda(A,a)\,.\,\lambda(B,b)\,. & \mathsf{ELT}(\mathsf{U},A,a,\mathsf{U},B,b) \\ i' & =_\beta & \lambda(A,a)\,. & i(\mathsf{U},A,a) \end{array}$$

dont les membres droits sont des termes bien formés dans le système U^-. (Là encore, il suffit de remarquer que les définitions des fonctions polymorphes EQV, ELT et i ne font pas intervenir la quantification de type $\overline{\forall}\alpha\,.\,\phi$.)

Les relations monomorphes EQV' et ELT' vérifient les mêmes propriétés que les relations polymorphes EQV et ELT, à cette différence près que ces propriétés s'expriment et se prouvent maintenant dans le système U^-. Par exemple, la réflexivité de la relation de bissimilarité s'écrit à présent

$$\forall A:(\mathsf{U}{\to}\mathsf{U}{\to}\mathsf{Prop})\,.\,\forall a:\mathsf{U}\,.\qquad \mathsf{EQV}'(A,a,A,a),$$

alors que dans le cas polymorphe, l'expression de cette même propriété nécessite l'utilisation de la quantification de type induite par la règle de formation $(\mathsf{Kind},\mathsf{Prop})$:

$$\overline{\forall}X\,.\,\forall A:(X{\to}X{\to}\mathsf{Prop})\,.\,\forall a:X\,.\qquad \mathsf{EQV}(X,A,a,X,A,a).$$

On remarquera cependant que si la plupart des preuves se transposent dans le système U^- simplement en remplaçant les termes ELT, EQV et i par les termes EQV', ELT' et i' tout en faisant disparaître les quantifications de types $\overline{\forall}\alpha\,.\,\phi$, la preuve d'injectivité de la fonction i' donnée par le terme de preuve

$$\begin{array}{l}\lambda(A,a),(B,b)\,.\quad \lambda\xi^{i(A,a)=_{\mathsf{U}} i(B,b)}.\quad \xi\,\Big(u\,\big(\lambda(Z,C,c)\,.\mathsf{EQV}(\mathsf{U},A,a,Z,C,c)\big)\Big)\;\mathsf{refl}'(A,a)\\ \quad :\quad \forall(A,a),(B,b)\,.\quad i(A,a)=_{\mathsf{U}} i(B,b)\quad\Rightarrow\quad \mathsf{EQV}'(A,a,B,b)\end{array}$$

déroge à cette règle puisqu'elle utilise de manière explicite la fonction polymorphe EQV. Heureusement, ceci ne pose aucun problème puisque le terme EQV est bien formé dans le système U^- et n'intervient ici que dans un contexte où la preuve de réflexivité

$$\mathsf{refl}'(A,a)\quad :\quad \mathsf{EQV}'(A,a,A,a)\quad =_\beta\quad \mathsf{EQV}(\mathsf{U},A,a,\mathsf{U},A,a)$$

n'est utilisée que de manière monomorphe.

Le paradoxe de Russell dans le système U^- Comme dans le cas polymorphe, on définit deux relations binaires

$$\mathsf{eqv}',\mathsf{elt}'\quad :\quad \mathsf{U}\to\mathsf{U}\to\mathsf{Prop}$$

en transportant dans le type U les relations EQV' et ELT' (définies sur la classe des graphes pointés (A,a)) au moyen de l'injection i' définie ci-dessus. De même, on définit la classe des ensembles en introduisant un prédicat $\mathsf{set}' : \mathsf{U}\to\mathsf{Prop}$ défini par

$$\mathsf{set}'\quad :=\quad \lambda u:\mathsf{U}\,.\quad \exists(A,a)\,.\quad u=_{\mathsf{U}} i'(A,a).$$

Les propriétés du prédicat set' et des relations binaires eqv' et elt' sont les mêmes que celles que nous avons établies au paragraphe 8.4.3 pour les fonctions set, eqv et elt. La transposition du reste de la démonstration s'effectue de manière analogue sans la moindre difficulté, ce qui nous permet de reformuler le paradoxe de Russell établi au paragraphe précédent dans le système U^-, d'où il ressort que :

Proposition 8.4.17 (Incohérence de λU^-) — *Dans le système U^-, toutes les propositions sont prouvables :*

$$(\bot)\qquad\qquad \forall a:\mathsf{Prop}\,.\,a.$$

Chapitre 9

Un modèle de la théorie des ensembles de Zermelo dans $F\omega.3$

Au cours du chapitre précédent, nous avons vu que la représentation des ensembles par des graphes pointés permet de traduire toutes les preuves du fragment intuitionniste de la théorie des ensembles de Frege dans les systèmes U et U^-, ce qui nous a permis de redémontrer l'incohérence de ces deux systèmes à partir d'une traduction du paradoxe de Russell. Cependant, le paradigme "ensemble = graphe pointé" a une portée bien plus générale, puisqu'il permet entre autres de démontrer la non-contradiction (relative) de l'axiome d'anti-fondation.

Dans ce chapitre, nous nous proposons de reprendre les mêmes idées, mais en les utilisant cette fois-ci dans le cadre d'une théorie des types (supposée) cohérente. Pour cela, nous allons nous placer dans la théorie des types de Church avec deux univers (*i.e.* le système $F\omega.3$, qui est un sous-système de CCω) et construire dans un type de ce système un modèle de la théorie des ensembles de Zermelo classique. De cette construction, nous déduirons le résultat d'expressivité logique suivant :

> *La puissance théorique du système $F\omega.3$ est au moins aussi grande que celle de la théorie des ensembles de Zermelo d'ordre supérieur :* $\mathrm{Z}\omega \leq F\omega.3$.

De ce résultat découle naturellement le fait que le système $F\omega.3$ est strictement plus fort que la théorie des ensembles de Zermelo (*i.e.* $\mathrm{Z} < F\omega.3$), ce qui nous permet de répondre positivement à un problème posé dans [17] par Thierry Coquand, qui conjecturait l'inégalité stricte $\mathrm{Z} < \mathrm{CC}\omega$ que notre résultat affine très sensiblement.

Il est intéressant de comparer notre résultat avec les travaux de Peter Aczel [5] et de Benjamin Werner [64], qui reposent très fortement sur la présence d'un mécanisme primitif de définitions inductives et d'une formulation (plus ou moins explicite) de l'axiome du choix. Ici, nous nous plaçons dans un système de types purs au sens le plus strict du terme, sans relation de cumulativité entre les univers prédicatifs et sans entiers primitifs.

Bien entendu, le choix d'un formalisme minimaliste a des conséquences très fortes quant à l'expressivité de la théorie des ensembles qu'il est possible de modéliser. En pratique, cela se traduit par la perte définitive du schéma d'axiomes de remplacement, qui confère à la théorie de Zermelo-Fränkel une puissance théorique qui se situe à des lieues au-dessus de celle de la théorie des ensembles de Zermelo.[1]

[1] On notera par ailleurs que l'impossibilité de dériver le schéma de remplacement dans $F\omega.3$ est plutôt une

Il est par ailleurs important de préciser que toutes les preuves formelles effectuées dans le système $F\omega.3$ (et que nous présenterons dans les sections 9.2, 9.3, 9.4 et 9.5) ont été construites et vérifiées sur machine. Les preuves ont d'abord été effectuées dans une version bridée du système Coq (*i.e.* sans utiliser la librairie standard et en n'utilisant qu'un nombre très réduit de tactiques de manière à contrôler la structure des termes de preuves), puis ont été ensuite revérifiées dans un système indépendant (basé sur la théorie des PTS) de manière à nous assurer de la bonne formation des termes obtenus dans le système $F\omega.3$ (sans cumulativité).

9.1 La théorie des types de Church avec deux univers

9.1.1 Le système de types purs $\lambda\omega.3$

La théorie des types de Church avec deux univers (notée $F\omega.3$ ou $\lambda\omega.3$) est le système de types purs à quatre sortes

$$\mathcal{S} \quad = \quad \{\mathsf{Prop};\ \mathsf{Type}_1;\ \mathsf{Type}_2;\ \mathsf{Type}_3\}$$

dont les axiomes sont donnés par

$$\mathbf{Axiom} \quad = \quad \{(\mathsf{Prop} : \mathsf{Type}_1);\ (\mathsf{Type}_1 : \mathsf{Type}_2);\ (\mathsf{Type}_2 : \mathsf{Type}_3)\}$$

et dont les règles de formation de produits dépendants sont définies par

$$\begin{array}{rcll} \mathbf{Rule} & = & \{(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop})\} & \cup \quad \text{(implication)} \\ & & \{(\mathsf{Type}_i, \mathsf{Prop}, \mathsf{Prop})\} & \cup \quad \text{(quantification universelle)} \\ & & \{(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{\max(i,j)})\} & \quad \text{(produit dépendant prédicatif)} \end{array}$$

où i et j décrivent toutes les combinaisons possibles des entiers 1, 2 et 3.

Intuitivement, cette théorie est le système $F\omega$ (*i.e.* la théorie des types simples de Church, dans sa version minimale intuitionniste) que l'on a étendu en ajoutant à la sorte Type (maintenant notée Type_1) deux autres sortes prédicatives Type_2 et Type_3 situées au-dessus d'elle.

Dans ce système, la sorte maximale Type_3 — qui est par définition mal typée — joue un rôle très similaire à celui de la sorte Type du Calcul des Constructions ou du système $F\omega$. En pratique, cette sorte n'apparaît que dans les jugements de la forme $\Gamma \vdash T : \mathsf{Type}_3$, où l'occurrence indiquée est la seule occurrence possible. Pour cette raison, on réserve la terminologie d'*univers* aux sortes Type_1 et Type_2 uniquement, d'où le nom de *théorie des types de Church avec deux univers* pour désigner un formalisme noté par ailleurs $\lambda\omega.3$.[2]

S'il est possible de former des produits dépendants dont les composantes vivent dans des sortes prédicatives différentes (en utilisant la règle $(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{\max(i,j)})$), la formulation

bonne nouvelle concernant la cohérence de cette théorie (ainsi que celle de ZF) : si nous avions pu dériver ce schéma d'axiomes, nous aurions construit *de facto* un modèle de ZF dans une théorie dont on peut par ailleurs prouver la cohérence dans ZF, ce qui nous aurait permis d'établir l'incohérence de ces deux systèmes d'après le second théorème d'incomplétude de Gödel.

[2]La notation $\lambda\omega.3$ correspond ici à la multiplication ordinale ($\omega.3 = \omega + \omega + \omega$) et est inspirée du fait que le modèle booléen ensembliste de cette théorie peut être défini dans l'ensemble $V_{\omega.3}$. On remarquera que cet ensemble contient l'ensemble $V_{\omega.2}$, qui est le modèle extensionnel standard de la théorie des ensembles de Zermelo dans ZF [39]. La recherche d'un modèle de la théorie des ensembles de Zermelo dans $\lambda\omega.3$ est donc cohérente vis-à-vis de cette première estimation en termes de théorie des modèles.

de la théorie que nous venons de présenter ne comporte pas de règle de cumulativité pour les univers prédicatifs, contrairement au Calcul des Constructions avec univers dont $F\omega.3$ est un sous-système. Dans ce qui suit, nous resterons dans le cadre strict de la théorie des PTS et nous ne supposerons pas que les types formés dans la sorte Type_i (pour $i = 1$ ou $i = 2$) sont également bien formés dans la sorte Type_{i+1}.

Comme par ailleurs, le système $F\omega.3$ est un PTS fonctionnel [26], le type de n'importe quel terme (dans un contexte donné) est unique à β-conversion près, et la règle de renforcement est admissible dans le système.

9.1.2 Une présentation stratifiée

À l'instar du système $F\omega$, le système $F\omega.3$ est un PTS sans *types dépendants*, puisqu'il ne dispose pas de la règle $(\mathsf{Prop}, \mathsf{Type}_i, \mathsf{Type}_i)$[3] qui permettrait de définir des types dépendant de termes de preuves.

Pour cette raison, nous travaillerons avec une présentation stratifiée du formalisme dans laquelle on séparera la partie prédicative du système (constituée des termes dont le type est dans une sorte prédicative Type_i) de sa partie logique et imprédicative (constituée des termes de preuves, dont le type est une proposition). Ainsi, on appellera

- *terme fonctionnel* (ou *terme*) tout terme M dont le type T est formé dans une des sortes Type_1, Type_2 ou Type_3 ;
- *terme de preuve* (ou *preuve*) tout terme M dont le type T est formé dans la sorte Prop.

Dans la suite de cette section, on utilisera les lettres M, N, T, U, ϕ, ψ — les lettres ϕ, ψ étant réservées aux propositions — pour désigner les termes (fonctionnels), tandis que les lettres t, u seront réservées aux (termes de) preuves. De même, on distinguera syntaxiquement les variables de termes des variables de preuves en notant les premières avec les lettres x, y, z et les secondes avec les lettres ξ, ξ'.

Le produit non-dépendant $\Pi \_ : \phi \,.\, \psi$ formé à l'aide de la règle $(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop})$ sera noté $\phi \Rightarrow \psi$, et le produit dépendant imprédicatif propositionnel $\Pi x : T \,.\, \phi$ (formé à l'aide de la règle $(\mathsf{Type}_i, \mathsf{Prop}, \mathsf{Prop})$) sera noté $\forall x : T \,.\, \phi$. Les produits dépendants ou non-dépendants formés dans les sortes prédicatives Type_i seront notés de manière standard $\Pi x : T \,.\, U$ et $T \to U$ respectivement (où $T \to U$ est une abréviation de $\Pi x : T \,.\, U$ dans le cas où $x \notin FV(U)$).

Séparation des contextes Soit Δ un contexte bien formé dans $F\omega.3$. Pour chaque déclaration $(x : T) \in \Delta$ (resp. $(\xi : \phi) \in \Delta$), le type T (resp. la proposition ϕ) ne dépend que de variables de termes déclarées auparavant dans le même contexte. Quitte à réordonner les déclarations figurant dans le contexte Δ[4], il est toujours possible de supposer que Δ est de la forme

$$\Delta \quad \equiv \quad \Sigma; \Gamma,$$

où Σ ne contient que des déclarations de variables de termes et où Γ ne contient que des déclarations de variables de preuves.

[3] Ou d'une règle de cumulativité $\mathsf{Prop} \subset \mathsf{Type}_i$ qui subsume la règle $(\mathsf{Prop}, \mathsf{Type}_i, \mathsf{Type}_i)$ en conjonction avec la règle $(\mathsf{Type}_i, \mathsf{Type}_i, \mathsf{Type}_i)$. Rappelons qu'en ajoutant à $F\omega.3$ les inclusions $\mathsf{Prop} \subset \mathsf{Type}_1$, $\mathsf{Type}_1 \subset \mathsf{Type}_2$ et $\mathsf{Type}_2 \subset \mathsf{Type}_3$, on obtient le Calcul des Constructions avec deux univers, noté CC_3.

[4] Ce qui est possible en utilisant le lemme d'échange [26] qui est une conséquence de la règle de renforcement.

On remarquera que l'ordre dans lequel les variables de termes sont déclarées dans Σ est important (en raison des dépendances qui peuvent apparaître entre les types de ces variables) alors qu'il ne l'est pas pour les variables de preuves déclarées dans Γ, dont les types ne peuvent dépendre que des variables de termes déclarées dans Σ. Dans ce qui suit, on parlera de *signature*[5] pour faire référence au préfixe Σ du contexte Δ (dans lequel sont déclarées les variables de termes), tandis qu'on appellera *contexte logique*, ou plus simplement *contexte*, le suffixe Γ du contexte Δ (dans lequel sont déclarées toutes les variables de preuves). Le contexte complet $\Delta \equiv \Sigma; \Gamma$ sera alors noté $\Delta \equiv \langle\Sigma\rangle\Gamma$.

La partie fonctionnelle La syntaxe de la partie fonctionnelle (prédicative) du système $F\omega.3$ est donnée par :

$$\begin{array}{lrcl}
\textbf{Termes} & M, N, T, U, \phi, \psi & ::= & x \mid \mathsf{Prop} \mid \mathsf{Type}_1 \mid \mathsf{Type}_2 \mid \mathsf{Type}_3 \\
& & \mid & \Pi x\!:\!T\,.\,U \mid \lambda x\!:\!T\,.\,M \mid M\ N \\
& & \mid & \phi \Rightarrow \psi \mid \forall x\!:\!T\,.\,\phi \\
\textbf{Signatures} & \Sigma & ::= & [] \mid \Sigma; [x:T]
\end{array}$$

Pour des raisons de commodité, nous avons inclus la sorte Type_3 dans la syntaxe des termes bien que cette sorte ne soit pas un objet bien typé dans le formalisme. Le système de types correspondant est bâti à partir de deux jugements

- $\Sigma \vdash$ ("la signature Σ est bien formée") et
- $\Sigma \vdash M : T$ ("dans la signature Σ, le terme M a pour type le terme T")

dont les règles d'inférence sont les suivantes :

$$\frac{}{[] \vdash} \qquad \frac{\Sigma \vdash T : \mathsf{Type}_i \qquad x \notin DV(\Sigma)}{\Sigma; [x:T] \vdash}$$

$$\frac{\Sigma \vdash \qquad (x:T) \in \Sigma}{\Sigma \vdash x : T} \qquad \frac{\Sigma \vdash}{\Sigma \vdash \mathsf{Prop} : \mathsf{Type}_1} \qquad \frac{\Sigma \vdash \qquad i \in \{1;2\}}{\Sigma \vdash \mathsf{Type}_i : \mathsf{Type}_{i+1}}$$

$$\frac{\Sigma \vdash T : \mathsf{Type}_i \qquad \Sigma; [x:T] \vdash U : \mathsf{Type}_j}{\Sigma \vdash \Pi x\!:\!T\,.\,U : \mathsf{Type}_{\max(i,j)}}$$

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

$$\frac{\Sigma \vdash \phi : \mathsf{Prop} \qquad \Sigma \vdash \psi : \mathsf{Prop}}{\Sigma \vdash \phi \Rightarrow \psi : \mathsf{Prop}} \qquad \frac{\Sigma; [x:T] \vdash \phi : \mathsf{Prop}}{\Sigma \vdash \forall x\!:\!T\,.\,\phi : \mathsf{Prop}}$$

$$\frac{\Sigma \vdash M : T \qquad \Sigma \vdash T' : \mathsf{Type}_i \qquad T =_\beta T'}{\Sigma \vdash M : T'}$$

Dans cette présentation, la constante Prop a perdu son statut de sorte pour devenir un type de données comme les autres, puisqu'il n'est pas possible d'introduire dans une signature

[5] Par analogie avec les signatures caractérisant les différentes théories du premier ordre.

une hypothèse dont le type est une proposition. Ce changement de statut justifie la présence de notations spécifiques pour désigner l'implication $\phi \Rightarrow \psi$ et la quantification universelle $\forall x : T \,.\, \phi$, qui sont maintenant des constructions distinctes des produits dépendants et non-dépendants formés dans les sortes prédicatives.

La partie logique La syntaxe de la partie logique (imprédicative) du système $F\omega.3$ constituée des termes de preuves et des contextes logiques est donnée par

$$\begin{array}{lllll}
\textbf{Preuves} & t, u & ::= & \xi & \text{(axiome)} \\
 & & | & \lambda \xi^{\phi} . \, t \quad | \quad t \; u & (\Rightarrow\text{-intro, } \Rightarrow\text{-elim}) \\
 & & | & \lambda x : T \,.\, t \quad | \quad t \; M & (\forall\text{-intro, } \forall\text{-elim}) \\
\textbf{Contextes} & \Gamma & ::= & [] \quad | \quad \Gamma ; [\xi : \phi]
\end{array}$$

Le système de typage des termes de preuves est bâti autour de deux jugements

- $\langle \Sigma \rangle \Gamma \vdash$ ("sous la signature Σ, le contexte logique Γ est bien formé") et
- $\langle \Sigma \rangle \Gamma \vdash t : \phi$ ("sous le contexte $\langle \Sigma \rangle \Gamma$, t est une preuve de la proposition ϕ")

dont les règles d'inférences sont les suivantes :

$$\frac{\Sigma \vdash}{\langle \Sigma \rangle [] \vdash} \qquad \frac{\langle \Sigma \rangle \Gamma \vdash \qquad \Sigma \vdash \phi : \mathsf{Prop} \qquad \xi \notin DV(\Gamma)}{\langle \Sigma \rangle \Gamma ; [\xi : \phi] \vdash}$$

$$\frac{\langle \Sigma \rangle \Gamma \vdash \qquad (\xi : \phi) \in \Gamma}{\langle \Sigma \rangle \Gamma \vdash \xi : \phi}$$

$$\frac{\langle \Sigma \rangle \Gamma ; [\xi : \phi] \vdash t : \psi}{\langle \Sigma \rangle \Gamma \vdash \lambda \xi^{\phi} . \, t : \phi \Rightarrow \psi} \qquad \frac{\langle \Sigma \rangle \Gamma \vdash t : \phi \Rightarrow \psi \qquad \langle \Sigma \rangle \Gamma \vdash u : \phi}{\langle \Sigma \rangle \Gamma \vdash t \; u : \psi}$$

$$\frac{\langle \Sigma ; [x : T] \rangle \Gamma \vdash t : \phi \qquad x \notin FV(\Gamma)}{\langle \Sigma \rangle \Gamma \vdash \lambda x : T \,.\, t : \forall x : T \,.\, \phi} \qquad \frac{\langle \Sigma \rangle \Gamma \vdash t : \forall x : T \,.\, \phi \qquad \Sigma \vdash M : T}{\langle \Sigma \rangle \Gamma \vdash t \; M : \phi \{ x := M \}}$$

$$\frac{\langle \Sigma \rangle \Gamma \vdash t : \phi \qquad \Sigma \vdash \phi' : \mathsf{Prop} \qquad \phi =_{\beta} \phi'}{\langle \Sigma \rangle \Gamma \vdash t : \phi'}$$

Notons que la séparation des contextes (au sens des PTS) en deux entités distinctes — les signatures et les contextes logiques — nécessite d'introduire une condition de bord $x \notin FV(\Gamma)$ dans la règle d'introduction du quantificateur universel, qui exprime que la variable $x : T$ sur laquelle porte l'introduction n'apparaît pas dans les variables libres des propositions ϕ_i figurant dans le contexte logique Γ. Cette condition de bord (standard dans le cadre des théories du premier ordre) s'explique par le fait que la variable $x : T$ sur laquelle porte la généralisation n'est pas nécessairement la dernière hypothèse introduite dans le contexte $\langle \Sigma \rangle \Gamma$, puisque le contexte logique Γ peut être non vide.[6]

[6]Dans la théorie des systèmes de types purs, cette condition de bord se justifie de la manière suivante. Supposons $\Sigma ; [x : T] ; \Gamma \vdash t : \phi$, avec $x \notin \Gamma$. Grâce à cette condition de bord, il est possible d'échanger la déclaration $(x : T)$ avec chacune des hypothèses $(\xi : \phi)$ figurant dans le contexte logique Γ de manière à la repousser à l'extrémité du contexte $\Sigma ; \Gamma$. Ces échanges successifs de déclarations (justifiés par l'admissibilité de la règle de renforcement [26]) aboutissent au jugement dérivable $\Sigma ; \Gamma ; [x : T] \vdash t : \phi$, auquel on applique ensuite la règle d'introduction du produit dépendant (dont la bonne formation est assurée par la règle $(\mathsf{Type}_i, \mathsf{Prop}, \mathsf{Prop})$) pour dériver le jugement $\Sigma ; \Gamma \vdash \lambda x : T \,.\, t : \forall x : T \,.\, \phi$.

Les propriétés du système Le système de types prédicatif basé sur les deux jugements $\Sigma \vdash$ et $\Sigma \vdash M : T$ vérifie les propriétés standard telles que

- la propriété de préservation du typage par β-réduction ;
- les règles admissibles d'affaiblissement et de renforcement ;
- la propriété de normalisation forte ;
- la décidabilité de la vérification et de l'inférence de type.

La partie logique du formalisme (basée sur les jugements $\langle\Sigma\rangle\Gamma \vdash$ et $\langle\Sigma\rangle\Gamma \vdash t : \phi$) vérifie les propriétés analogues (préservation du typage des preuves par β-réduction, renforcement, élimination des coupures, décidabilité des relations $\langle\Sigma\rangle\Gamma \vdash$ et $\langle\Sigma\rangle\Gamma \vdash t : \phi$) ainsi que quelques autres propriétés logiques spécifiques telles que

- l'admissibilité des règles d'affaiblissement et de contraction (cette dernière étant une conséquence du lemme de substitutivité dans les PTS) ;
- la possibilité d'échanger l'ordre dans lequel les déclarations $(\xi : \phi)$ apparaissent dans un contexte logique Γ donné.

La plupart de ces propriétés sont des conséquences immédiates de la théorie des PTS (transposées au cadre de la présentation stratifiée que nous venons d'effectuer). Les propriétés de normalisation forte (pour la partie fonctionnelle) et d'élimination des coupures (pour la partie logique) sont des cas particuliers du théorème de normalisation forte dans le Calcul des Constructions avec univers dont $F\omega.3$ est un sous-système.

9.1.3 La logique intuitionniste de $F\omega.3$

Comme dans la théorie des types simples de Church (qui est un sous-système de $F\omega.3$), les constantes logiques $\top$ et $\bot$, les connecteurs logiques $\neg$, $\wedge$ et $\vee$, la quantification existentielle $\exists x : T \,.\, \phi$ et l'égalité de Leibniz $M =_T N$ (où T est un type formé dans une des sortes Type_1, Type_2 ou Type_3) sont définissables à partir des constructions primitives $\phi \Rightarrow \psi$ et $\forall x : T \,.\, \phi$ en utilisant les codages imprédicatifs standards que nous avons déjà présentés dans le chapitre précédent. Bien entendu, ces différentes constructions s'accompagnent des schémas d'introduction et d'élimination correspondants, qui constituent des propositions prouvables dans le formalisme (*cf* paragraphe 8.1.4).

Notons que la quantification existentielle $\exists x : T \,.\, \phi$ et l'égalité de Leibniz $M =_T N$ sont définies chacune sur trois niveaux différents, suivant que le type T est dans la sorte Type_1, dans la sorte Type_2 ou dans la sorte Type_3 (à l'instar de la quantification universelle $\forall x : T \,.\, \phi$, qui est définie pour n'importe quel type T habitant l'une de ces trois sortes).

9.2 Un type prouvablement infini dans Type_2

Dans cette section, nous nous proposons de construire un type de données dans Type_2 dont nous allons montrer (dans le formalisme lui-même) qu'il est infini. Pour cela, nous allons définir un codage prédicatif du type des entiers de Church, et nous verrons qu'à travers ce codage, les axiomes de Peano sont prouvables dans $F\omega.3$. Cette construction est une étape importante dans la définition du modèle de Z que nous allons présenter, puisque c'est elle qui va nous permettre de réaliser l'axiome de l'infini de la théorie des ensembles.

Il est important de remarquer que dans $F\omega.3$ — qui est une théorie des types ne disposant ni d'entiers primitifs ni de types inductifs primitifs — la logique imprédicative et la présence d'univers sont cruciales pour accéder à l'infini réalisé. En effet, dans des théories des types imprédicatives sans univers telles que le système $F\omega$ ou le Calcul des Constructions, il n'est pas possible de définir le moindre type prouvablement infini, à moins bien entendu de poser les axiomes correspondants.[7]

C'est pour cette même raison que nous définirons le type nat dans la sorte Type_2 et non dans la sorte Type_1. En effet, nous avons vu au chapitre 3 qu'il est possible d'interpréter dans CCω (qui contient $F\omega.3$) tous les types de la sorte Type_1 par des ensembles héréditairement finis. Dans le formalisme considéré, ceci se traduit par le fait que toute tentative d'accéder à l'infini réalisé dans la sorte Type_1 est vouée à l'échec.

9.2.1 Le type des entiers de Church dans Type_2

Le type nat des entiers de Church est défini dans la sorte Type_2 par :

$$\mathsf{nat} \quad : \quad \mathsf{Type}_2 \qquad := \qquad \Pi X : \mathsf{Type}_1 \,.\, X \to (X \to X) \to X.$$

Comme dans le codage imprédicatif standard, chaque entier naturel $n \in \omega$ est représenté dans le type nat par l'itérateur polymorphe $\overline{n}$ défini par :

$$\overline{n} \quad \equiv \quad \lambda X : \mathsf{Type}_1 \,.\, \lambda x : X \,.\, \lambda f : (X \to X) \,.\, \underbrace{f(\cdots f}_{n}(x)\cdots) \quad : \quad \mathsf{nat}.$$

Cependant, l'utilisation d'un codage prédicatif introduit une différence notable par rapport au codage imprédicatif du système F, qui est que dans $F\omega.3$, la sorte dans laquelle est définie le type nat (ici la sorte Type_2) se situe un niveau au-dessus de la sorte parcourue par la variable de type X (ici la sorte Type_1) exprimant le polymorphisme de l'itérateur $\overline{n} : \mathsf{nat}$.

Par conséquent, s'il est possible d'utiliser l'itérateur polymorphe $\overline{n} : \mathsf{nat}$ pour itérer n fois n'importe quelle fonction définie sur un type X formé dans la sorte Type_1, il n'est pas possible d'utiliser cet itérateur pour itérer une fonction définie sur le type nat lui-même, puisque celui-ci se situe dans la sorte prédicative supérieure. Nous verrons au paragraphe 9.2.2 que cette restriction rend sensiblement plus complexe la définition de la fonction prédécesseur, dont la construction est une étape cruciale dans la preuve d'injectivité de la fonction successeur.

La constante $\mathsf{0} : \mathsf{nat}$ et la fonction successeur $\mathsf{S} : \mathsf{nat} \to \mathsf{nat}$ sont définies de la manière habituelle en posant :

$$\begin{array}{lcll} \mathsf{0} & : & \mathsf{nat} & := \quad \lambda X : \mathsf{Type}_1 \,.\, \lambda x : X \,.\, \lambda f : (X \to X) \,.\, x \\ \mathsf{S} & : & \mathsf{nat} \to \mathsf{nat} & := \quad \lambda n : \mathsf{nat} \,.\, \lambda X : \mathsf{Type}_1 \,.\, \lambda x : X \,.\, \lambda f : (X \to X) \,.\, f(n(X,x,f)). \end{array}$$

Ces deux définitions sont justifiées par le fait que pour tout entier naturel $n \in \omega$ on a l'égalité intensionnelle :

$$\underbrace{\mathsf{S}(\cdots \mathsf{S}(}_{n}\,\mathsf{0})\cdots) \quad =_\beta \quad \overline{n} \quad \equiv \quad \lambda X : \mathsf{Type}_1 \,.\, \lambda x : X \,.\, \lambda f : (X \to X) \,.\, \underbrace{f(\cdots f}_{n}(x)\cdots).$$

[7] De fait, la cohérence logique de $F\omega$ ou du Calcul des Constructions (sans univers) peut être établie dans l'arithmétique de Peano. D'un point de vue logique, ces deux théories des types sont donc des théories "infra-arithmétiques". En revanche, la preuve du theorème de normalisation forte (pour ces deux théories) ne peut pas être effectuée dans une théorie plus faible que la théorie des ensembles de Zermelo (classique ou intuitionniste), qui est sensiblement plus forte que l'arithmétique d'ordre supérieur (HAω ou PAω).

Cependant, le fait que tous les entiers de Church $\mathsf{S}(\cdots\mathsf{S}(\mathsf{0})\cdots)$ soient des termes de type nat ne signifie pas que ce sont les seuls habitants de ce type (à β-convertion près),[8] puisque le principe de récurrence n'est pas dérivable à partir de notre codage. Pour remédier à ce problème, il est nécessaire d'introduire un prédicat $\mathsf{wf\_nat} : \mathsf{nat} \to \mathsf{Prop}$ définissant la plus petite classe sur nat contenant $\mathsf{0}$ et stable par la fonction S. Le prédicat $\mathsf{wf\_nat}$ est défini par un codage imprédicatif standard qui est le suivant :

$$\begin{array}{lcl} \mathsf{wf\_nat} & : & \mathsf{nat} \to \mathsf{Prop} \quad := \\ & & \lambda n : \mathsf{nat}\,.\quad \forall P : (\mathsf{nat} \to \mathsf{Prop})\,. \\ & & \qquad P(\mathsf{0}) \;\Rightarrow\; \big(\forall p : \mathsf{nat}\,.\quad P(p) \;\Rightarrow\; P(\mathsf{S}(p))\big) \;\Rightarrow\; P(n). \end{array}$$

Dans ce qui suit, nous dirons qu'un terme n de type nat est *un entier bien formé* s'il satisfait la proposition $\mathsf{wf\_nat}(n)$, et nous relativiserons les quantifications universelle et existentielle sur le type nat à l'aide du prédicat $\mathsf{wf\_nat}$ chaque fois que cela sera nécessaire.

Proposition 9.2.1 (Clôture par 0 et S) — *La classe des entiers bien formés contient* 0 *et est stable par la fonction successeur :*

1. $\mathsf{wf\_nat}(\mathsf{0})$ *;*
2. $\forall n : \mathsf{nat}\,.\quad \mathsf{wf\_nat}(n) \;\Rightarrow\; \mathsf{wf\_nat}(\mathsf{S}(n))$.

De plus, la classe des entiers bien formés vérifie le principe de récurrence donné par :

$$\begin{array}{l} \forall P : \mathsf{nat} \to \mathsf{Prop}\,. \\ \quad P(\mathsf{0}) \;\Rightarrow\; \big(\forall p : \mathsf{nat}\,.\quad \mathsf{wf\_nat}(p) \;\Rightarrow\; P(p) \;\Rightarrow\; P(\mathsf{S}(p))\big) \;\Rightarrow \\ \qquad \forall n : \mathsf{nat}\,.\quad \mathsf{wf\_nat}(n) \;\Rightarrow\; P(n). \end{array}$$

Preuve. Les points 1 et 2 résultent immédiatement de la définition du prédicat $\mathsf{wf\_nat}$. Pour démontrer le principe de récurrence, il suffit d'instancier l'hypothèse $\mathsf{wf\_nat}(n)$ avec le prédicat $Q : \mathsf{nat} \to \mathsf{Prop}$ défini par $Q \equiv \lambda p : \mathsf{nat}\,.\,\mathsf{wf\_nat}(p) \wedge P(p)$. (La preuve de ce principe, qui est en réalité caché dans l'hypothèse $\mathsf{wf\_nat}(n)$, est quasiment une tautologie.) □

Proposition 9.2.2 (Discrimination) — *Dans le type* nat*, le terme* $\mathsf{0}$ *est différent de n'importe quel terme de la forme* $\mathsf{S}(n)$ *:*

$$\forall n : \mathsf{nat}\,.\quad \neg\; \mathsf{0} =_{\mathsf{nat}} \mathsf{S}(n)$$

Preuve. On considère le prédicat $P : \mathsf{nat} \to \mathsf{Prop}$ défini par

$$P \qquad := \qquad \lambda n : \mathsf{nat}\,.\,n(\mathsf{Prop}, \top, \lambda x : \mathsf{Prop}\,.\,\bot),$$

et on remarque que les propositions $P(\mathsf{0})$ et $\top$ sont convertibles, de même que les propositions $P(\mathsf{S}(n))$ et $\bot$ quel que soit le terme $n : \mathsf{nat}$. Si p est une preuve de $\top$, on a alors

$$\lambda n : \mathsf{nat}\,.\,\lambda \xi^{\mathsf{0} =_{\mathsf{nat}} \mathsf{S}(n)}.\quad \xi\ P\ p \qquad : \qquad \forall n : \mathsf{nat}\,.\quad \mathsf{0} =_{\mathsf{nat}} \mathsf{S}(n) \;\Rightarrow\; \bot.$$ □

[8] Bien entendu, il est facile de montrer que les entiers de Church sont les seuls termes clos de type nat en forme normale, mais le raisonnement par analyse de la forme normale qui permet de le démontrer ne peut pas être effectué dans le formalisme lui-même.

9.2.2 Injectivité de la fonction successeur

Pour montrer l'injectivité de la fonction successeur, nous devons d'abord construire une fonction prédécesseur $\mathsf{pred} : \mathsf{nat} \to \mathsf{nat}$ satisfaisant les égalités intensionnelles

$$\mathsf{pred}(0) \;=_\beta\; 0 \qquad \text{et} \qquad \mathsf{pred}\Big(\underbrace{\mathsf{S}(\cdots\mathsf{S}}_{n+1}(0)\cdots)\Big) \;=_\beta\; \underbrace{\mathsf{S}(\cdots\mathsf{S}}_{n}(0)\cdots)$$

pour tout entier naturel $n \in \omega$.

La fonction prédécesseur dans le système F Dans le système F, la fonction prédécesseur est généralement construite à partir d'une fonction step de type $\mathsf{nat} \times \mathsf{nat} \to \mathsf{nat} \times \mathsf{nat}$ qui à chaque couple d'entiers $\langle n, m\rangle$ associe le couple $\langle m, \mathsf{S}(m)\rangle$.[9] En itérant n fois la fonction step sur le couple $\langle \mathsf{0}, \mathsf{0}\rangle$, on obtient alors le couple

$$\overline{n}\ (\mathsf{nat} \times \mathsf{nat})\ \langle \mathsf{0}, \mathsf{0}\rangle\ \mathsf{step} \qquad =_\beta \qquad \Big\langle \underbrace{\mathsf{S}(\cdots\mathsf{S}}_{n-1}(0)\cdots),\ \underbrace{\mathsf{S}(\cdots\mathsf{S}}_{n}(0)\cdots)\Big\rangle$$

(en convenant que $0-1$ désigne 0) dont la première composante est le prédécesseur de l'entier de Church $\overline{n}$. Il suffit alors de poser

$$\mathsf{pred} \quad : \quad \mathsf{nat} \to \mathsf{nat} \qquad := \qquad \lambda n : \mathsf{nat}\,.\,\pi_1(n\ (\mathsf{nat} \times \mathsf{nat})\ \langle \mathsf{0}, \mathsf{0}\rangle\ \mathsf{step})$$

(où $\pi_1(p)$ désigne la première projection du couple $p : \mathsf{nat} \times \mathsf{nat}$) pour obtenir la fonction qui a le comportement calculatoire souhaité.

Dans le système $F\omega.3$, tous les types de données sont définis dans les sortes prédicatives Type_1, Type_2 et Type_3, et il n'est pas possible de reprendre le codage imprédicatif du type $\mathsf{nat} \times \mathsf{nat}$ tel qu'il est défini dans le système F. Par ailleurs, le codage prédicatif des entiers de Church nous permet d'itérer n fois (avec $n : \mathsf{nat}$) n'importe quelle fonction définie sur un type de données X de type Type_1, mais ne nous permet pas d'itérer une fonction définie sur le type des entiers (lequel est défini dans la sorte Type_2), et encore moins une fonction définie sur le type des couples d'entiers.

Afin de contourner ce problème, nous allons baser la construction de la fonction prédécesseur non pas sur une fonction step de type $\mathsf{nat}\times\mathsf{nat} \to \mathsf{nat}\times\mathsf{nat}$, mais sur une famille de fonctions

$$\mathsf{step}(X, f) \quad : \quad X\times X \to X\times X$$

paramétrée par un type de données $X : \mathsf{Type}_1$ et une fonction $f : X \to X$ arbitraires, et qui à chaque couple $\langle x, y\rangle : X \times X$ associe le couple $\mathsf{step}(X, f)\langle x, y\rangle = \langle y, f(y)\rangle$. Pour cela, nous devons d'abord définir pour tout type $X : \mathsf{Type}_1$ la représentation du produit cartésien $X \times X$ dans la sorte Type_1.

[9] Rappelons que dans le système F, les types nat et $\mathsf{nat} \times \mathsf{nat}$ sont définis par $\mathsf{nat} \equiv \Pi X.X\to(X\to X)\to X$ et $\mathsf{nat} \times \mathsf{nat} \equiv \Pi X.(\mathsf{nat} \to \mathsf{nat} \to X) \to X$. Le couple $\langle n, m\rangle : (\mathsf{nat} \times \mathsf{nat})$ est défini par

$$\langle n, m\rangle \quad \equiv \quad \Lambda X.\lambda f : (\mathsf{nat} \to \mathsf{nat} \to X)\,.\,f\ n\ m,$$

et les projections $\pi_1(p), \pi_2(p) : \mathsf{nat}$ d'un couple $p : (\mathsf{nat} \times \mathsf{nat})$ sont définies par

$$\pi_1(p) \quad \equiv \quad p\ \mathsf{nat}\ (\lambda n, m : \mathsf{nat}\,.\,n) \qquad \text{et} \qquad \pi_2(p) \quad \equiv \quad p\ \mathsf{nat}\ (\lambda n, m : \mathsf{nat}\,.\,m).$$

Le type des paires dans Type_1 Considérons la fonction square de type $\mathsf{Type}_1 \to \mathsf{Type}_1$ définie par

$$\mathsf{square} \quad : \quad \mathsf{Type}_1 \to \mathsf{Type}_1 \qquad := \qquad \lambda X : \mathsf{Type}_1 \,.\, (X \to X \to X) \to X$$

et qui à chaque type $X : \mathsf{Type}_1$ associe le type $\mathsf{square}(X) =_\beta (X \to X \to X) \to X$.

Intuitivement, $X \to X \to X$ représente ici le type (noté bool_X) des booléens spécialisés sur le type X. Par conséquent, $\mathsf{square}(X)$ n'est autre que le type $\mathsf{bool}_X \to X$, dont les éléments sont les fonctions f associant à chaque booléen $b : \mathsf{bool}_X$ un objet de type X, et par lesquelles il est possible de représenter n'importe quel élément du produit cartésien $X \times X$.

Dans le cadre de cette représentation, le constructeur $\mathsf{pair}(X) : X \to X \to \mathsf{square}(X)$ et les projections $\mathsf{fst}(X), \mathsf{snd}(X) : \mathsf{square}(X) \to X$ sont donnés par :

$$\begin{array}{lcl}
\mathsf{pair} & : & \Pi X : \mathsf{Type}_1 \,.\, X \to X \to \mathsf{square}(X) \\
& := & \lambda X : \mathsf{Type}_1 \,.\, \lambda x, y : X \,.\, \lambda f : (X \to X \to X) \,.\, f(x, y) \\
\\
\mathsf{fst} & : & \Pi X : \mathsf{Type}_1 \,.\, \mathsf{square}(X) \to X \\
& := & \lambda X : \mathsf{Type}_1 \,.\, \lambda p : (X \to X \to X) \to X \,.\, p(\lambda x, _ : X \,.\, x) \\
\\
\mathsf{snd} & : & \Pi X : \mathsf{Type}_1 \,.\, \mathsf{square}(X) \to X \\
& := & \lambda X : \mathsf{Type}_1 \,.\, \lambda p : (X \to X \to X) \to X \,.\, p(\lambda _, y : X \,.\, y)
\end{array}$$

Il est intéressant de remarquer qu'en général, le type $\mathsf{square}(X)$ est une représentation approximative du produit cartésien $X \times X$ qui contient plus d'éléments que d'objets de la forme $\mathsf{pair}(X, x, y)$, où x et y décrivent le type X.[10] Cependant, ce codage satisfait les règles de réduction suivantes

$$\begin{array}{lcl}
\mathsf{fst}(X, \mathsf{pair}(X, M_1, M_2)) & \twoheadrightarrow_\beta & M_1 \\
\mathsf{snd}(X, \mathsf{pair}(X, M_1, M_2)) & \twoheadrightarrow_\beta & M_2
\end{array}$$

qui seront suffisantes pour nos besoins.

Le définition des fonctions step et pred Considérons le terme step défini par

$$\begin{array}{l}
\mathsf{step} \quad : \quad \Pi X : \mathsf{Type}_1 \,.\, (X \to X) \to \mathsf{square}(X) \to \mathsf{square}(X) \qquad := \\
\qquad \lambda X : \mathsf{Type}_1 \,.\, \lambda f : (X \to X) \,.\, \lambda p : \mathsf{square}(X) \,.\quad \mathsf{pair}\big(X, \mathsf{snd}(X, p), f(\mathsf{snd}(X, p))\big)
\end{array}$$

Pour chaque type $X : \mathsf{Type}_1$ et chaque fonction $f : X \to X$, le terme $\mathsf{step}(X, f)$ est une fonction de type $\mathsf{square}(X) \to \mathsf{square}(X)$ qui à chaque couple $\langle x, y \rangle : \mathsf{square}(X)$ (en notant $\langle x, y \rangle$ le terme $\mathsf{pair}(X, x, y)$) associe le couple $\mathsf{step}(X, f)\langle x, y \rangle = \langle y, f(y) \rangle : \mathsf{square}(X)$.

En itérant n fois (avec $n \in \omega$) la fonction $\mathsf{step}(X, f)$ sur un couple $\langle x, x \rangle$ construit à partir d'un objet $x : X$ arbitraire, on obtient alors le couple

$$\mathsf{step}(X, f)^n \langle x, x \rangle \quad =_\beta \quad \langle \underbrace{f(\cdots f}_{n-1}(x) \cdots), \ \underbrace{f(\cdots f}_{n}(x) \cdots) \rangle$$

[10] En effet, si dans le modèle ensembliste standard de $F\omega.3$ le type X est dénoté par un ensemble à n éléments, le type $\mathsf{square}(X)$ est quant à lui dénoté par un ensemble à $n^{(n^{2n})}$ éléments, ce qui est bien supérieur aux n^2 éléments du produit cartésien $X \times X$.

dont il suffit d'abstraire la première composante par rapport aux variables $X : \mathsf{Type}_1$, $x : X$ et $f : X \to X$ pour reconstruire l'entier de Church $\overline{n-1}$. Formellement, ceci nous permet de définir la fonction prédécesseur en posant :

$$\begin{aligned}\mathsf{pred} \quad &: \quad \mathsf{nat} \to \mathsf{nat} \qquad := \\ &\lambda n : \mathsf{nat} \,.\, \lambda X : \mathsf{Type}_1 \,.\, \lambda x : X \,.\, \lambda f : (X \to X) \,. \\ &\qquad \mathsf{fst}\Big(X, n\big(\mathsf{square}(X), \mathsf{pair}(X,x,x), \mathsf{step}(X,f)\big)\Big).\end{aligned}$$

Comme dans le système F, on a pour tout entier naturel $n \in \omega$ l'égalité intensionnelle

$$\mathsf{pred}(\overline{n}) \quad =_\beta \quad \overline{n-1}$$

(en convenant que $0 - 1 = 0$). En revanche, lorsque n désigne une variable de type nat, les termes $\mathsf{pred}(\mathsf{S}(n))$ et n ne sont pas β-convertibles (ce qui est déjà le cas dans le système F). D'un point de vue logique, ceci se traduit par le fait que, dans le système $F\omega.3$, la preuve de l'égalité $\mathsf{pred}(\mathsf{S}(n)) =_{\mathsf{nat}} n$ n'est pas une instance particulière de la réflexivité de l'égalité de Leibniz, mais que cette égalité ne peut être établie que par une récurrence sur l'entier de Church $n : \mathsf{nat}$, en utilisant explicitement l'hypothèse $\mathsf{wf\_nat}(n)$:

Lemme 9.2.3 (Prédécesseur du successeur) — *Pour tout entier n : nat bien formé, $\mathsf{pred}(\mathsf{S}(n))$ est égal à n :*

$$\forall n : \mathsf{nat} \,. \quad \mathsf{wf\_nat}(n) \quad \Rightarrow \quad \mathsf{pred}(\mathsf{S}(n)) =_{\mathsf{nat}} n.$$

Preuve. Ce résultat se démontre par récurrence sur n à l'aide de l'hypothèse $\mathsf{wf\_nat}(n)$. Le cas de base est immédiat, puisque les termes $\mathsf{pred}(\mathsf{S}(\mathsf{0}))$ et $\mathsf{0}$ sont convertibles. Supposons à présent que n est un entier bien formé tel que $\mathsf{pred}(\mathsf{S}(n)) =_{\mathsf{nat}} n$. De cette hypothèse découle l'égalité $\mathsf{S}(\mathsf{pred}(\mathsf{S}(n))) =_{\mathsf{nat}} \mathsf{S}(n)$, et comme par ailleurs les termes $\mathsf{S}(\mathsf{pred}(\mathsf{S}(n)))$ et $\mathsf{pred}(\mathsf{S}(\mathsf{S}(n))$ sont convertibles,[11] on a par conséquent $\mathsf{pred}(\mathsf{S}(\mathsf{S}(n))) =_{\mathsf{nat}} \mathsf{S}(n)$. □

Lemme 9.2.4 (Injectivité de la fonction successeur) — *La fonction successeur S est injective sur la classe des entiers bien formés :*

$$\forall n, m : \mathsf{nat} \,. \quad \mathsf{wf\_nat}(n) \quad \Rightarrow \quad \mathsf{wf\_nat}(m) \quad \Rightarrow \quad \mathsf{S}(n) =_{\mathsf{nat}} \mathsf{S}(m) \quad \Rightarrow \quad n =_{\mathsf{nat}} m.$$

Preuve. Si $\mathsf{S}(n) =_{\mathsf{nat}} \mathsf{S}(m)$, on a $\mathsf{pred}(\mathsf{S}(n)) =_{\mathsf{nat}} \mathsf{pred}(\mathsf{S}(m))$, d'où $n =_{\mathsf{nat}} m$ en vertu du lemme précédent (d'après les hypothèses $\mathsf{wf\_nat}(n)$ et $\mathsf{wf\_nat}(m)$). □

9.2.3 Quelques propriétés arithmétiques

Pour terminer cette section consacrée à l'arithmétique dans le type $\mathsf{nat} : \mathsf{Type}_2$, nous allons maintenant définir la relation d'ordre $n \leq m$ et établir ses principales propriétés, dont nous nous servirons au paragraphe 9.4.6 pour construire le graphe pointé représentant l'ensemble des entiers de von Neumann. La relation d'ordre $\mathsf{le} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{Prop}$ est définie par

$$\begin{aligned}\mathsf{le} \quad &: \quad \mathsf{nat} \to \mathsf{nat} \to \mathsf{Prop} \qquad := \\ &\lambda n, m : \mathsf{nat} \,. \quad \forall P : \mathsf{nat} \to \mathsf{Prop} \,. \\ &\qquad P(n) \quad \Rightarrow \quad \big(\forall p : \mathsf{nat} \,.\; P(p) \;\Rightarrow\; P(\mathsf{S}(p))\big) \quad \Rightarrow \quad P(m),\end{aligned}$$

[11] Pour ce type de vérification, qui est loin d'être immédiate à la main, l'aide d'un assistant à la démonstration est rapidement indispensable.

et on notera $n \leq m$ la proposition $\mathsf{le}(n, m)$.

La définition de la relation $n \leq m$ est très similaire à celle du prédicat $\mathsf{wf\_nat} : \mathsf{nat} \to \mathsf{Prop}$. En particulier, les termes $\mathsf{le}(\mathsf{0}, n)$ et $\mathsf{wf\_nat}(n)$ sont β-convertibles.

Proposition 9.2.5 (Propriétés de la relation $n \leq m$) — *La relation $n \leq m$ satisfait les propriétés suivantes :*

1.	$\forall n : \mathsf{nat}\,.$	$n \leq n$				
2.	$\forall n, m : \mathsf{nat}\,.$	$n \leq m$	$\Rightarrow$	$n \leq \mathsf{S}(m)$		
3.	$\forall n, m, p : \mathsf{nat}\,.$	$n \leq m$	$\Rightarrow$	$m \leq p$	$\Rightarrow$	$n \leq p$
4.	$\forall n, m : \mathsf{nat}\,.$	$\mathsf{wf\_nat}(n)$	$\Rightarrow$	$n \leq m$	$\Rightarrow$	$\mathsf{wf\_nat}(m)$
5.	$\forall n : \mathsf{nat}\,.$	$\mathsf{wf\_nat}(n)$	$\Rightarrow$	$\mathsf{0} \leq n$		
6.	$\forall n : \mathsf{nat}\,.$	$\neg\ \mathsf{S}(n) \leq \mathsf{0}$				

Preuve. Les points 1, 2 et 3 résultent immédiatement de la définition du terme le. Le point 4 est un cas particulier de 3 puisque les termes $\mathsf{wf\_nat}(n)$ et $\mathsf{0} \leq n$ sont β-convertibles, et le point 5 est une tautologie pour la même raison. Le point 6 se prouve en éliminant l'hypothèse $\mathsf{S}(n) \leq \mathsf{0}$ avec le prédicat $P : \mathsf{nat} \to \mathsf{Prop}$ défini par

$$P \quad \equiv \quad \lambda p : \mathsf{nat}\,.\ \ n(\mathsf{Prop}, \bot, \lambda\_ : \mathsf{Prop}\,.\,\top),$$

en remarquant que les propositions $P(\mathsf{0})$ et $\bot$ sont β-convertibles, de même que les propositions $P(\mathsf{S}(p))$ et $\top$ pour tout terme p de type nat. □

Lemme 9.2.6 (Inversion de la relation $n \leq m$) — *Si n et m sont deux termes de type nat tels que $n \leq m$, alors ou bien n et m sont égaux, ou bien il existe un terme $p : \mathsf{nat}$ tel que $n \leq p$ et $m =_{\mathsf{nat}} \mathsf{S}(p)$:*

$$\forall n, m : \mathsf{nat}\,.\quad n \leq m \quad \Rightarrow \quad n =_{\mathsf{nat}} m \quad \vee \quad \big(\exists p : \mathsf{nat}\,.\ \ n \leq p \ \wedge\ m =_{\mathsf{nat}} \mathsf{S}(p)\big).$$

Preuve. Cette proposition se démontre en éliminant l'hypothèse $n \leq m$ avec le prédicat $P : \mathsf{nat} \to \mathsf{Prop}$ défini par

$$P \quad \equiv \quad \lambda k : \mathsf{nat}\,.\quad n =_{\mathsf{nat}} k \quad \vee \quad \big(\exists p : \mathsf{nat}\,.\ \ n \leq p \ \wedge\ k =_{\mathsf{nat}} \mathsf{S}(p)\big).$$

Le cas de base est immédiat, et le cas inductif se démontre par disjonction de cas à l'aide de la proposition précédente. □

Proposition 9.2.7 — *Si n et m sont deux entiers bien formés tels que $n \leq \mathsf{S}(m)$, alors ou bien $n \leq m$, ou bien $n = \mathsf{S}(m)$:*

$$\begin{array}{l}\forall n, m : \mathsf{nat}\,.\ \ \mathsf{wf\_nat}(n) \ \Rightarrow\ \mathsf{wf\_nat}(m) \ \Rightarrow \\ \qquad n \leq \mathsf{S}(m) \quad \Rightarrow \quad n \leq m \ \ \vee \ \ n =_{\mathsf{nat}} \mathsf{S}(m).\end{array}$$

Preuve. Cette propriété est une conséquence immédiate de la proposition précédente. Notons que la démonstration utilise l'injectivité de la fonction successeur, d'où la nécessité de supposer $\mathsf{wf\_nat}(n)$ et $\mathsf{wf\_nat}(m)$. □

9.3 Graphes pointés dans $F\omega.3$

Comme dans le chapitre précédent, nous allons représenter les ensembles par des graphes pointés, c'est-à-dire par des triplets (X, A, a) formés d'un type X, d'une relation binaire $A : X{\to}X{\to}\mathsf{Prop}$ et d'un objet $a : X$. Nous représenterons les relations de bissimilarité et d'appartenance par des relations

$$\mathsf{EQV}(X, A, a, Y, B, b) \qquad \text{et} \qquad \mathsf{ELT}(X, A, a, Y, B, b)$$

très similaires à celles que nous avons introduites au chapitre précédent.

Avant de donner la définition des relations EQV et ELT, nous devons d'abord déterminer la sorte dans laquelle se situent les types de base des graphes pointés que nous allons manipuler. Ainsi que nous l'avons déjà évoqué au début de la section 9.2, nous ne pouvons pas nous contenter de travailler dans la sorte Type_1 puisque celle-ci ne contient aucun type prouvablement infini.[12] Pour cette raison nous ne considérerons que des graphes pointés dont le type de base est dans la sorte Type_2, et nous utiliserons les abréviations suivantes

$$\begin{aligned}
\lambda(X, A, a).\, M &\equiv \lambda X : \mathsf{Type}_2 \,.\, \lambda A : X{\to}X{\to}\mathsf{Prop} \,.\, \lambda a : X \,.\, M \\
\forall(X, A, a).\, \phi &\equiv \forall X : \mathsf{Type}_2 \,.\, \forall A : X{\to}X{\to}\mathsf{Prop} \,.\, \forall a : X \,.\, \phi \\
\exists(X, A, a).\, \phi &\equiv \exists X : \mathsf{Type}_2 \,.\, \exists A : X{\to}X{\to}\mathsf{Prop} \,.\, \exists a : X \,.\, \phi
\end{aligned}$$

dont on vérifiera aisément qu'elles sont bien définies dans $F\omega.3$ pourvu que le terme M (resp. le terme ϕ) est bien typé (resp. est une proposition bien formée) dans une signature où X, A et a sont des variables de types Type_2, $X{\to}X{\to}\mathsf{Prop}$ et X respectivement.

9.3.1 Bissimilarité et appartenance

Les relations de bissimilarité et d'appartenance sont définies de la même manière qu'au chapitre précédent, à cette différence près que la sorte Type du système U est remplacée par la sorte Type_2 du système $F\omega.3$.

$$\begin{aligned}
&\mathsf{EQV} \;:\; \Pi X : \mathsf{Type}_2 \,.\, (X{\to}X{\to}\mathsf{Prop}) \to X \to \Pi Y : \mathsf{Type}_2 \,.\, (Y{\to}Y{\to}\mathsf{Prop}) \to Y \to \mathsf{Prop} \;:= \\
&\quad \lambda(X, A, a) \,.\, \lambda(Y, B, b) \,.\quad \exists R : (X{\to}Y{\to}\mathsf{Prop}) \,. \\
&\qquad \big(\forall x, x' : X \,.\, \forall y : Y \,.\quad A(x', x) \;\Rightarrow\; R(x, y) \;\Rightarrow\; \exists y' : Y \,.\, R(x', y') \;\wedge\; B(y', y)\big) \quad \wedge \\
&\qquad \big(\forall y, y' : Y \,.\, \forall x : X \,.\quad B(y', y) \;\Rightarrow\; R(x, y) \;\Rightarrow\; \exists x' : X \,.\, R(x', y') \;\wedge\; A(x', x)\big) \quad \wedge \\
&\qquad R(a, b)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{ELT} \;:\; \Pi X : \mathsf{Type}_2 \,.\, (X{\to}X{\to}\mathsf{Prop}) \to X \to \Pi Y : \mathsf{Type}_2 \,.\, (Y{\to}Y{\to}\mathsf{Prop}) \to Y \to \mathsf{Prop} \;:= \\
&\quad \lambda(X, A, a) \,.\, \lambda(Y, B, b) \,.\quad \exists b' : Y \,.\quad B(b', b) \;\wedge\; (X, A, a) \approx (Y, B, b').
\end{aligned}$$

Dans la suite de ce chapitre, les relations $\mathsf{EQV}(X, A, a, Y, B, b)$ et $\mathsf{ELT}(X, A, a, Y, B, b)$ seront notées $(X, A, a) \approx (Y, B, b)$ et $(X, A, a) \in (Y, B, b)$ respectivement.

[12] Il est facile de vérifier que toute la construction qui suit peut être effectuée avec des graphes pointés dans Type_1, à cette différence près que l'axiome de l'infini n'est plus dérivable. On obtient alors dans $F\omega.3$ (et en réalité dans $F\omega.2$) un modèle de la théorie des ensembles sans axiome de l'infini, qui est équiconsistante avec l'arithmétique de Peano.

Les deux propositions qui suivent expriment que la relation de bissimilarité est une relation d'équivalence, et que la relation d'appartenance est compatible à gauche et à droite vis-à-vis de la relation de bissimilarité. La preuve de ces deux résultats s'effectue de la même manière qu'au chapitre précédent, simplement en remplaçant dans les termes de preuves la sorte Type du système U par la sorte Type_2 du système $F\omega.3$.

Proposition 9.3.1 (Équivalence) — *La relation de bissimilarité entre les graphes pointés est une relation d'équivalence :*

$$\forall (X,A,a)\,. \qquad (X,A,a) \approx (X,A,a)$$

$$\forall (X,A,a),(Y,B,b)\,. \quad (X,A,a) \approx (Y,B,b) \;\Rightarrow\; (Y,B,b) \approx (X,A,a)$$

$$\forall (X,A,a),(Y,B,b),(Z,C,c)\,. \quad (X,A,a) \approx (Y,B,b) \;\Rightarrow\; (Y,B,b) \approx (Z,C,c) \;\Rightarrow\; (X,A,a) \approx (Z,C,c)$$

Proposition 9.3.2 (Compatibilité) — *La relation d'appartenance sur les graphes pointés est compatible à gauche et à droite par rapport à la relation de bissimilarité :*

$$\forall (X,A,a),(Y,B,b),(Z,C,c).\quad (X,A,a) \approx (Y,B,b) \;\Rightarrow\; (Y,B,b) \in (Z,C,c) \;\Rightarrow\; (X,A,a) \in (Z,C,c)$$

$$\forall (X,A,a),(Y,B,b),(Z,C,c).\quad (X,A,a) \in (Y,B,b) \;\Rightarrow\; (Y,B,b) \approx (Z,C,c) \;\Rightarrow\; (X,A,a) \in (Z,C,c)$$

Rappelons que pour établir la compatibilité à droite de la relation d'appartenance, il est nécessaire de recourir au lemme suivant :

Lemme 9.3.3 — *Si (X,A,a) et (Y,B,b) sont des graphes pointés bissimilaires, et si $a' : X$ est un sommet du premier graphe tel que $A(a',a)$, alors il existe un sommet $b' : Y$ du second graphe tel que $B(b',b)$ et tel que les graphes pointés (X,A,a') et (Y,B,b') sont bissimilaires :*

$$\forall (X,A,a),(Y,B,b).\quad (X,A,a) \approx (Y,B,b) \;\Rightarrow\; \forall a' : X\,.\quad A(a',a) \;\Rightarrow\; \exists b' : Y\,.\quad B(b',b) \;\wedge\; (X,A,a') \approx (Y,B,b').$$

Nous terminerons ce paragraphe en énonçant la traduction de l'axiome d'extensionnalité en termes de graphes pointés dans le système $F\omega.3$:

Lemme 9.3.4 (Extensionnalité) — *Si deux graphes pointés ont les mêmes éléments au sens de la relation* ELT*, alors ces graphes pointés sont bissimilaires :*

$$\forall (X,A,a).\ \forall (Y,B,b)\quad \big(\forall (Z,C,c).\quad (Z,C,c) \in (X,A,a) \;\Leftrightarrow\; (Z,C,c) \in (Y,B,b)\big) \;\Rightarrow\; (X,A,a) \approx (Y,B,b).$$

Preuve. Comme au paragraphe 8.4.2, cette proposition se démontre en remarquant que l'hypothèse selon laquelle (X,A,a) et (Y,B,b) ont les mêmes éléments entraîne que la relation $R : X \to Y \to \mathsf{Prop}$ définie par

$$R \;\equiv\; \lambda x : X\,.\,\lambda y : Y\,.\quad (X,A,x) \approx (Y,B,y) \;\vee\; (x =_X a \;\wedge\; y =_Y b)$$

est une bissimulation de (X,A,a) dans (Y,B,b). □

9.3.2 Quelques structures de données

La réalisation des axiomes de la théorie des ensembles de Zermelo dans le système $F\omega.3$ passe par la construction de divers graphes pointés représentant, par exemple, la paire formée par deux ensembles, la réunion des éléments d'un ensemble, l'ensemble des parties d'un ensemble donné, etc. Pour définir les types de base de chacun de ces graphes pointés, nous nous servirons de deux constructeurs de types $\mathsf{opt}(X)$ (lifting) et $\mathsf{sum}(X,Y)$ (somme liftée) correspondant aux définitions inductives suivantes

$$\begin{array}{l}\textbf{Inductive}\quad \mathsf{opt}\ [X:\mathsf{Type}_2]\ :\ \mathsf{Type}_2\ :=\\ \quad|\ \ \mathsf{some}\ :\ X\ \to\ \mathsf{opt}(X)\\ \quad|\ \ \mathsf{none}\ :\ \mathsf{opt}(X)\end{array}$$

$$\begin{array}{l}\textbf{Inductive}\quad \mathsf{sum}\ [X,Y:\mathsf{Type}_2]\ :\ \mathsf{Type}_2\ :=\\ \quad|\ \ \mathsf{inl}\ :\ X\ \to\ \mathsf{sum}(X,Y)\\ \quad|\ \ \mathsf{inr}\ :\ Y\ \to\ \mathsf{sum}(X,Y)\\ \quad|\ \ \mathsf{out}\ :\ \mathsf{sum}(X,Y)\end{array}$$

Bien entendu, les deux définitions ci-dessus ne font du sens que dans une théorie des types disposant de types inductifs primitifs. Dans le cadre du système $F\omega.3$, nous devrons donc nous contenter de codages approximatifs permettant de définir les constructeurs some, none, inl, inr et out sans toutefois bénéficier des schémas d'élimination correspondants.

Le constructeur de type opt Le constructeur de type $\mathsf{opt} : \mathsf{Type}_2 \to \mathsf{Type}_2$ qui à chaque type $X : \mathsf{Type}_2$ associe le type lifté $\mathsf{opt}(X) : \mathsf{Type}_2$ est défini par

$$\mathsf{opt}\quad :=\quad \lambda X:\mathsf{Type}_2\,.\,(X{\to}\mathsf{Prop})\to\mathsf{Prop},$$

et les constructeurs some et none qui lui sont associés sont donnés par

$$\begin{array}{lcll}\mathsf{some} & : & \Pi X:\mathsf{Type}_2\,.\,X\to\mathsf{opt}(X) & := \quad \lambda X:\mathsf{Type}_2\,.\,\lambda x:X\,.\,\lambda f:(X{\to}\mathsf{Prop})\,.\,f(x)\\ \mathsf{none} & : & \Pi X:\mathsf{Type}_2\,.\,\mathsf{opt}(X) & := \quad \lambda X:\mathsf{Type}_2\,.\,\lambda f:(X{\to}\mathsf{Prop})\,.\,\bot\end{array}$$

Cette définition nous permet de montrer que les fonctions some et none sont des constructeurs, c'est-à-dire des fonctions injectives ayant des images deux-à-deux disjointes :

Lemme 9.3.5 (Injectivité du constructeur some**)** — *Pour tout type* $X : \mathsf{Type}_2$*, le constructeur* $\mathsf{some}(X) : X \to \mathsf{opt}(X)$ *est une fonction injective :*

$$\forall X:\mathsf{Type}_2\,.\quad \forall x_1,x_2:X\,.\quad \mathsf{some}(X,x_1)=_{\mathsf{opt}(X)}\mathsf{some}(X,x_2)\quad\Rightarrow\quad x_1=_X x_2.$$

Preuve. Considérons une preuve ξ de l'égalité $\mathsf{some}(X,x_1)=_{\mathsf{opt}(X)}\mathsf{some}(X,x_2)$ ainsi que le prédicat $P : \mathsf{opt}(X) \to \mathsf{Prop}$ défini par

$$P\quad\equiv\quad \lambda z:\mathsf{opt}(X)\,.\ z\ (\lambda x:X\,.\,x_1=_X x).$$

En remarquant que pour tout x de type X, les propositions $P(\mathsf{some}(X,x))$ et $x_1 =_X x$ sont convertibles, on vérifie immédiatement que si p_1 désigne une preuve de l'égalité $x_1 =_X x_1$ (réflexivité), alors le terme de preuve $\xi\ P\ p_1$ est une preuve de l'égalité $x_1 =_X x_2$. □

Lemme 9.3.6 (Discrimination) — *Pour tout type* $X : \mathsf{Type}_2$ *et pour tout terme* x *de type* X*, les termes* $\mathsf{some}(X,x)$ *et* $\mathsf{none}(X)$ *de type* $\mathsf{opt}(X)$ *sont différents :*

$$\forall X : \mathsf{Type}_2\,.\quad \forall x : X\,.\quad \neg\ \mathsf{some}(X,x) =_{\mathsf{opt}(X)} \mathsf{none}(X).$$

Preuve. Soient ξ une preuve de l'égalité $\mathsf{some}(X,x) =_{\mathsf{opt}(X)} \mathsf{none}(X)$ et $P : \mathsf{opt}(X) \to \mathsf{Prop}$ le prédicat défini par

$$P \quad \equiv \quad \lambda z : \mathsf{opt}(X)\,.\ z\ (\lambda\_ : X\,.\,\top).$$

En remarquant que les propositions $P(\mathsf{some}(X,x))$ et $\top$ sont convertibles, de même que les propositions $P(\mathsf{none}(X))$ et $\bot$, on vérifie aisément que $\xi\ P\ p$ est une preuve de la proposition absurde $\bot$, où p désigne une preuve de la proposition $\top$. □

Remarque 9.3.7 — Le schéma d'élimination correspondant à la définition inductive mentionnée précédemment n'est pas dérivable, puisque le type $\mathsf{opt}(X)$ contient des objets qui ne sont ni de la forme $\mathsf{some}(X,x)$ (avec $x : X$) ni de la forme $\mathsf{none}(X)$, comme par exemple le terme $\lambda f : (X \to \mathsf{Prop})\,.\,\top$ de type $\mathsf{opt}(X)$. La présence de tels objets "parasites" ne posera aucun problème en pratique, car les graphes pointés que nous construirons sur les types de la forme $\mathsf{opt}(X)$ ne feront référence qu'aux objets réguliers de type $\mathsf{opt}(X)$, c'est-à-dire les objets définis uniquement à partir des constructeurs some et none. Par conséquent, les objets non-réguliers de type $\mathsf{opt}(X)$ ne seront pas accessibles à partir de la racine et seront de ce fait invisibles vis-à-vis de la relation de bissimilarité (*cf* Fig. 8.2 p. 263).

Le constructeur de type sum Le constructeur de type $\mathsf{sum} : \mathsf{Type}_2 \to \mathsf{Type}_2 \to \mathsf{Type}_2$ qui à chaque couple de types $X, Y : \mathsf{Type}_2$ associe la somme lifté $\mathsf{sum}(X,Y) : \mathsf{Type}_2$ est défini par

$$\mathsf{sum} \quad := \quad \lambda X, Y : \mathsf{Type}_2\,.\,(X \to \mathsf{Prop}) \to (Y \to \mathsf{Prop}) \to \mathsf{Prop},$$

et les constructeurs inl, inr et out qui lui sont associés sont définis par :

$$\begin{array}{lcl}
\mathsf{inl} & : & \Pi X, Y : \mathsf{Type}_2\,.\,X \to \mathsf{sum}(X,Y) \\
& := & \lambda X, Y : \mathsf{Type}_2\,.\,\lambda x : X\,.\,\lambda f : (X \to \mathsf{Prop})\,.\,\lambda g : (Y \to \mathsf{Prop})\,.\,f(x) \\
\\
\mathsf{inr} & : & \Pi X, Y : \mathsf{Type}_2\,.\,Y \to \mathsf{sum}(X,Y) \\
& := & \lambda X, Y : \mathsf{Type}_2\,.\,\lambda y : Y\,.\,\lambda f : (X \to \mathsf{Prop})\,.\,\lambda g : (Y \to \mathsf{Prop})\,.\,g(y) \\
\\
\mathsf{out} & : & \Pi X, Y : \mathsf{Type}_2\,.\,\mathsf{sum}(X,Y) \\
& := & \lambda X, Y : \mathsf{Type}_2\,.\,\lambda f : (X \to \mathsf{Prop})\,.\,\lambda g : (Y \to \mathsf{Prop})\,.\,\bot
\end{array}$$

Les deux lemmes suivants (injectivité et discrimination) expriment le fait que les termes inl, inr et out sont des constructeurs de la somme liftée $\mathsf{sum}(X,Y)$. Ces deux résultats se prouvent de manière analogue aux lemmes 9.3.5 et 9.3.6.

Lemme 9.3.8 (Injectivité des constructeurs inl **et** inr**)** — *Pour tous types* $X, Y : \mathsf{Type}_2$*, les constructeurs* $\mathsf{inl}(X,Y) : X \to \mathsf{sum}(X,Y)$ *et* $\mathsf{inr}(X,Y) : Y \to \mathsf{sum}(X,Y)$ *sont des fonctions injectives :*

$$\forall X, Y : \mathsf{Type}_2\,.\quad \forall x_1, x_2 : X\,.\quad \mathsf{inl}(X,Y,x_1) =_{\mathsf{sum}(X,Y)} \mathsf{inl}(X,Y,x_2) \quad \Rightarrow \quad x_1 =_X x_2$$

$$\forall X, Y : \mathsf{Type}_2\,.\quad \forall y_1, y_2 : Y\,.\quad \mathsf{inr}(X,Y,y_1) =_{\mathsf{sum}(X,Y)} \mathsf{inr}(X,Y,y_2) \quad \Rightarrow \quad y_1 =_Y y_2$$

Lemme 9.3.9 (Discrimination) — *Pour tous types $X, Y : \mathsf{Type}_2$, les images des constructeurs $\mathsf{inl}(X,Y)$, $\mathsf{inr}(X,Y)$ et $\mathsf{out}(X,Y)$ sont deux à deux disjointes :*

$$\begin{array}{lll}
\forall X,Y:\mathsf{Type}_2\,. & \forall x:X\,. & \neg\ \mathsf{inl}(X,Y,x) =_{\mathsf{sum}(X,Y)} \mathsf{out}(X,Y) \\
\forall X,Y:\mathsf{Type}_2\,. & \forall y:Y\,. & \neg\ \mathsf{inr}(X,Y,y) =_{\mathsf{sum}(X,Y)} \mathsf{out}(X,Y) \\
\forall X,Y:\mathsf{Type}_2\,. & \forall x:X\,.\,\forall y:Y\,. & \neg\ \mathsf{inl}(X,Y,x) =_{\mathsf{sum}(X,Y)} \mathsf{inr}(X,Y,y)
\end{array}$$

Remarque 9.3.10 — Là encore, le schéma d'élimination correspondant n'est pas dérivable, puisque le type $\mathsf{sum}(X,Y)$ contient des objets qui ne sont ni de la forme $\mathsf{inl}(X,Y,x)$ (avec $x : X$), ni de la forme $\mathsf{inr}(X,Y,y)$ (avec $y : Y$), ni de la forme $\mathsf{out}(X,Y)$, tels que le terme $\lambda f : (X \to \mathsf{Prop})\,.\,\lambda g : (Y \to \mathsf{Prop})\,.\,\top$ par exemple. La remarque 9.3.7 concernant l'invisiblité (en pratique) des objets non-réguliers de type $\mathsf{opt}(X)$ s'applique également aux objets non-réguliers de type $\mathsf{sum}(X,Y)$.

Plongements et bissimilarité Dans la suite de ce chapitre, nous utiliserons les constructeurs de type $\mathsf{opt}(X)$ et $\mathsf{sum}(X,Y)$ pour définir des graphes pointés de la forme $(\mathsf{opt}(X), R, r)$ ou $(\mathsf{sum}(X,Y), R, r)$ à partir de graphes pointés (X, A, a) et (Y, B, b) donnés par hypothèse. Au cours de ce type de construction, nous serons fréquemment amenés à recopier la structure de graphe de (X, A) et/ou de (Y, B) dans le nouveau graphe basé sur le type $\mathsf{opt}(X)$ ou bien sur le type $\mathsf{sum}(X,Y)$ au moyen des injections $\mathsf{some}(X)$, $\mathsf{inl}(X,Y)$ et/ou $\mathsf{inr}(X,Y)$.

Pour que ce processus de "délocalisation" ait un sens, il faudra encore nous assurer que la relation de bissimilarité est conservée par ces injections, ce que nous ferons en établissant pour chaque construction un *lemme de délocalisation* dont l'énoncé sera de la forme

$$\begin{array}{llcll}
\forall x:X\,. & (X,A,x) & \approx & \big(\mathsf{opt}(X),\ R,\ \mathsf{some}(X,x)\big), & \text{ou bien} \\
\forall x:X\,. & (X,A,x) & \approx & \big(\mathsf{sum}(X,Y),\ R,\ \mathsf{inl}(X,Y,x)\big), & \text{et/ou} \\
\forall y:Y\,. & (Y,B,y) & \approx & \big(\mathsf{sum}(X,Y),\ R,\ \mathsf{inr}(X,Y,y)\big). &
\end{array}$$

(Bien entendu, le choix de la formule ou des formules à satisfaire parmi les trois formules écrites ci-dessus dépendra de l'axiome traité et de la définition de la relation R.) Afin de simplifier la preuve de ces lemmes de délocalisation, nous utiliserons la notion de *plongement de graphe* suivante :

Définition 9.3.11 (Plongement de graphes) — Soient (X, A) et (Y, B) deux graphes. On appelle *plongement* de (X, A) dans (Y, B) toute fonction $f : X \to Y$ satisfaisant les deux conditions suivantes :

1. pour tous $x, x' : X$, $A(x', x)$ entraîne $B(f(x'), f(x))$
2. pour tout $x : X$ et pour tout $y' : Y$ tel que $B(y', f(x))$, il existe un objet $x' : X$ tel que $f(x') =_Y y$ et $A(x', x)$.

Intuitivement, un plongement de graphes est un morphisme de graphes $f : (X, A) \to (Y, B)$ tel que dans le graphe cible, tout arc de la forme $B(y', f(x))$ peut être relevé dans le graphe source en un arc $A(x', x)$, avec $y' =_Y f(x')$. Dans le système $F\omega.3$, la notion de plongement

de graphes est représentée par le terme suivant :

$$\begin{array}{rcl}\mathsf{embed} & : & \Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to \\ & & \quad \Pi Y : \mathsf{Type}_2 \,.\, (Y \to Y \to \mathsf{Prop}) \to (X \to Y) \to \mathsf{Prop} \quad :=\end{array}$$

$$\begin{array}{l}\lambda X : \mathsf{Type}_2 \,.\, \lambda A : (X \to X \to \mathsf{Prop}) \,.\, \lambda Y : \mathsf{Type}_2 \,.\, \lambda B : (Y \to Y \to \mathsf{Prop}) \,.\, \lambda f : (X \to Y) \,. \\ \qquad \big(\forall x, x' : X \,. \quad A(x', x) \quad \Rightarrow \quad B(f(x'), f(x))\big) \\ \quad \wedge \quad \big(\forall x : X \,. \forall y' : Y \,. \quad B(y', f(x)) \quad \Rightarrow \quad \exists x' : X \,. \quad f(x') = y \quad \wedge \quad A(x', x)\big)\end{array}$$

L'intérêt des plongements de graphes est de fournir automatiquement une famille de bissimulations entre les graphes pointés (X, A, x) construits sur le graphe source (X, A) et leurs images $(Y, B, f(x))$ dans le graphe cible (Y, B) par le plongement f :

Lemme 9.3.12 (Bissimulation par plongement) — *Soient (X, A) et (Y, B) deux graphes basés dans la sorte Type_2. Si $f : X \to Y$ est un plongement de (X, A) dans (Y, B), alors pour tout objet $x : X$, les graphes pointés (X, A, x) et $(Y, B, f(x))$ sont bissimilaires :*

$$\begin{array}{l}\forall X : \mathsf{Type}_2 \,. \forall A : (X \to X \to \mathsf{Prop}) \,. \forall Y : \mathsf{Type}_2 \,. \forall B : (Y \to Y \to \mathsf{Prop}) \,. \\ \quad \forall f : (X \to Y) \,. \quad \mathsf{embed}(X, A, Y, B, f) \quad \Rightarrow \quad \forall x : X \,. \quad (X, A, x) \approx (Y, B, f(x))\end{array}$$

Preuve. Il suffit de vérifier que pour tout plongement de graphes $f : (X, A) \to (Y, B)$, la relation binaire $R : X \to Y \to \mathsf{Prop}$ définie par

$$R \quad \equiv \quad \lambda x : X \,. \lambda y : Y \,. \quad y =_X f(x)$$

est une bissimulation du graphe pointé (X, A, x) dans le graphe pointé $(Y, B, f(x))$, et cela quel que soit le choix du sommet $x : X$. □

Dans la suite de ce chapitre, nous utiliserons exclusivement le lemme ci-dessus dans le cas où le plongement f est une injection de la forme $\mathsf{some}(X)$, $\mathsf{inl}(X, Y)$ ou $\mathsf{inr}(X, Y)$.

9.4 La traduction des axiomes de Zermelo

Dans cette section, nous nous proposons de montrer que tous les ensembles définissables dans la théorie des ensembles de Zermelo intuitionniste peuvent être représentés dans le système $F\omega.3$ par des graphes pointés satisfaisant les mêmes propriétés, en traduisant les relations d'égalité et d'appartenance de la théorie des ensembles par les relations $(X, A, a) \approx (Y, B, b)$ et $(X, A, a) \in (Y, B, b)$ respectivement.

Comme les règles de déduction du calcul des prédicats (du premier ordre) en logique intuitionniste sont dérivables dans le système $F\omega.3$, il nous suffit de montrer que la traduction de chacun des axiomes de la théorie des ensembles de Zermelo (*cf* Fig. 9.1) est prouvable dans $F\omega.3$. Nous ne traiterons ici que les axiomes "existentiels" (*i.e.* l'axiome de la paire, les axiomes de compréhension, l'axiome des parties, l'axiome de la réunion et l'axiome de l'infini) puisque les axiomes d'égalité, les axiomes de compatibilité et l'axiome d'extensionnalité ont déjà été traités dans la section précédente.

Par ailleurs, le cadre intuitionniste dans lequel nous nous plaçons nous amènera naturellement à prouver les (traductions des) axiomes de la théorie des ensembles de Zermelo sous

Axiomes d'égalité

$$\forall x \qquad x = x$$
$$\forall x, y \qquad x = y \quad \Rightarrow \quad y = x$$
$$\forall x, y, z \qquad x = y \quad \Rightarrow \quad y = z \quad \Rightarrow \quad x = z$$

Axiomes de compatibilité

$$\forall x, y, z \qquad x = y \quad \Rightarrow \quad y \in z \quad \Rightarrow \quad x \in z$$
$$\forall x, y, z \qquad x \in y \quad \Rightarrow \quad y = z \quad \Rightarrow \quad x \in z$$

Axiome d'extensionnalité

$$\forall a, b \qquad (\forall x \quad x \in a \quad \Leftrightarrow \quad x \in b) \quad \Rightarrow \quad a = b$$

Axiome de la paire

$$\forall a, b \qquad \exists c \quad \forall x \qquad x \in c \quad \Leftrightarrow \quad x = a \ \vee \ x = b$$

Schéma d'axiomes de compréhension

$$\forall x_1, \ldots, x_n, a \qquad \exists b \quad \forall x \qquad x \in b \quad \Leftrightarrow \quad x \in a \ \wedge \ \phi_{x_1,\ldots,x_n,x}$$

pour chaque formule $\phi_{x_1,\ldots,x_n,x}$ dont les variables libres figurent parmi les variables $x_1, \ldots, x_n, x$.

Axiome des parties

$$\forall a \qquad \exists b \quad \forall x \qquad x \in b \quad \Leftrightarrow \quad x \subset a$$

où $x \subset y$ est une abréviation pour $\forall z \ \ z \in x \ \Rightarrow \ z \in y$.

Axiome de la réunion

$$\forall a \qquad \exists b \quad \forall x \qquad x \in b \quad \Leftrightarrow \quad (\exists y \ \ y \in a \ \wedge \ x \in y)$$

Axiome de l'infini

$$\exists a \qquad \varnothing \in a \quad \wedge \quad \big(\forall x \quad x \in a \quad \Rightarrow \quad s(x) \in a\big)$$

où $\varnothing$ désigne l'ensemble vide dont l'existence découle d'une instance du schéma de compréhension, et où $s(x)$ désigne l'ensemble $x \cup \{x\}$ dont l'existence est une conséquence de l'axiome de la paire et de l'axiome de la réunion.

FIG. 9.1 – Les axiomes de la théorie des ensembles de Zermelo

une forme skolémisée. En effet, chaque preuve d'existence fera l'objet d'une construction explicite dans laquelle nous introduirons trois fonctions (paramétrées par les trois composantes des graphes pointés donnés par les hypothèses) retournant respectivement le type, la relation binaire et la racine du graphe pointé qui satisfait dans le système $F\omega.3$ les (traductions des) propriétés donnés par l'axiome correpondant.

Comme en pratique, les types de base des graphes pointés que nous allons construire seront tous de la forme $\mathsf{opt}(X)$ ou $\mathsf{sum}(X,Y)$, et comme leurs racines seront toutes de la forme $\mathsf{none}(X)$ ou $\mathsf{out}(X,Y)$, nous n'aurons en réalité qu'à introduire (pour chaque axiome) la fonction construisant la relation binaire du graphe pointé final à partir de la structure des graphes pointés donnés par les hypothèses.

9.4.1 L'axiome de la paire

Soient (A,X,a) et (Y,B,b) deux graphes pointés. La paire (non ordonnée) formée par ces deux graphes pointés est représentée par le graphe pointé (T,R,r) dont les composantes sont données par

- $T \;\equiv\; \mathsf{sum}(X,Y)$;
- R est la relation binaire sur $\mathsf{sum}(X,Y)$ engendrée par les clauses suivantes :
  1. si $A(x',x)$, alors $R\big(\mathsf{inl}(X,Y,x'),\mathsf{inl}(X,Y,x)\big)$;
  2. si $B(y',y)$, alors $R\big(\mathsf{inr}(X,Y,y'),\mathsf{inr}(X,Y,y)\big)$;
  3. $R\big(\mathsf{inl}(X,Y,a),\mathsf{out}(X,Y)\big)$;
  4. $R\big(\mathsf{inr}(X,Y,b),\mathsf{out}(X,Y)\big)$.
- $r \;\equiv\; \mathsf{out}(X,Y)$.

Formellement, la relation R est donnée par

$$R \quad\equiv\quad \mathsf{PAIR}(X,A,a,Y,B,b),$$

où PAIR désigne le constructeur de relation défini dans le système $F\omega.3$ par

$$\begin{array}{rl}
\mathsf{PAIR} \;:\; & \Pi X:\mathsf{Type}_2\,.\,(X{\to}X{\to}\mathsf{Prop}) \to X \to \\
& \quad \Pi Y:\mathsf{Type}_2\,.\,(Y{\to}Y{\to}\mathsf{Prop}) \to Y \to \\
& \qquad \mathsf{sum}(X,Y) \to \mathsf{sum}(X,Y) \to \mathsf{Prop} \quad :=
\end{array}$$

$$\begin{array}{lll}
\lambda(X,A,a),(Y,B,b)\,. & \lambda z',z:\mathsf{sum}(X,Y)\,. & \\
\quad \big(\exists x',x:X\,. & z' = \mathsf{inl}(X,Y,x') \;\wedge\; z = \mathsf{inl}(X,Y,x) \;\wedge\; A(x',x)\big) & \\
\vee \quad \big(\exists y',y:Y\,. & z' = \mathsf{inr}(X,Y,y') \;\wedge\; z = \mathsf{inr}(X,Y,y) \;\wedge\; B(y',y)\big) & \\
\vee \quad \big(z' = \mathsf{inl}(X,Y,a) & \wedge\; z = \mathsf{out}(X,Y)\big) & \\
\vee \quad \big(z' = \mathsf{inr}(X,Y,b) & \wedge\; z = \mathsf{out}(X,Y)\big). &
\end{array}$$

Intuitivement, le graphe pointé (T,R,r) est obtenu en recollant les deux graphes pointés (X,A,a) et (Y,B,b) (clauses 1 et 2) et en connectant les anciennes racines a et b à la nouvelle racine $r \equiv \mathsf{out}(X,Y)$ (clauses 3 et 4). D'après la définition de la relation R, il est clair que les injections

$$\mathsf{inl}(X,Y) : X \to \mathsf{sum}(X,Y) \qquad \text{et} \qquad \mathsf{inr}(X,Y) : Y \to \mathsf{sum}(X,Y)$$

plongent respectivement les graphes (X, A) et (Y, B) dans le graphe (T, R) au sens de la définition 9.3.11, d'où il ressort d'après le lemme 9.3.12 que :

Lemme 9.4.1 (Délocalisation) — *Pour tous sommets $x : X$ et $y : Y$, on a les équivalences de bissimulation*

$$(X, A, x) \;\approx\; \big(T, R, \mathsf{inl}(X, Y, x)\big) \qquad \textit{et} \qquad (Y, B, y) \;\approx\; \big(T, R, \mathsf{inr}(X, Y, y)\big).$$

Lemme 9.4.2 (Introduction) — *Les graphes pointés (X, A, a) et (Y, B, b) sont des éléments du graphe pointé (T, R, r) :*

$$(X, A, a) \in (T, R, r) \qquad \textit{et} \qquad (Y, B, b) \in (T, R, r).$$

Preuve. La relation $(X, A, a) \in (T, R, r)$ découle immédiatement des relations $(X, A, a) \approx \big(T, R, \mathsf{inl}(X, Y, a)\big)$ (lemme de délocalisation) et $R(\mathsf{inl}(X, Y, a), r)$ (clause 3). On raisonne de la même manière pour montrer que $(Y, B, b) \in (T, R, r)$. □

Lemme 9.4.3 (Élimination) — *Si (Z, C, c) est un élément du graphe pointé (T, R, r), alors (Z, C, c) est bissimilaire à (X, A, a) ou à (Y, B, b) :*

$$\forall (Z, C, c)\,.\quad (Z, C, c) \in (T, R, r) \quad\Rightarrow\quad (Z, C, c) \approx (X, A, a) \;\vee\; (Z, C, c) \approx (Y, B, b).$$

Preuve. Si (Z, C, c) est un graphe pointé tel que $(Z, C, c) \in (T, R, r)$, il existe un sommet $r' : T$ tel que $(Z, C, c) \approx (T, R, r')$ et $R(r', r)$. Comme $r \equiv \mathsf{out}(X, Y)$, l'assertion $R(r', r)$ ne peut provenir que d'une des clauses 3 où 4 définissant R. S'il s'agit de la clause 3, on a alors $r' \equiv \mathsf{inl}(X, Y, a)$ d'où il ressort que $(Z, C, c) \approx \big(T, R, \mathsf{inl}(X, Y, a)\big) \approx (X, A, a)$. S'il s'agit de la clause 4, on montre de la même façon que $(Z, C, c) \approx (Y, B, b)$. □

Ces deux lemmes nous permettent de conclure que la traduction de l'axiome de la paire est prouvable dans $F\omega.3$:

Proposition 9.4.4 (Axiome de la paire) — *Quels que soient les graphes pointés (X, A, a) et (Y, B, b), il existe un graphe pointé (T, R, r) dont les seuls éléments sont (à une bissimulation près) les graphes pointés (X, A, a) et (Y, B, b) :*

$$\begin{array}{l}\forall (X, A, a), (Y, B, b)\,.\quad \exists (T, R, r)\,. \\ \quad \forall (Z, C, c)\,.\quad (Z, C, c) \in (T, R, r) \quad\Leftrightarrow\quad (Z, C, c) \approx (X, A, a) \;\vee\; (Z, C, c) \approx (Y, B, b).\end{array}$$

9.4.2 Les axiomes de compréhension

Prédicats compatibles Considérons un prédicat

$$P \quad : \quad \Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}$$

défini sur la classe des graphes pointés. On dit que le prédicat P est *compatible* (vis-à-vis de la relation de bissimilarité) si pour tous graphes pointés (X, A, a) et (Y, B, b), les assertions $P(X, A, a)$ et $(X, A, a) \approx (Y, B, b)$ entraînent $P(Y, B, b)$. Dans le système $F\omega.3$, cette assertion est représenté par la proposition $\mathsf{COMPAT}(P)$, où COMPAT est la fonction définie par :

$$\begin{array}{l}\mathsf{COMPAT} \quad : \quad \big(\Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \to \mathsf{Prop} \quad := \\ \quad \lambda P : \big(\Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big)\,.\quad \forall (X, A, a), (Y, B, b)\,. \\ \qquad P(X, A, a) \quad\Rightarrow\quad (X, A, a) \approx (Y, B, b) \quad\Rightarrow\quad P(Y, B, b).\end{array}$$

Il est facile de vérifier que tout prédicat formé à partir des relations binaires $(X, A, a) \approx (Y, B, b)$ et $(X, A, a) \in (Y, B, b)$ à l'aide des connecteurs logiques $\bot$, $\Rightarrow$, $\wedge$, $\vee$ et des quantificateurs $\forall(X, A, a)\,.\,\phi$ et $\exists(Y, A, a)\,.\,\phi$ est un prédicat compatible (nous préciserons ce point au paragraphe 9.5.1).

Le schéma d'axiomes de compréhension Soient (X, A, a) un graphe pointé et

$$P \quad : \quad \Pi X : \mathsf{Type}_2\,.\,(X{\to}X{\to}\mathsf{Prop}) \to X \to \mathsf{Prop}$$

un prédicat défini sur la classe des graphes pointés. L'ensemble des graphes pointés appartenant à (X, A, a) et satisfaisant la propriété P est représenté par le graphe pointé (T, R, r) dont les composantes sont données par

- $T \;\equiv\; \mathsf{opt}(X)$;
- $R : \mathsf{opt}(X) \to \mathsf{opt}(X) \to \mathsf{Prop}$ est la relation engendrée par les clauses suivantes
  1. si $A(x', x)$, alors $R\big(\mathsf{some}(X, x'), \mathsf{some}(X, x)\big)$;
  2. si $A(x', a)$ et $P(X, A, x')$, alors $R\big(\mathsf{some}(X, x'), \mathsf{none}(X)\big)$.
- $r \;\equiv\; \mathsf{none}(X)$.

Formellement, la relation R est définie par

$$R \quad \equiv \quad \mathsf{FOLD}(X, A, a, P),$$

où FOLD désigne le constructeur de relation défini dans le système $F\omega.3$ par

$$\begin{array}{l}
\mathsf{FOLD} \;:\; \Pi X : \mathsf{Type}_2\,.\,(X{\to}X{\to}\mathsf{Prop}) \to X \to \\
\qquad \big(\Pi Y : \mathsf{Type}_2\,.\,(Y{\to}Y{\to}\mathsf{Prop}){\to}Y{\to}\mathsf{Prop}\big) \to \\
\qquad\quad \mathsf{opt}(X) \to \mathsf{opt}(X) \to \mathsf{Prop} \quad := \\
\\
\lambda(X, A, a)\,.\quad \lambda P : \big(\Pi Y : \mathsf{Type}_2\,.\,(Y{\to}Y{\to}\mathsf{Prop}){\to}Y{\to}\mathsf{Prop}\big)\,.\quad \lambda z', z : \mathsf{opt}(X)\,. \\
\qquad \big(\exists x', x : X\,.\quad z' = \mathsf{some}(X, x') \quad\wedge\quad z = \mathsf{some}(X, x) \quad\wedge\quad A(x', x)\big) \\
\vee \quad \big(\exists x' : X\,.\qquad z' = \mathsf{some}(X, x') \quad\wedge\quad z = \mathsf{none}(X) \qquad\wedge\quad A(x', a) \quad\wedge\quad P(X, A, x')\big).
\end{array}$$

Intuitivement, le graphe pointé (T, R, r) est obtenu en délocalisant la structure du graphe pointé (X, A, a) dans le type $\mathsf{opt}(X)$ (clause 1), et en ne connectant à la nouvelle racine $r \equiv \mathsf{none}(X)$ que les (images des) sommets $x' : A$ connectés à l'ancienne racine $a : X$ par la relation R et tels que le graphe pointé (X, A, x') satisfasse la propriété P (clause 2). Il est par ailleurs clair que l'injection

$$\mathsf{some}(X) \quad : \quad X \to \mathsf{opt}(X)$$

constitue un plongement du graphe (X, A) dans le graphe (T, R), d'où il ressort (d'après le lemme 9.3.12) que :

Lemme 9.4.5 (Délocalisation) — *Pour tout sommet $x : X$ on l'équivalence de bissimulation suivante :*

$$(X, A, x) \;\approx\; \big(T, R, \mathsf{some}(X, x)\big).$$

Lemme 9.4.6 (Introduction) — *Si P est un prédicat compatible, alors pour tout graphe pointé (Y, B, b) on a*

$$(Y, B, b) \in (X, A, a) \quad \Rightarrow \quad P(Y, B, b) \quad \Rightarrow \quad (Y, B, b) \in (T, R, r).$$

Preuve. Supposons que P est compatible, et considérons un graphe pointé (Y, B, b) tel que $(Y, B, b) \in (X, A, a)$ et $P(Y, B, b)$. Par définition de la relation d'appartenance, il existe un sommet $x' : X$ tel que $(Y, B, b) \approx (X, A, x')$ et $A(x', a)$. Comme P est compatible, on a $P(X, A, x')$, d'où $R(\mathsf{some}(X, x'), \mathsf{none}(X))$ (clause 2). Mais comme les graphes pointés (X, A, x') et $(T, R, \mathsf{some}(X, x'))$ sont bissimilaires, on a $(Y, B, b) \approx (T, R, \mathsf{some}(X, x'))$, d'où $(Y, B, b) \in (T, R, r)$ puisque $R(\mathsf{some}(X, x'), r)$. □

Lemme 9.4.7 (Élimination) — *Si P est un prédicat compatible, alors pour tout graphe pointé (Y, B, b) on a*

$$(Y, B, b) \in (T, R, r) \quad \Rightarrow \quad (Y, B, b) \in (X, A, a) \quad \wedge \quad P(Y, B, b).$$

Preuve. Soit (Y, B, b) un graphe pointé tel que $(Y, B, b) \in (T, R, r)$. Il existe un sommet $r' : \mathsf{opt}(X)$ tel que $(Y, B, b) \approx (T, R, r')$ et $R(r', r)$. Comme $r \equiv \mathsf{none}(X)$, l'assertion $R(r', r)$ ne peut venir que de la 2ème clause définissant R, d'où il existe un sommet $x' : X$ tel que $r' = \mathsf{some}(X, x')$, $A(x', a)$ et $P(X, A, x')$. Comme les graphes pointés (T, R, r') et (X, A, x') sont bissimilaires (lemme de délocalisation), on a $(Y, B, b) \approx (X, A, x')$ d'où il découle que $(Y, B, b) \in (X, A, a)$ puisque $A(x', a)$. Comme par ailleurs P est un prédicat compatible tel que $P(X, A, x')$, on a également $P(Y, B, b)$. □

De ces deux lemmes découle la proposition suivante :

Proposition 9.4.8 (Axiomes de compréhension) — *Pour tout prédicat compatible*

$$P \quad : \quad \Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}$$

et pour tout graphe pointé (X, A, a), il existe un graphe pointé (T, R, r) dont les éléments sont exactement les graphes pointés appartenant à (X, A, a) qui satisfont le prédicat P :

$$\begin{array}{l} \forall P : \big(\Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \,. \quad \mathsf{COMPAT}(P) \quad \Rightarrow \\ \quad \forall (X, A, a) \,. \quad \exists (T, R, r) \,. \quad \forall (Y, B, b) \,. \\ \qquad (Y, B, b) \in (T, R, r) \quad \Leftrightarrow \quad (Y, B, b) \in (X, A, a) \quad \wedge \quad P(Y, B, b). \end{array}$$

Remarque 9.4.9 — Il est intéressant de noter que le résultat ci-dessus exprime à l'aide d'une seule proposition du système $F\omega.3$ la validité de la traduction d'un ensemble *infini* de formules de la théorie des ensembles. Ceci est dû au fait que dans le système $F\omega.3$ il est possible d'exprimer la quantification universelle sur tous les prédicats (définis sur la classe des graphes pointés) alors qu'en théorie des ensembles (qui est une théorie du premier ordre), les prédicats ne sont pas des objets de première classe.

Cette constatation montre déjà que le système $F\omega.3$ est en réalité *strictement plus fort* (d'un point de vue logique) que la théorie des ensembles de Zermelo intuitionniste (et même classique ainsi que nous le verrons dans la section 9.6). En effet, ce système permet non seulement d'exprimer la quantification du second ordre en théorie des ensembles, mais il permet également d'utiliser cette quantification au sein de la formule $P(X, A, a)$ sur laquelle porte le schéma de compréhension (ce qui est essentiel pour avoir l'inégalité stricte).

9.4.3 L'axiome des parties

La relation d'inclusion En théorie des ensembles, la relation d'inclusion $x \subset y$ n'est pas une relation primitive, mais une abréviation définie par

$$x \subset y \quad \equiv \quad (\forall z \quad z \in x \;\Rightarrow\; z \in y).$$

Dans le système $F\omega.3$, cette relation se traduit naturellement en termes de graphes pointés par la relation binaire $\mathsf{SUB}(X, A, a, Y, B, b)$ définie par

$$\begin{array}{l}\mathsf{SUB} \;:\; \Pi X : \mathsf{Type}_2 \,.\, (X{\to}X{\to}\mathsf{Prop}) \to X \to \Pi Y : \mathsf{Type}_2 \,.\, (Y{\to}Y{\to}\mathsf{Prop}) \to Y \to \mathsf{Prop} \quad := \\ \quad \lambda(X, A, a), (Y, B, b) \,.\quad \forall (Z, C, c) \,.\quad (Z, C, c) \in (X, A, a) \quad \Rightarrow \quad (Z, C, c) \in (Y, B, b)\end{array}$$

et que l'on notera $(X, A, a) \subset (Y, B, b)$ par la suite.

L'axiome des parties Soit (X, A, a) un graphe pointé. L'ensemble des parties de (X, A, a) (au sens de la relation SUB) est représenté par le graphe pointé (T, R, r) dont les composantes sont données par

- $T \;\equiv\; \mathsf{sum}(X, X{\to}\mathsf{Prop})$;
- R est la relation binaire sur $\mathsf{sum}(X, X{\to}\mathsf{Prop})$ définie par les clauses suivantes :
  1. si $A(x', x)$, alors $R\big(\mathsf{inl}(X, X{\to}\mathsf{Prop}, x'), \mathsf{inl}(X, X{\to}\mathsf{Prop}, x)\big)$;
  2. si $A(x', a)$ et $p(x')$, alors $R\big(\mathsf{inl}(X, X{\to}\mathsf{Prop}, x'), \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$;
  3. $R\big(\mathsf{inr}(X, X{\to}\mathsf{Prop}, p), \mathsf{out}(X, Y)\big)$ (pour tout prédicat $p : X \to \mathsf{Prop}$).
- $r \;\equiv\; \mathsf{out}(X, X{\to}\mathsf{Prop})$.

Formellement, la relation R est définie par

$$R \quad \equiv \quad \mathsf{POWER}(X, A, a),$$

où POWER désigne le constructeur de relation défini dans le système $F\omega.3$ par :

$$\begin{array}{l}\mathsf{POWER} \;:\; \Pi X : \mathsf{Type}_2 \,.\, (X{\to}X{\to}\mathsf{Prop}) \to X \to \\ \qquad\qquad \mathsf{sum}(X, X{\to}\mathsf{Prop}) \to \mathsf{sum}(X, X{\to}\mathsf{Prop}) \to \mathsf{Prop} \quad := \\ \\ \quad \lambda(X, A, a) \,.\, \lambda z', z : \mathsf{sum}(X, X{\to}\mathsf{Prop}) \,. \\ \qquad \big(\exists x', x : X \,. \qquad\qquad z' = \mathsf{inl}(X, X{\to}\mathsf{Prop}, x') \;\wedge \\ \qquad\qquad\qquad\qquad\quad z = \mathsf{inl}(X, X{\to}\mathsf{Prop}, x) \;\wedge\; A(x', x)\big) \\ \vee \quad \big(\exists x' : X \,.\, \exists p : (X{\to}\mathsf{Prop}) \,. \quad z' = \mathsf{inl}(X, X{\to}\mathsf{Prop}, x') \;\wedge \\ \qquad\qquad\qquad\qquad\quad z = \mathsf{inr}(X, X{\to}\mathsf{Prop}, p) \;\wedge\; A(x', a) \;\wedge\; p(x')\big) \\ \vee \quad \big(\exists p : (X{\to}\mathsf{Prop}) \,. \qquad z' = \mathsf{inr}(X, X{\to}\mathsf{Prop}, p) \;\wedge\; z = \mathsf{out}(X, X{\to}\mathsf{Prop})\big).\end{array}$$

Intuitivement, le graphe pointé (T, R, r) est construit en ajoutant au graphe (X, A, a) (délocalisé dans la première composante du type $\mathsf{sum}(X, Y)$) un nouveau sommet pour chaque prédicat p de type $X \to \mathsf{Prop}$. Ainsi que nous le verrons dans la preuve du lemme 9.4.11, cette construction est basée sur l'idée que chaque graphe pointé (Y, B, b) qui est un "sous-ensemble"

de (X, A, a) peut être représenté dans la seconde composante du type $\mathsf{sum}(X, X{\to}\mathsf{Prop})$ par le prédicat $p : X \to \mathsf{Prop}$ défini par

$$p \quad \equiv \quad \lambda x : X\,. \quad A(x,a) \ \wedge\ (X,A,x) \in (Y,B,b),$$

et dont les "éléments" sont les sommets $x : X$ représentant les éléments de (Y, B, b) (c'est-à-dire tels que $(X, A, x) \in (Y, B, b)$). Pour que cette représentation ait un sens, il faut également supposer que les sommets x tels que $p(x)$ représentent également des éléments du graphe pointé (X, A, a), d'où la condition $A(x, a)$.

De cette manière, la structure de graphe sur le type $\mathsf{sum}(X, X{\to}\mathsf{Prop})$ est construite en transportant la structure du graphe (X, A) par l'injection $\mathsf{inl}(X, X{\to}\mathsf{Prop})$ (clause 1), puis en connectant chaque sommet $x' : X$ tel que $A(x', a)$ à tous les prédicats $p : X \to \mathsf{Prop}$ satisfaisant $p(x')$ (clause 2), et enfin en connectant chaque prédicat $p : X{\to}\mathsf{Prop}$ à la nouvelle racine $r \equiv \mathsf{out}(X, X{\to}\mathsf{Prop})$ (clause 3). Notons que cette construction n'est possible que parce que les types X et $X{\to}\mathsf{Prop}$ sont au même niveau, c'est-à-dire dans la sorte Type_2.

Comme par ailleurs l'injection $\mathsf{inl}(X, X{\to}\mathsf{Prop}) : X \to \mathsf{sum}(X, X{\to}\mathsf{Prop})$ est un plongement du graphe (X, A) dans le graphe (T, R), on a :

Lemme 9.4.10 (Délocalisation) — *Pour tout sommet $x : X$, on a l'équivalence de bissimulation*

$$(X, A, x) \ \approx\ \big(T, R, \mathsf{inl}(X, X{\to}\mathsf{Prop}, x)\big).$$

Lemme 9.4.11 (Introduction) — *Si (Y, B, b) est un graphe pointé inclus dans (X, A, a), alors (Y, B, b) appartient à (T, R, r) :*

$$\forall (Y,B,b)\,. \quad (Y,B,b) \subset (X,A,a) \quad \Rightarrow \quad (Y,B,b) \in (T,R,r).$$

Preuve. Soit (Y, B, b) un graphe pointé tel que $(Y, B, b) \subset (X, A, a)$. Considérons le prédicat $p : X \to \mathsf{Prop}$ défini par

$$p \quad \equiv \quad \lambda x : X\,. \quad A(x,a) \ \wedge\ (X,A,x) \in (Y,B,b).$$

Pour montrer que les graphes pointés (Y, B, b) et $\big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$ sont bissimilaires, il suffit de montrer (Prop. 9.3.4) qu'ils ont exactement les mêmes éléments.

- (Y, B, b) *est inclus dans* $\big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$. Soit (Z, C, c) un graphe pointé tel que $(Z, C, c) \in (Y, B, b)$. Comme $(Y, B, b) \subset (X, A, a)$, on a $(Z, C, c) \in (X, A, a)$, et il existe un sommet $x' : X$ tel que $(Z, C, c) \approx (X, A, x')$ et $A(x', a)$. Comme par hypothèse $(Z, C, c) \in (Y, B, b)$, nous avons donc $(X, A, x') \in (Y, B, b)$ (Prop. 9.3.2) d'où il ressort que $p(x')$. D'après la clause 2, nous avons la relation

  $$R\big(\mathsf{inl}(X, X{\to}\mathsf{Prop}, x'),\ \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big),$$

  et comme les graphes pointés (Z, C, c), (X, A, x') et $\big(T, R, \mathsf{inl}(X, X{\to}\mathsf{Prop}, x')\big)$ sont bissimilaires, on a finalement $(Z, C, c) \in \big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$.
- $\big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$ *est inclus dans* (Y, B, b). Soit (Z, C, c) un graphe pointé tel que $(Z, C, c) \in \big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$. D'après cette hypothèse, il existe un sommet $s : T$ tel que $(Z, C, c) \approx (T, R, s)$ et $R\big(s, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$. D'après la définition de R, l'assertion $R\big(s, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$ provient de la clause 2, d'où il existe un sommet

$x' : X$ tel que $s = \mathsf{inl}(X, X{\to}\mathsf{Prop}, x')$, $A(x', a)$ et $p(x')$. Par définition du prédicat p, nous avons donc $(X, A, x') \in (Y, B, b)$, et comme

$$(Z, C, c) \quad \approx \quad \big(T, R, \mathsf{inl}(X, X{\to}\mathsf{Prop}, x')\big) \quad \approx \quad (X, A, x')$$

il vient $(Z, C, c) \in (Y, B, b)$ d'après un argument de compatibilité immédiat.

Nous venons de montrer que $(Y, B, b) \approx \big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$. La 3ème clause définissant R nous donne $R\big(\mathsf{inr}(X, X{\to}\mathsf{Prop}, p), r\big)$, d'où il ressort que $(Y, B, b) \in (T, R, r)$. □

Lemme 9.4.12 (Élimination) — *Tous les graphes pointés appartenant à (T, R, r) sont inclus dans (X, A, a) :*

$$\forall (Y, B, b)\,.\quad (Y, B, b) \in (T, R, r) \quad \Rightarrow \quad (Y, B, b) \subset (X, A, a).$$

Preuve. Supposons que $(Y, B, b) \in (T, R, r)$. Soit $r' : T$ un sommet tel que $(Y, B, b) \approx (T, R, r')$ et $R(r', r)$. Comme $r \equiv \mathsf{out}(X, X{\to}\mathsf{Prop})$, l'assertion $R(r', r)$ provient de la 3ème clause définissant R, d'où il existe un prédicat $p : X \to \mathsf{Prop}$ tel que $r' = \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)$. Pour montrer l'inclusion $(Y, B, b) \subset (X, A, a)$, considérons un graphe pointé (Z, C, c) tel que $(Z, C, c) \in (Y, B, b)$. Comme $(Y, B, b) \approx \big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$, on a $(Z, C, c) \in \big(T, R, \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$, d'où il existe un sommet $r'' : T$ tel que $(Z, C, c) \approx (T, R, r'')$ et $R\big(r'', \mathsf{inr}(X, X{\to}\mathsf{Prop}, p)\big)$. Cette dernière assertion ne peut provenir que de la 2ème clause définissant R, d'où il existe un sommet $x'' : X$ tel que $r'' = \mathsf{inl}(X, X{\to}\mathsf{Prop}, x'')$, $A(x'', a)$ et $p(x'')$. D'après le lemme de délocalisation, on a

$$(Z, C, c) \quad \approx \quad \big(T, R, \mathsf{inl}(X, X{\to}\mathsf{Prop}, x'')\big) \quad \approx \quad (X, A, x''),$$

d'où il ressort que $(Z, C, c) \in (X, A, a)$ puisque $A(x'', a)$. □

Les deux lemmes ci-dessus nous permettent de conclure que la traduction de l'axiome des parties est une proposition prouvable dans le système $F\omega.3$:

Proposition 9.4.13 (Axiome des parties) — *Pour tout graphe pointé (X, A, a), il existe un graphe pointé (T, R, r) dont les éléments sont exactement tous les graphes pointés inclus dans (X, A, a) :*

$$\begin{array}{l}\forall (X, A, a)\,.\quad \exists (T, R, r)\,.\quad \forall (Y, B, b)\,.\\ \quad (Y, B, b) \in (T, R, r) \quad \Leftrightarrow \quad (Y, B, b) \subset (X, A, a).\end{array}$$

9.4.4 L'axiome de la réunion

Soit (X, A, a) un graphe pointé quelconque. La réunion des éléments de (X, A, a) est représentée par le graphe (T, R, r) dont les composantes sont définies par

- $T \equiv \mathsf{opt}(X)$;
- R est la relation binaire engendrée par les clauses suivantes :
  1. si $A(x', x)$, alors $R\big(\mathsf{some}(X, x'), \mathsf{some}(X, x)\big)$;
  2. si $A(x', x)$ et $A(x, a)$, alors $R\big(\mathsf{some}(X, x'), \mathsf{none}(X)\big)$.
- $r \equiv \mathsf{none}(X)$.

Formellement, la relation R est définie par

$$R \quad \equiv \quad \mathsf{UNION}(X, A, a),$$

où UNION désigne le constructeur de relation défini dans le système $F\omega.3$ par :

$$\begin{array}{l}\mathsf{UNION} \;:\; \Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \\ \qquad\qquad \mathsf{opt}(X) \to \mathsf{opt}(X) \to \mathsf{Prop} \quad := \\ \\ \lambda(X, A, a) \,.\quad \lambda z', z : \mathsf{opt}(X) \,. \\ \qquad \big(\exists x', x : X \,.\quad z' = \mathsf{some}(X, x') \quad\wedge\quad z = \mathsf{some}(X, x) \quad\wedge\quad A(x', x)\big) \\ \vee \quad \big(\exists x', x : X \,.\quad z' = \mathsf{some}(X, x') \quad\wedge\quad z = \mathsf{none}(X) \qquad\wedge\quad A(x', x) \quad\wedge\quad A(x, a)\big).\end{array}$$

Intuitivement, le graphe pointé (T, R, r) est obtenu à partir du graphe pointé (X, A, a) en reliant tous les sommets situés deux arcs en dessous de la racine a à la nouvelle racine $r \equiv \mathsf{none}(X)$ (clause 2). Comme l'injection $\mathsf{some}(X) : X \to \mathsf{opt}(X)$ est clairement un plongement du graphe (X, A) dans le graphe (T, R), le lemme 9.3.12 nous donne la propriété attendue :

Lemme 9.4.14 (Délocalisation) — *Pour tout sommet $x : X$ on a l'équivalence de bissimulation*

$$(X, A, x) \quad \approx \quad \big(T, R, \mathsf{some}(X, x)\big).$$

Lemme 9.4.15 (Introduction) — *Si (Y, B, b) et (Z, C, c) sont deux graphes pointés tels que $(Y, B, b) \in (Z, C, c)$ et $(Z, C, c) \in (X, A, a)$, alors $(Y, B, b) \in (T, R, r)$:*

$$\begin{array}{l}\forall (Y, B, b), (Z, C, c) \,. \\ \quad (Y, B, b) \in (Z, C, c) \quad\Rightarrow\quad (Z, C, c) \in (X, A, a) \quad\Rightarrow\quad (Y, B, b) \in (T, R, r).\end{array}$$

Preuve. Soient (Y, B, b) et (Z, C, c) deux graphes pointés tels que $(Y, B, b) \in (Z, C, c)$ et $(Z, C, c) \in (X, A, a)$. D'après ces deux relations, il existe des sommets $x, x' : X$ tels que

$$(Y, B, b) \approx (X, A, x'), \qquad (Z, C, c) \approx (X, A, x), \qquad A(x', x) \qquad \text{et} \qquad A(x, a).$$

Les relations $A(x', x)$ et $A(x, a)$ nous donnent $R\big(\mathsf{some}(X, x'), r\big)$ (2ème clause définissant R), et comme $(X, A, x') \approx \big(T, R, \mathsf{some}(X, x')\big)$, il vient $(Y, B, b) \approx (X, A, x') \in (T, R, r)$. □

Lemme 9.4.16 (Élimination) — *Pour tout graphe pointé (Y, B, b) appartenant au graphe pointé (T, R, r), il existe un graphe pointé (Z, C, c) tel que $(Y, B, b) \in (Z, C, c)$ et $(Z, C, c) \in (X, A, a)$:*

$$\begin{array}{l}\forall (Y, B, b) \,.\quad (Y, B, b) \in (T, R, r) \quad\Rightarrow \\ \quad \exists (Z, C, c) \,.\quad (Y, B, b) \in (Z, C, c) \quad\wedge\quad (Z, C, c) \in (X, A, a).\end{array}$$

Preuve. Soient (Y, B, b) un graphe pointé tel que $(Y, B, b) \in (T, R, r)$, et $r' : T$ un sommet tel que $(Y, B, b) \approx (T, R, r')$ et $R(r', r)$. Comme $r = \mathsf{none}(X)$, l'assertion $R(r', r)$ provient de la 2ème clause définissant R, d'où il existe des sommets $x, x' : X$ tels que $r' = \mathsf{some}(X, x')$, $A(x', x)$ et $A(x, a)$. Des équivalences de bissimulation

$$(Y, B, b) \;\approx\; \big(T, R, \mathsf{some}(X, x')\big) \qquad \text{et} \qquad \big(T, R, \mathsf{some}(X, x)\big) \;\approx\; (X, A, x)$$

découlent immédiatement les relations

$$(Y, B, b) \;\in\; \big(T, R, \mathsf{some}(X, x)\big) \qquad \text{et} \qquad \big(T, R, \mathsf{some}(X, x)\big) \;\in\; (X, A, a)$$

puisque $R\big(\mathsf{some}(X, x'), \mathsf{some}(X, x)\big)$ (d'après la clause 1) et $A(x, a)$. □

D'après les deux lemmes ci-dessus, la traduction de l'axiome de la réunion est prouvable dans le système $F\omega.3$:

Proposition 9.4.17 (Axiome de la réunion) — *Quel que soit le graphe pointé* (X, A, a)*, il existe un graphe pointé* (T, R, r) *dont les éléments sont exactement les graphes pointés* (Y, B, b) *appartenant à au moins un élément de* (X, A, a) *:*

$$\begin{array}{l}\forall (X, A, a)\,.\quad \exists (T, R, r)\,.\quad \forall (Y, B, b)\,.\\ \quad (Y, B, b) \in (T, R, r) \quad\Leftrightarrow\quad \big(\exists (Z, C, c)\,.\quad (Y, B, b) \in (Z, C, c) \quad\wedge\quad (Z, C, c) \in (X, A, a)\big)\end{array}$$

9.4.5 Ensemble vide et successeur

Avant de nous attaquer à la construction du graphe pointé représentant l'ensemble des entiers de von Neumann, nous devons d'abord construire le graphe pointé représentant $0 = \varnothing$ ainsi que le graphe pointé représentant le successeur de x, noté $s(x)$ et égal à $x \cup \{x\}$, pour chaque ensemble x.

Dans la théorie des ensembles de Zermelo, l'existence de l'ensemble vide se prouve à l'aide d'une instance particulière du schéma de compréhension, en considérant l'ensemble des $x \in a$ tels que $\bot$, où a désigne un ensemble quelconque. L'existence de l'ensemble $s(x) = x \cup \{x\}$ est une conséquence de l'axiome de la paire et de l'axiome de la réunion, puisque $s(x)$ n'est rien d'autre que la réunion des éléments de la paire $\{x; \{x\}\}$ (où le singleton $\{x\}$ est lui-même construit comme la paire $\{x; x\}$).

Bien qu'il soit possible de simuler ces constructions dans le système $F\omega.3$ (en utilisant les traductions des axiomes de compréhension, de la paire et de la réunion), il est plus simple de construire les graphes pointés correspondants de manière directe.

Construction du graphe pointé vide L'ensemble vide est représenté par le graphe pointé $(\mathsf{unit}, \mathsf{VOID}, \mathsf{id})$ dont les composantes sont données par

- $\mathsf{unit} \;\equiv\; \Pi X : \mathsf{Type}_1\,.\, X \to X$;
- $\mathsf{VOID} \;\equiv\; \lambda x, y : \mathsf{unit}\,.\, \bot$;
- $\mathsf{id} \;\equiv\; \lambda X : \mathsf{Type}_1\,.\, \lambda x : X\,.\, x$.

Comme par définition, le graphe pointé $(\mathsf{unit}, \mathsf{VOID}, \mathsf{id})$ ne comporte aucun arc (*i.e.* la proposition $\mathsf{VOID}(x, y)$ est toujours fausse), on a de manière immédiate :

Lemme 9.4.18 (Élimination) — *Le graphe pointé* $(\mathsf{unit}, \mathsf{VOID}, \mathsf{id})$ *n'a aucun élément :*

$$\forall (X, A, a)\,.\quad \neg\ (X, A, a) \in (\mathsf{unit}, \mathsf{VOID}, \mathsf{id}).$$

Il est facile de vérifier à l'aide de la propriété d'extensionnalité (Prop. 9.3.4) que le graphe pointé vide est unique à une bissimulation près.[13]

[13] Notons que le choix du type unit est purement arbitraire : nous aurions pu construire le graphe pointé vide à partir de n'importe quel type X habité et en plaçant la racine sur n'importe quel objet de type X.

Construction du successeur Soit (X, A, a) un graphe pointé. Le successeur de (X, A, a) est représenté par le graphe pointé (T, R, r) dont les composantes sont données par :

- $T \equiv \mathsf{opt}(X)$;
- R est la relation engendrée par les clauses suivantes :
  1. si $A(x', x)$, alors $R\big(\mathsf{some}(X, x'), \mathsf{some}(X, x)\big)$;
  2. si $A(x', a)$, alors $R\big(\mathsf{some}(X, x'), \mathsf{none}(X)\big)$;
  3. $R\big(\mathsf{some}(X, a), \mathsf{none}(X)\big)$.
- $r \equiv \mathsf{none}(X)$.

Formellement, la relation binaire R sur le type $\mathsf{opt}(X)$ est définie par

$$R \quad \equiv \quad \mathsf{SUCC}(X, A, a)$$

où SUCC désigne le constructeur de relation défini dans le système $F\omega.3$ par :

$$\begin{array}{l} \mathsf{SUCC} \;:\; \Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \\ \qquad\qquad \mathsf{opt}(X) \to \mathsf{opt}(X) \to \mathsf{Prop} \quad := \\ \\ \lambda(X, A, a) \,.\quad \lambda z', z : \mathsf{opt}(X) \,. \\ \qquad \big(\exists x', x : X \,.\quad z' = \mathsf{some}(X, x') \;\wedge\; z = \mathsf{some}(X, x) \;\wedge\; A(x', x)\big) \\ \vee \quad \big(\exists x' : X \,.\qquad z' = \mathsf{some}(X, x') \;\wedge\; z = \mathsf{none}(X) \;\wedge\; A(x', a)\big) \\ \vee \quad \big(z' = \mathsf{some}(X, a) \;\wedge\; z = \mathsf{none}(X)\big). \end{array}$$

Intuitivement, le graphe pointé (T, R, r) est formé à partir d'une copie du graphe pointé (X, A, a) (clause 1) dans laquelle on a connecté à la nouvelle racine $r \equiv \mathsf{none}(X)$ tous les sommets de X connectés à l'ancienne racine a (clause 2) ainsi que l'ancienne racine a elle-même (clause 3). D'après la définition de la relation R, il est clair que l'injection $\mathsf{some}(X)$ de type $X \to \mathsf{opt}(X)$ est un plongement du graphe (X, A) dans le graphe (T, R), d'où il ressort d'après le lemme 9.3.12 que :

Lemme 9.4.19 (Délocalisation) — *Pour tout sommet $x : X$, on a l'équivalence de bissimulation :*

$$(X, A, x) \quad \approx \quad \big(T, R, \mathsf{some}(X, x)\big).$$

Proposition 9.4.20 (Caractérisation du successeur) — *Les éléments du graphe pointé (T, R, r) sont exactement les graphes pointés (Y, B, b) appartenant à (X, A, a) ou bissimilaires à (X, A, a) :*

$$\forall (Y, B, b) \,. \qquad (Y, B, b) \in (T, R, r) \quad \Leftrightarrow \quad (Y, B, b) \in (X, A, a) \;\vee\; (Y, B, b) \approx (X, A, a).$$

Preuve. Comme $(X, A, a) \approx \big(T, R, \mathsf{some}(X, a)\big)$ et $R(\mathsf{some}(X, a), r)$ (clause 3), on a la relation $(X, A, a) \in (T, R, r)$. De plus, si $(Y, B, b) \in (X, A, a)$, il existe un sommet $x' : X$ tel que $(Y, B, b) \approx (X, A, x')$ et $A(x', a)$. On a alors $(Y, B, b) \approx (X, A, x') \approx \big(T, R, \mathsf{some}(X, x')\big)$ et $R(\mathsf{some}(X, x'), r)$ (clause 2), d'où l'on tire $(Y, B, b) \in (T, R, r)$.

Réciproquement, considérons un graphe pointé (Y, B, b) tel qu'on ait $(Y, B, b) \in (T, R, r)$. D'après cette hypothèse, il existe un sommet $r' : T$ tel que $(Y, B, b) \approx (T, R, r')$ et $R(r', r)$. Puisque $r \equiv \mathsf{none}(X)$, la relation $R(r', r)$ ne peut provenir que de la clause 2 ou de la clause 3. S'il s'agit de la clause 2, on montre aisément que $(Y, B, b) \in (X, A, a)$. S'il s'agit de la clause 3, on en déduit de la même façon que $(Y, B, b) \approx (X, A, a)$. □

Lemme 9.4.21 (Compatibilité) — *Si* (X, A, a) *et* (Y, B, b) *sont des graphes pointés bissimilaires, alors leurs successeurs respectifs le sont également :*

$$\begin{array}{l}\forall (X,A,a),(Y,B,b)\,.\quad (X,A,a) \approx (Y,B,b) \quad \Rightarrow \\ \quad \big(\mathsf{opt}(X), \mathsf{SUCC}(X,A,a), \mathsf{none}(X)\big) \quad \approx \quad \big(\mathsf{opt}(Y), \mathsf{SUCC}(Y,B,b), \mathsf{none}(Y)\big).\end{array}$$

Preuve. En utilisant la caractérisation des éléments du successeur par la proposition précédente, on montre aisément que l'hypothèse $(X, A, a) \approx (Y, B, b)$ entraîne l'inclusion

$$\big(\mathsf{opt}(X), \mathsf{SUCC}(X,A,a), \mathsf{none}(X)\big) \quad \subset \quad \big(\mathsf{opt}(Y), \mathsf{SUCC}(Y,B,b), \mathsf{none}(Y)\big).$$

L'inclusion réciproque découle d'un argument de symétrie immédiat, et on conclut par extensionnalité (Prop. 9.3.4). □

9.4.6 L'axiome de l'infini

Dans ce paragraphe, nous nous proposons de construire dans le système $F\omega.3$ le graphe pointé représentant l'ensemble des entiers de von Neumann à partir du type nat que nous avons introduit dans la section 9.2.

Des entiers de Church aux entiers de von Neumann Dans les présentations usuelles de la théorie des ensembles, on introduit généralement l'ensemble des entiers de von Neumann (noté ω) comme le plus petit ensemble contenant l'ensemble vide $\varnothing$ et stable par l'opération qui à tout x associe son successeur $s(x) = x \cup \{x\}$.[14] On définit alors la relation d'ordre strict sur ω en posant $n < m \;\equiv\; n \in m$, et on vérifie ensuite la correction de cette définition en montrant qu'elle permet de dériver toutes les propriétés arithmétiques attendues.

Dans ce paragraphe, nous allons reprendre la même idée, mais en effectuant le chemin dans le sens inverse. Autrement dit, nous allons partir des propriétés arithmétiques établies au paragraphe 9.2.3 pour introduire la relation $n < m$, puis nous montrerons qu'en munissant le type nat de cette relation, on obtient une structure de graphe $(\mathsf{nat}, \mathsf{lt})$ dont la propriété essentielle est qu'à chaque fois qu'on positionne la racine sur un entier de Church n bien formé, on obtient un graphe pointé $(\mathsf{nat}, \mathsf{lt}, n)$ qui est précisément la représentation de l'entier de von Neumann correspondant.

Pour cela, on considère la relation binaire lt définie par :

$$\mathsf{lt} \quad : \quad \mathsf{nat} \to \mathsf{nat} \to \mathsf{Prop} \quad := \quad \lambda n, m : \mathsf{nat}\,. \quad \mathsf{wf\_nat}(n) \quad \wedge \quad \mathsf{le}(\mathsf{S}(n), m).$$

Contrairement à la définition de la relation le donnée au paragraphe 9.2.3, nous avons incorporé dans la définition de l'ordre strict lt l'hypothèse de bonne formation de l'entier n, pour des raisons de commodité qui apparaîtront dans les preuves qui suivent.

Lemme 9.4.22 (Bonne formation) — *Si* n *et* m *sont deux objets de type* nat *tels que* $\mathsf{lt}(n, m)$*, alors* n *et* m *sont des entiers bien formés :*

$$\forall n, m : \mathsf{nat}\,. \quad \mathsf{lt}(n,m) \quad \Rightarrow \quad \mathsf{wf\_nat}(n) \quad \wedge \quad \mathsf{wf\_nat}(m).$$

[14] Bien entendu, la construction du *plus petit* ensemble contenant $\varnothing$ et stable par $x \mapsto s(x)$ nécessite de savoir qu'il existe au moins un ensemble contenant l'ensemble vide et stable par la fonction successeur (mais pas nécessairement le plus petit), ce qui est précisément l'objet de l'axiome de l'infini.

Preuve. Résulte des propositions 9.2.1 et 9.2.5 (4ème point). □

Proposition 9.4.23 (Propriétés de la relation $\mathsf{lt}(n,m)$) — *La relation $\mathsf{lt}(n,m)$ satisfait les propriétés suivantes :*

1. $\forall n,m:\mathsf{nat}.\quad \mathsf{wf\_nat}(n) \quad\Rightarrow\quad \mathsf{lt}(n,\mathsf{S}(n))$
2. $\forall n:\mathsf{nat}.\quad \mathsf{lt}(n,m) \quad\Rightarrow\quad \mathsf{lt}(n,\mathsf{S}(m))$
3. $\forall n:\mathsf{nat}.\quad \neg\,\mathsf{lt}(n,\mathsf{0})$
4. $\forall n,m:\mathsf{nat}.\quad \mathsf{wf\_nat}(m) \quad\Rightarrow\quad \mathsf{lt}(n,\mathsf{S}(m)) \quad\Rightarrow\quad \mathsf{lt}(n,m) \quad\vee\quad n=m$

Preuve. Les points 1, 2 et 3 résultent directement de la proposition 9.2.5. Le dernier point est une conséquence immédiate de la proposition 9.2.7 (la preuve utilise l'injectivité de la fonction successeur, d'où la nécessité de supposer que m est un entier bien formé). □

Notons que les quatre formules établies par la proposition ci-dessus n'ont pas été choisies au hasard. Si on remplace mentalement la relation $\mathsf{lt}(n,m)$ par la relation d'appartenance $n \in m$, on s'aperçoit immédiatement que les points 1, 2 et 4 correspondent à la définition ensembliste du successeur de m (les points 1 et 2 donnant les conditions d'introduction, tandis que le 4ème point donne la condition d'élimination), et que le 3ème point correspond à la caractérisation de l'ensemble vide. Cette remarque permet alors de démontrer les deux lemmes suivants :

Lemme 9.4.24 — *Le graphe pointé $(\mathsf{nat},\mathsf{lt},\mathsf{0})$ est bissimilaire au graphe pointé vide :*

$$(\mathsf{nat},\mathsf{lt},\mathsf{0}) \quad\approx\quad (\mathsf{unit},\mathsf{VOID},\mathsf{id})$$

Preuve. Il suffit pour cela de vérifier que le graphe pointé $(\mathsf{nat},\mathsf{lt},\mathsf{0})$ n'a aucun élément, ce qui est trivial d'après le 3ème point de la proposition précédente. □

Lemme 9.4.25 — *Si n est un entier bien formé, le graphe pointé $(\mathsf{nat},\mathsf{lt},\mathsf{S}(n))$ est bissimilaire au successeur du graphe pointé $(\mathsf{nat},\mathsf{lt},n)$:*

$$\forall n:\mathsf{nat}.\quad \mathsf{wf\_nat}(n) \quad\Rightarrow\quad (\mathsf{nat},\mathsf{lt},\mathsf{S}(n)) \quad\approx\quad \big(\mathsf{opt}(\mathsf{nat}),\mathsf{SUCC}(\mathsf{nat},\mathsf{lt},n),\mathsf{none}(\mathsf{nat})\big).$$

Preuve. Soit n un entier bien formé. Pour montrer que les graphes pointés $(\mathsf{nat},\mathsf{lt},\mathsf{S}(n))$ et $\big(\mathsf{opt}(\mathsf{nat}),\mathsf{SUCC}(\mathsf{nat},\mathsf{lt},n),\mathsf{none}(\mathsf{nat})\big)$ sont bissimilaires, il suffit de montrer qu'ils ont les mêmes éléments. L'inclusion directe se prouve à l'aide du 4ème point de la proposition 9.4.23, et l'inclusion réciproque découle des points 1 et 2 (en utilisant dans chacun des deux sens la caractérisation des éléments du graphe pointé successeur par la proposition 9.4.20). □

D'après les deux lemmes ci-dessus, il est clair (par une récurrence immédiate) que pour tout entier n, le graphe pointé $(\mathsf{nat},\mathsf{lt},\overline{n})$ représente l'entier de von Neumann n (où $\overline{n}$ désigne l'entier de Church correspondant dans le type nat).

Le graphe pointé des entiers de von Neumann Dans le graphe $(\mathsf{nat},\mathsf{lt})$, chaque entier de von Neumann est représenté par l'entier de Church correspondant. Comme en revanche il n'existe aucun sommet pour représenter l'ensemble de tous les entiers de von Neumann, nous devons ajouter à ce graphe un sommet supplémentaire, que nous connecterons à tous les autres. Pour cela, on introduit un nouveau graphe pointé

$$\big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big)$$

basé sur le type $\mathsf{opt}(\mathsf{nat})$, et dont la relation binaire OMEGA est définie par les deux clauses suivantes :

1. si $\mathsf{lt}(n, m)$, alors $\mathsf{OMEGA}\big(\mathsf{some}(\mathsf{nat}, n), \mathsf{some}(\mathsf{nat}, m)\big)$;
2. si $\mathsf{wf\_nat}(n)$, alors $\mathsf{OMEGA}\big(\mathsf{some}(\mathsf{nat}, n), \mathsf{none}(\mathsf{nat})\big)$.

Dans le système $F\omega.3$, la relation OMEGA est donnée formellement par :

$$\begin{array}{l} \mathsf{OMEGA} \quad : \quad \mathsf{opt}(\mathsf{nat}) \to \mathsf{opt}(\mathsf{nat}) \to \mathsf{Prop} \quad := \\ \\ \quad \lambda x, y : \mathsf{opt}(\mathsf{nat})\,. \\ \qquad \big(\exists n, m : \mathsf{nat}\,. \quad x = \mathsf{some}(\mathsf{nat}, n) \quad \wedge \quad y = \mathsf{some}(\mathsf{nat}, m) \quad \wedge \quad \mathsf{lt}(n, m)\big) \\ \quad \vee \quad \big(\exists n : \mathsf{nat}\,. \quad\;\; x = \mathsf{some}(\mathsf{nat}, n) \quad \wedge \quad y = \mathsf{none}(\mathsf{nat}) \quad \wedge \quad \mathsf{wf\_nat}(n)\big). \end{array}$$

Il est clair que l'injection $\mathsf{some}(\mathsf{nat}) : \mathsf{nat} \to \mathsf{opt}(\mathsf{nat})$ est un plongement du graphe $(\mathsf{nat}, \mathsf{lt})$ dans le graphe $\big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}\big)$, d'où il ressort que :

Lemme 9.4.26 (Délocalisation) — *Pour tout objet* $n : \mathsf{nat}$ *on a l'équivalence de bissimulation :*

$$(\mathsf{nat}, \mathsf{lt}, n) \quad \approx \quad \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{some}(\mathsf{nat}, n)\big).$$

Des résultats que nous avons déjà démontrés nous pouvons maintenant déduire la prouvabilité de la traduction de l'axiome de l'infini dans le système $F\omega.3$:

Proposition 9.4.27 (Axiome de l'infini) — *Le graphe pointé*

$$\big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big)$$

contient le graphe pointé vide, et est stable par la fonction qui à tout graphe pointé associe son successeur :

1. $(\mathsf{unit}, \mathsf{VOID}, \mathsf{id}) \;\in\; \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big)$
2. $\forall (X, A, a)\,. \quad (X, A, a) \;\in\; \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big) \quad \Rightarrow$
$\big(\mathsf{opt}(X), \mathsf{SUCC}(X, A, a), \mathsf{none}(X)\big) \;\in\; \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big).$

Preuve. Le premier point est évident d'après les équivalences

$$(\mathsf{unit}, \mathsf{VOID}, \mathsf{id}) \quad \approx \quad (\mathsf{nat}, \mathsf{lt}, 0) \quad \approx \quad \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{some}(\mathsf{nat}, 0)\big)$$

et la relation $\mathsf{OMEGA}(\mathsf{some}(\mathsf{nat}, 0), \mathsf{none}(\mathsf{nat}))$ (clause 2). Supposons à présent que (X, A, a) est un graphe pointé appartenant à $\big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big)$. D'après cette hypothèse, il existe $x : \mathsf{opt}(\mathsf{nat})$ tel que $(X, A, a) \approx \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, x\big)$ et $\mathsf{OMEGA}\big(x, \mathsf{none}(\mathsf{nat})\big)$. D'après la définition du terme OMEGA, la relation $\mathsf{OMEGA}\big(x, \mathsf{none}(\mathsf{nat})\big)$ ne peut provenir que de la clause 2, d'où il existe un entier n bien formé tel que $x = \mathsf{some}(\mathsf{nat}, n)$. D'après le lemme de délocalisation, on a alors

$$(X, A, a) \quad \approx \quad \big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{some}(\mathsf{nat}, n)\big) \quad \approx \quad (\mathsf{nat}, \mathsf{lt}, n),$$

d'où il ressort d'après les lemmes 9.4.21 et 9.4.25 que

$$\begin{array}{rcl}\big(\mathsf{opt}(X),\mathsf{SUCC}(X,A,a),\mathsf{none}(X)\big) & \approx & \big(\mathsf{opt}(\mathsf{nat}),\mathsf{SUCC}(\mathsf{nat},\mathsf{lt},n),\mathsf{none}(\mathsf{nat})\big)\\ & \approx & (\mathsf{nat},\mathsf{lt},\mathsf{S}(n))\\ & \approx & \big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{some}(\mathsf{S}(n))\big).\end{array}$$

Comme $\mathsf{S}(n)$ est également un entier bien formé, nous avons $\mathsf{OMEGA}\big(\mathsf{some}(\mathsf{S}(n)),\mathsf{none}(\mathsf{nat})\big)$, d'où l'on tire que $\big(\mathsf{opt}(X),\mathsf{SUCC}(X,A,a),\mathsf{none}(X)\big) \in \big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big)$. □

Dans la théorie des ensembles de Zermelo, l'axiome de l'infini exprime l'existence d'un ensemble a contenant l'ensemble $0 = \varnothing$ et stable par la fonction successeur $x \mapsto s(x) = x \cup \{x\}$, mais il n'exprime pas le fait que cet ensemble est le plus petit possible (ce qui est indispensable pour dériver le principe de récurrence).

Pour construire le sous-ensemble $\omega \subset a$ dont les éléments sont exactement les entiers de von Neumann (et qui ne contient rien d'autre), on considère alors l'ensemble de toutes les parties de a (axiome des parties) contenant 0 et stables par successeur (compréhension), puis on calcule l'intersection des éléments de cet ensemble (compréhension) que l'on note ω. Il ne reste plus qu'à vérifier que l'ensemble ω contient 0 et est stable par successeur, et qu'il satisfait en outre le principe de récurrence, ce qui est immédiat par construction.[15]

Dans le système $F\omega.3$, le graphe pointé $\big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big)$ que nous venons de construire est en réalité le plus petit graphe pointé contenant le graphe pointé vide et stable par la construction successeur. Il n'est donc pas nécessaire d'effectuer la moindre construction supplémentaire pour pouvoir établir le principe de récurrence :

Proposition 9.4.28 (Principe de récurrence) — *Le graphe pointé*

$$\big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big)$$

est le plus petit graphe pointé contenant le graphe pointé vide, et stable par la fonction qui à tout graphe pointé associe son successeur :

$$\begin{array}{l}\forall P : \big(\Pi X : \mathsf{Type}_2 \,.\, (X{\to}X{\to}\mathsf{Prop}){\to}X{\to}\mathsf{Prop}\big)\,. \quad \mathsf{COMPAT}(P) \quad \Rightarrow\\ \quad P(\mathsf{unit},\mathsf{VOID},\mathsf{id}) \quad \Rightarrow\\ \quad \Big(\forall (X,A,a) \in \big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big)\,.\\ \qquad\quad P(X,A,a) \quad \Rightarrow \quad P\big(\mathsf{opt}(X),\mathsf{SUCC}(X,A,a),\mathsf{none}(X)\big)\Big) \quad \Rightarrow\\ \quad \forall (X,A,a) \in \big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big)\,. \quad P(X,A,a).\end{array}$$

Preuve. En utilisant la compatibilité du prédicat P, on montre que pour tout entier n bien formé, on a

$$(\mathsf{nat},\mathsf{lt},n) \;\in\; \big(\mathsf{opt}(\mathsf{nat}),\mathsf{OMEGA},\mathsf{none}(\mathsf{nat})\big) \qquad \text{et} \qquad P(\mathsf{nat},\mathsf{lt},n).$$

[15] On remarquera la très grande similarité entre cette construction de l'ensemble ω et la définition du prédicat wf_nat (*cf* paragraphe 9.2.1), ce qui a déjà été souligné à de nombreuses reprises. Cette remarque illustre le fait qu'en dépit d'une formulation au premier ordre, la théorie des ensembles (que ce soit Z ou ZF) est fondamentalement une théorie imprédicative, ce qui est dû à la formulation du schéma de compréhension dans laquelle la formule ϕ (dont les quantifications ne sont pas nécessairement bornées) peut référer à l'ensemble que l'on est en train de construire.

Cette démontration s'effectue par récurrence sur n (en utilisant l'hypothèse $\mathsf{wf\_nat}(n)$). Le cas de base est immédiat, et le cas inductif utilise l'équivalence de bissimulation du lemme 9.4.25 et la propriété de compatibilité du lemme 9.4.21. On conclut alors en remarquant que tous les graphes pointés appartenant à $\big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big)$ sont (à une équivalence de bissimulation près) de la forme $(\mathsf{nat}, \mathsf{lt}, n)$, où n est un entier bien formé, et qu'ils satisfont tous de ce fait la propriété P. □

9.5 Le type des graphes pointés dans Type_3

9.5.1 Une première traduction

Dans la section précédente, nous avons montré que la traduction de chacun des axiomes de la théorie des ensembles de Zermelo en termes de graphes pointés est une proposition prouvable dans le système $F\omega.3$. Pour cela, nous nous sommes appuyés implicitement sur une fonction de traduction qui à chaque formule ϕ de la théorie des ensembles associe une proposition ϕ^* dans le système $F\omega.3$. Nous allons maintenant expliciter cette fonction de traduction.

Considérons une énumération $(x_i)_{i\in\omega}$ de toutes les variables du langage de la théorie des ensembles. À chaque variable x_i de la théorie des ensembles, on associe trois variables X_i, A_i et a_i du système $F\omega.3$, de manière à satisfaire les deux critères suivants :

1. les variables X_i, A_i et a_i sont deux-à-deux distinctes ;
2. si $i \neq j$, alors $\{X_i; A_i; a_i\} \cap \{X_j; A_j; a_j\} = \varnothing$.

À toute formule ϕ de la théorie des ensembles, on associe le terme $\phi^\dagger$ défini par :

$$\begin{array}{lcl}
(x_i = x_j)^\dagger & \equiv & \mathsf{EQV}(X_i, A_i, a_i, X_j, A_j, a_j) \\
(x_i \in x_j)^\dagger & \equiv & \mathsf{ELT}(X_i, A_i, a_i, X_j, A_j, a_j) \\
(\bot)^\dagger & \equiv & \bot \\
(\phi \Rightarrow \psi)^\dagger & \equiv & \phi^\dagger \Rightarrow \psi^\dagger \\
(\phi \wedge \psi)^\dagger & \equiv & \phi^\dagger \wedge \psi^\dagger \\
(\phi \vee \psi)^\dagger & \equiv & \phi^\dagger \vee \psi^\dagger \\
(\forall x_i\ \phi)^\dagger & \equiv & \forall(X_i, A_i, a_i)\,.\,\phi^\dagger \\
(\exists x_i\ \phi)^\dagger & \equiv & \exists(X_i, A_i, a_i)\,.\,\phi^\dagger
\end{array}$$

Notons que dans cette correspondance, les symboles $\bot$, $\Rightarrow$, $\wedge$, $\vee$, $\forall$ et $\exists$ n'ont pas la même signification suivant qu'ils figurent dans la colonne de gauche ou dans la colonne de droite. Dans la colonne de gauche, ces symboles désignent les connecteurs et les quantificateurs usuels du langage des théories du premier ordre tandis que dans la colonne de droite, ils désignent les codages imprédicatifs standards des connecteurs logiques (intuitionnistes) et des quantificateurs (définis sur la classe des graphes pointés) dans le système $F\omega.3$.

Lemme 9.5.1 (Correction) — *Si ϕ est une formule de la théorie des ensembles dont les variables libres sont les variables $x_{i_1}, \ldots, x_{i_n}$, sa traduction $\phi^\dagger$ satisfait les propriétés suivantes :*

1. $FV(\phi^\dagger) = \{X_{i_1}, A_{i_1}, a_{i_1}, \ldots, X_{i_n}, A_{i_n}, a_{i_n}\}$;
2. *Dans toute signature Σ dans laquelle les variables X_{i_j}, A_{i_j} et a_{i_j} (pour tout $j \in [1..n]$) sont déclarées avec les types Type_2, $X_{i_j}{\to}X_{i_j}{\to}\mathsf{Prop}$ et X_{i_j} respectivement, le jugement de typage $\Sigma \vdash \phi^\dagger : \mathsf{Prop}$ est dérivable.*

Preuve. Par une récurrence immédiate sur la structure de la formule ϕ. □

Dans ce qui suit, on associera à tout ensemble fini de variables de la théorie des ensembles $\{x_{i_1}; \ldots; x_{i_n}\}$ une signature du système $F\omega.3$ notée $[x_{i_1}; \ldots; x_{i_n}]^\dagger$ et définie par :

$$[x_{i_1}; \ldots; x_{i_n}]^\dagger \equiv \left[\begin{array}{ccc} X_{i_1} : \mathsf{Type}_2 \; ; & A_{i_1} : X_{i_1} \to X_{i_1} \to \mathsf{Prop} \; ; & a_{i_1} : X_{i_1} \; ; \\ \vdots & \vdots & \vdots \\ X_{i_n} : \mathsf{Type}_2 \; ; & A_{i_n} : X_{i_n} \to X_{i_n} \to \mathsf{Prop} \; ; & a_{i_n} : X_{i_n} \end{array}\right].$$

Lemme 9.5.2 (Compatibilité) — *Pour toute formule ϕ de la théorie des ensembles et pour toute variable x_{i_0} de la théorie des ensembles, la proposition*

$$\mathsf{COMPAT}\big(\lambda(X_{i_0}, A_{i_0}, a_{i_0})\,.\;\phi^\dagger\big)$$

est prouvable dans le contexte vide sous la signature $[x_{i_1}, \ldots, x_{i_n}]^\dagger$, où $x_{i_1}, \ldots, x_{i_n}$ désignent les variables libres de la formule $(\forall x_{i_0}\ \phi)$.

Preuve. Par une récurrence immédiate sur la structure de la formule ϕ, en utilisant le lemme 9.3.2. □

Proposition 9.5.3 (Validité) — *Soit ϕ une formule de la théorie des ensembles dont les variables libres sont $x_{i_1}, \ldots, x_{i_n}$. Si ϕ est prouvable dans la théorie des ensembles de Zermelo intuitionniste, alors $\phi^\dagger$ est prouvable dans le contexte vide sous la signature $[x_{i_1}; \ldots; x_{i_n}]^\dagger$.*

Preuve. Ce résultat se démontre par récurrence sur la structure de la preuve de la formule ϕ (en supposant que cette preuve est effectuée dans un système de déduction intuitionniste dans le style de Hilbert par exemple). Comme les connecteurs de la théorie des ensembles se traduisent par leurs équivalents en théorie des types de même que les règles de déduction correspondantes (dont les formes traduites sont dérivables dans le système $F\omega.3$), il suffit de vérifier que la traduction de chacun des axiomes de la théorie des ensembles de Zermelo est prouvable dans le système $F\omega.3$, ce que nous avons déjà fait dans la section 9.4. Notons que la preuve de la validité des axiomes de compréhension repose sur le lemme de compatibilité établi ci-dessus. □

Remarque 9.5.4 — Dans la figure 9.1, nous avons pris une certaine liberté avec le langage de la théorie des ensembles (qui est une théorie du premier ordre sans symbole de constante ni symbole de fonction) pour exprimer l'axiome de l'infini puisque nous avons utilisé un symbole de constante $\varnothing$ et un symbole de fonction s d'arité 1 pour désigner l'ensemble vide et la fonction successeur respectivement. Bien entendu, il est toujours possible d'exprimer l'axiome de l'infini sans recourir à ces symboles additionels, en utilisant par exemple la formulation suivante

$$\begin{array}{ll} \exists a & \big(\exists x_0 \quad (\forall y \quad \neg\, y \in x_0) \quad \wedge \quad x_0 \in a\big) \quad \wedge \\ & \big(\forall x \quad x \in a \quad \Rightarrow \quad \exists x' \quad (\forall y \quad y \in x' \quad \Leftrightarrow \quad y \in x \ \vee \ y = x) \quad \wedge \quad x' \in a\big) \end{array}$$

dont la traduction dans le système $F\omega.3$ est une proposition prouvable (ce qui se vérifie immédiatement à l'aide des propositions 9.4.18, 9.4.20 et 9.4.27).

Nous pouvons maintenant énoncer un premier résultat de cohérence relative :

Corollaire 9.5.5 (Cohérence relative) — *Si le système $F\omega.3$ est cohérent, alors la théorie des ensembles de Zermelo intuitionniste est cohérente également :*

$$\mathrm{IZ} \quad \leq \quad F\omega.3$$

Preuve. Il suffit de remarquer que toute preuve de la formule $\bot$ en théorie des ensembles de Zermelo intuitionniste se traduit par la correspondance ci-dessus en une preuve de la proposition $\bot^\dagger \equiv \bot$ dans le système $F\omega.3$. □

Nous verrons dans les deux dernières sections de ce chapitre qu'il est possible de renforcer ce résultat, d'abord en remplaçant IZ par Z (*i.e.* en passant à la théorie des ensembles de Zermelo classique), mais aussi en montrant que l'inégalité ci-dessus est en réalité une inégalité stricte (c'est-à-dire qu'il est possible d'exprimer et de prouver la cohérence logique de la théorie des ensembles de Zermelo à l'intérieur du système $F\omega.3$).

9.5.2 Le type des ensembles

Bien que la traduction décrite au paragraphe précédent soit suffisante pour nous permettre d'établir le résultat de cohérence relative $\mathrm{IZ} \leq F\omega.3$, cette traduction est difficilement utilisable en pratique puisqu'elle nécessite de représenter chaque ensemble par trois paramètres. Dans ce paragraphe, nous nous proposons de transposer dans le système $F\omega.3$ le codage utilisé au paragraphe 8.4.3 (effectué à l'origine dans une théorie incohérente) afin de construire un type U dont les habitants sont susceptibles de représenter les ensembles.

Comme dans le chapitre précédent, le type U est défini à partir d'une représentation approximative du type somme dépendante $\Sigma X : \mathsf{Type}_2 . (X \to X \to \mathsf{Prop}) \times X$ en posant :

$$\mathsf{U} \quad : \quad \mathsf{Type}_3 \quad := \quad \big(\Pi X : \mathsf{Type}_2 . (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \to \mathsf{Prop}.$$

Notons que dans le système $F\omega.3$, la sorte du type U (ici la sorte Type_3) se situe un niveau au-dessus de la sorte dans laquelle sont définis les types de base des graphes pointés (*i.e.* la sorte Type_2). Il n'est donc plus possible d'utiliser le type U comme un type de base de graphe pointé comme nous avons pu le faire dans le chapitre précédent afin d'obtenir un paradoxe.

La fonction i qui à chaque graphe pointé (X, A, a) associe son représentant $i(X, A, a)$ dans le type U est définie comme au chapitre précédent par

$$\begin{array}{l} i \quad : \quad \Pi X : \mathsf{Type}_2 . (X \to X \to \mathsf{Prop}) \to X \to \mathsf{U} \quad := \\ \quad \lambda(X, A, a) . \lambda f : \big(\Pi X : \mathsf{Type}_2 . (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) . \quad f(X, A, a), \end{array}$$

et on vérifie de la même façon que cette fonction est injective dans le sens suivant :

Lemme 9.5.6 (Injectivité) — *La fonction i est injective, en ce sens que si deux graphes pointés (X, A, a) et (Y, B, b) ont la même image par i dans le type U, alors les graphes pointés (X, A, a) et (Y, B, b) sont bissimilaires :*

$$\forall (X, A, a), (Y, B, b). \quad i(X, A, a) =_{\mathsf{U}} i(Y, B, b) \quad \Rightarrow \quad (X, A, a) \approx (Y, B, b).$$

Preuve. La preuve s'effectue de la même manière que pour le lemme 8.4.5. □

Comme la fonction i n'est pas surjective (le contre-exemple donné au paragraphe 8.4.3 s'adapte sans difficulté au cadre du système $F\omega.3$), il est nécessaire d'introduire un prédicat $\mathsf{set} : \mathsf{U} \to \mathsf{Prop}$ délimitant l'image de la fonction i, lequel est défini par

$$\mathsf{set} \quad : \quad \mathsf{U} \to \mathsf{Prop} \quad := \quad \lambda u : \mathsf{U} \,.\quad \exists (X, A, a) \,.\quad u = i(X, A, a).$$

On appellera *ensemble* tout terme u de type U tel que $\mathsf{set}(u)$, et on utilisera les abréviations suivantes

$$\begin{array}{lcl} \forall u : \mathsf{set} \,.\, \phi & \equiv & \forall u : \mathsf{U} \,.\, \mathsf{set}(u) \Rightarrow \phi \\ \exists u : \mathsf{set} \,.\, \phi & \equiv & \exists u : \mathsf{U} \,.\, \mathsf{set}(u) \wedge \phi \end{array}$$

pour désigner la quantification universelle (resp. la quantification existentielle) de la proposition ϕ par rapport à la variable $u : \mathsf{U}$ restreinte à la classe des ensembles.

Les relations de bissimilarité et d'appartenance (définies à l'origine sur la classe des graphes pointés à l'aide des fonctions EQV et ELT) sont représentées dans le type U par deux relations binaires eqv et elt définies par :

$$\begin{array}{l} \mathsf{eqv} \quad : \quad \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda u, v : \mathsf{U} \,.\quad \exists (X, A, a), (Y, B, b) \,. \\ \qquad u = i(X, A, a) \quad \wedge \quad v = i(Y, B, b) \quad \wedge \quad (X, A, a) \approx (Y, B, b) \end{array}$$

$$\begin{array}{l} \mathsf{elt} \quad : \quad \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda u, v : \mathsf{U} \,.\quad \exists (X, A, a), (Y, B, b) \,. \\ \qquad u = i(X, A, a) \quad \wedge \quad v = i(Y, B, b) \quad \wedge \quad (X, A, a) \in (Y, B, b) \end{array}$$

Par la suite, les relations $\mathsf{eqv}(u, v)$ et $\mathsf{elt}(u, v)$ seront notées $u \approx v$ et $u \in v$ respectivement. D'après cette définition, il est immédiat que chacune des assertions $u \approx v$ et $u \in v$ entraîne que u et v sont tous les deux des ensembles :

Lemme 9.5.7 (Domaine des relations eqv et elt) — *Les relations binaires $u \approx v$ et $u \in v$ sont définies sur la classe des ensembles, c'est-à-dire :*

$$\begin{array}{llll} \forall u, v : \mathsf{U} \,.\quad u \approx v \quad \Rightarrow \quad \mathsf{set}(u) & & \forall u, v : \mathsf{U} \,.\quad u \in v \quad \Rightarrow \quad \mathsf{set}(u) \\ \forall u, v : \mathsf{U} \,.\quad u \approx v \quad \Rightarrow \quad \mathsf{set}(v) & & \forall u, v : \mathsf{U} \,.\quad u \in v \quad \Rightarrow \quad \mathsf{set}(v) \end{array}$$

En utilisant les propositions 9.3.1 et 9.3.2, on montre que la relation $u \approx v$ est une relation d'équivalence partielle sur le type U dont le domaine est la classe des ensembles, et que la relation d'appartenance $u \in v$ est compatible à gauche et à droite vis-à-vis de cette relation d'équivalence partielle. La preuve de transitivité de la relation $u \approx v$ ainsi que les preuves de compatibilité reposent sur l'injectivité de la fonction i établie par le lemme 9.5.6.

Proposition 9.5.8 (Relation d'équivalence partielle) — *La relation binaire $u \approx v$ est une relation d'équivalence partielle définie sur la classe des objets $u : \mathsf{U}$ tels que $\mathsf{set}(u)$:*

$$\begin{array}{lll} \forall u : \mathsf{U} \,. & u \approx u \quad \Leftrightarrow \quad \mathsf{set}(u) & \\ \forall u, v : \mathsf{U} \,. & u \approx v \quad \Rightarrow \quad v \approx u & \\ \forall u, v, w : \mathsf{U} \,. & u \approx v \quad \Rightarrow \quad v \approx w \quad \Rightarrow \quad u \approx w & \end{array}$$

Proposition 9.5.9 (Compatibilité) — *La relation d'appartenance $u \in v$ est compatible à gauche et à droite vis-à-vis de la relation d'équivalence partielle $u \approx v$:*

$$\forall u, v, w : \mathsf{U}\,. \qquad u \approx v \quad \Rightarrow \quad v \in w \quad \Rightarrow \quad u \in w$$
$$\forall u, v, w : \mathsf{U}\,. \qquad u \in v \quad \Rightarrow \quad v \approx w \quad \Rightarrow \quad u \in w$$

9.5.3 Inclusion et prédicats compatibles

La relation d'inclusion Au paragraphe 9.4.3, nous avons défini sur la classe des graphes pointés une relation binaire $\mathsf{SUB}(X, A, a, Y, B, b)$ exprimant le fait que tout graphe pointé appartenant à (X, A, a) (au sens de la relation ELT) appartient également au graphe pointé (Y, B, b). De manière similaire, on définit sur le type U une relation binaire $\mathsf{sub}(u, v)$ en posant

$$\begin{array}{l} \mathsf{sub} \quad : \quad \mathsf{U} \to \mathsf{U} \to \mathsf{Prop} \quad := \\ \quad \lambda u, v : \mathsf{U}\,. \quad \forall w : \mathsf{U}\,. \quad w \in u \quad \Rightarrow \quad w \in v. \end{array}$$

et on vérifie aisément que cette nouvelle traduction de la relation d'inclusion est reliée à l'ancienne de la manière suivante :

Lemme 9.5.10 (Équivalence des relations SUB et sub) — *Quels que soient les graphes pointés (X, A, a) et (Y, B, b), les assertions $\mathsf{SUB}(X, A, a, Y, B, b)$ et $\mathsf{sub}\big(i(X, A, a), i(Y, B, b)\big)$ sont équivalentes :*

$$\forall (X, A, a), (Y, B, b)\,. \quad \mathsf{SUB}(X, A, a, Y, B, b) \quad \Leftrightarrow \quad \mathsf{sub}\big(i(X, A, a), i(Y, B, b)\big).$$

Prédicats compatibles Ainsi que nous l'avons vu au paragraphe 9.4.2, la traduction des axiomes de compréhension nécessite de travailler avec des prédicats compatibles (définis sur la classe des graphes pointés), c'est-à-dire avec des fonctions à trois arguments

$$P \quad : \quad \Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}$$

satisfaisant la proposition $\mathsf{COMPAT}(P)$. De même, on appellera *prédicat compatible* sur le type U tout terme p de type $\mathsf{U} \to \mathsf{Prop}$ tel que $\mathsf{compat}(p)$, où compat est le terme défini par :

$$\begin{array}{l} \mathsf{compat} \quad : \quad (\mathsf{U} \to \mathsf{Prop}) \to \mathsf{Prop} \quad := \\ \quad \lambda p : (\mathsf{U} \to \mathsf{Prop})\,. \quad \forall u, v : \mathsf{U}\,. \quad p(u) \quad \Rightarrow \quad u \approx v \quad \Rightarrow \quad p(v). \end{array}$$

Du point de vue de la compatibilité, il n'y a pas de différence essentielle entre les prédicats définis sur le type U (*i.e.* les termes de type $\mathsf{U} \to \mathsf{Prop}$) et ceux qui sont définis sur la classe des graphes pointés (*i.e.* les termes de type $\Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}$), puisqu'il est possible de passer des premiers aux seconds au moyen de la fonction cvpred définie par

$$\begin{array}{l} \mathsf{cvpred} \quad : \quad (\mathsf{U} \to \mathsf{Prop}) \quad \to \quad \Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop} \quad := \\ \quad \lambda p : (\mathsf{U} \to \mathsf{Prop})\,.\,\lambda (X, A, a)\,.\; p(i(X, A, a)), \end{array}$$

On vérifie alors aisément qu'à travers cette fonction de conversion, les deux notions de compatibilité (définies par les termes COMPAT et compat respectivement) sont équivalentes :

Lemme 9.5.11 — *Pour tout prédicat p de type* $\mathsf{U} \to \mathsf{Prop}$*, les propositions* $\mathsf{compat}(p)$ *et* $\mathsf{COMPAT}(\mathsf{cvpred}(p))$ *sont équivalentes :*

$$\forall p : (\mathsf{U} \to \mathsf{Prop})\,. \qquad \mathsf{compat}(p) \quad \Leftrightarrow \quad \mathsf{COMPAT}(\mathsf{cvpred}(p)).$$

Dans ce qui suit, nous relativiserons la plupart des quantifications du second ordre à la classe des prédicats compatibles, en utilisant pour cela les abréviations

$$\begin{aligned} \forall p : \mathsf{compat}\,.\,\phi \quad &\equiv \quad \forall p : (\mathsf{U} \to \mathsf{Prop})\,.\,\mathsf{compat}(p) \Rightarrow \phi \\ \exists p : \mathsf{compat}\,.\,\phi \quad &\equiv \quad \exists p : (\mathsf{U} \to \mathsf{Prop})\,.\,\mathsf{compat}(p) \wedge \phi. \end{aligned}$$

similaires aux notations $\forall u : \mathsf{set}\,.\,\phi$ et $\exists u : \mathsf{set}\,.\,\phi$ que nous avons introduites dans le cadre de la quantification du premier ordre.

9.5.4 Les axiomes de Zermelo dans le type U

Dans la section précédente, nous avons défini des constructeurs de relations PAIR, FOLD, POWER, UNION, VOID, SUCC et OMEGA à partir desquels nous avons construit des graphes pointés correspondant aux différentes constructions de la théorie des ensembles de Zermelo. Par exemple, nous avons montré au paragraphe 9.4.1 que 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

$$\big(\mathsf{sum}(X, Y), \mathsf{PAIR}(X, A, a, Y, B, b), \mathsf{out}(X, Y)\big)$$

dont la relation binaire est formée en appliquant le constructeur de relation PAIR aux paramètres constitutifs des graphes pointés (X, A, a) et (Y, B, b).

Maintenant que nous disposons d'un type de données U dans lequel nous pouvons représenter n'importe quel graphe pointé, il est possible de transformer ces constructeurs de relations en véritables *constructeurs d'ensembles* qui opèrent directement sur les images (par l'injection i) des graphes pointés dans le type U. Pour cela, on introduit des termes pair, fold, power, union, void, succ et omega dont la définition est donnée dans la figure 9.2.

Si la définition des constantes void et omega est naturelle — il s'agit simplement des images des graphes pointés $\big(\mathsf{unit}, \mathsf{VOID}, \mathsf{id}\big)$ et $\big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big)$ par l'injection i dans le type U — la définition des termes pair, fold, power, union et succ est justifiée par le fait que pour tous graphes pointés (X, A, a), (Y, B, b) et pour tout prédicat $p : \mathsf{U} \to \mathsf{Prop}$ on a les égalités intensionnelles suivantes :

$$\begin{aligned} \mathsf{pair}(i(X, A, a), i(Y, B, b)) \quad &=_\beta \quad i\big(\mathsf{sum}(X, Y), \mathsf{PAIR}(X, A, a, Y, B, b), \mathsf{out}(X, Y)\big) \\ \mathsf{fold}(i(X, A, a), p) \quad &=_\beta \quad i\big(\mathsf{opt}(X), \mathsf{FOLD}(X, A, a, \mathsf{cvpred}(p)), \mathsf{none}(X)\big) \\ \mathsf{power}(i(X, A, a)) \quad &=_\beta \quad i\big(\mathsf{sum}(X, X \to \mathsf{Prop}), \mathsf{POWER}(X, A, a), \mathsf{out}(X, X \to \mathsf{Prop})\big) \\ \mathsf{union}(i(X, A, a)) \quad &=_\beta \quad i\big(\mathsf{opt}(X), \mathsf{UNION}(X, A, a), \mathsf{none}(X)\big) \\ \mathsf{succ}(i(X, A, a)) \quad &=_\beta \quad i\big(\mathsf{opt}(X), \mathsf{SUCC}(X, A, a), \mathsf{none}(X)\big) \end{aligned}$$

De cette remarque, il ressort clairement que si u et v sont des ensembles (c'est à dire des images de graphes pointés par l'injection i), et si $p : \mathsf{U} \to \mathsf{Prop}$ est un prédicat quelconque, alors les termes $\mathsf{pair}(u, v)$, $\mathsf{fold}(u, p)$, $\mathsf{power}(u)$, $\mathsf{union}(u)$ et $\mathsf{succ}(u)$ sont également des ensembles :

$$
\begin{array}{l}
\mathsf{pair} \;:\; \mathsf{U} \to \mathsf{U} \to \mathsf{U} \;:=\; \\
\quad \lambda u, v : \mathsf{U} \,.\, \lambda f : \big(\Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \,. \\
\qquad u\Big(\lambda(X, A, a) \,.\, v\Big(\lambda(Y, B, b) \,.\, f\big(\mathsf{sum}(X, Y),\ \mathsf{pair}(X, A, a, Y, B, b),\ \mathsf{out}(X, Y)\big)\Big)\Big) \\
\\
\mathsf{fold} \;:\; \mathsf{U} \to (\mathsf{U} \to \mathsf{Prop}) \to \mathsf{U} \;:=\; \\
\quad \lambda u : \mathsf{U} \,.\, \lambda p : (\mathsf{U} \to \mathsf{Prop}) \,.\, \lambda f : \big(\Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \,. \\
\qquad u\Big(\lambda(X, A, a) \,.\, f\big(\mathsf{opt}(X),\ \mathsf{FOLD}(X, A, a, \mathsf{cvpred}(P)),\ \mathsf{none}(X)\big)\Big) \\
\\
\mathsf{power} \;:\; \mathsf{U} \to \mathsf{U} \;:=\; \\
\quad \lambda u : \mathsf{U} \,.\, \lambda f : \big(\Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \,. \\
\qquad u\Big(\lambda(X, A, a) \,.\, f\big(\mathsf{sum}(X, X \to \mathsf{Prop}),\ \mathsf{POWER}(X, A, a),\ \mathsf{out}(X, X \to \mathsf{Prop})\big)\Big) \\
\\
\mathsf{union} \;:\; \mathsf{U} \to \mathsf{U} \;:=\; \\
\quad \lambda u : \mathsf{U} \,.\, \lambda f : \big(\Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \,. \\
\qquad u\Big(\lambda(X, A, a) \,.\, f\big(\mathsf{opt}(X),\ \mathsf{UNION}(X, A, a),\ \mathsf{none}(X)\big)\Big) \\
\\
\mathsf{void} \;:\; \mathsf{U} \;:=\; i\big(\mathsf{unit}, \mathsf{VOID}, \mathsf{id}\big) \\
\\
\mathsf{succ} \;:\; \mathsf{U} \to \mathsf{U} \;:=\; \\
\quad \lambda u : \mathsf{U} \,.\, \lambda f : \big(\Pi X : \mathsf{Type}_2 \,.\, (X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}\big) \,. \\
\qquad u\Big(\lambda(X, A, a) \,.\, f\big(\mathsf{opt}(X),\ \mathsf{SUCC}(X, A, a),\ \mathsf{none}(X)\big)\Big) \\
\\
\mathsf{omega} \;:\; \mathsf{U} \;:=\; i\big(\mathsf{opt}(\mathsf{nat}), \mathsf{OMEGA}, \mathsf{none}(\mathsf{nat})\big)
\end{array}
$$

FIG. 9.2 – Définition des constructeurs d'ensembles dans le type U

Lemme 9.5.12 (Axiomes de relativisation) — *Les propositions* (i) *à* (vii) *de la figure 9.3 sont prouvables dans le système* $F\omega.3$.

Plus généralement, le type U et l'injection i nous permettent maintenant de reformuler tous les axiomes de la théorie des ensembles de Zermelo dans le système $F\omega.3$ de manière à ce que chaque ensemble soit représenté par un seul objet de la théorie des types (contrairement à la représentation par les graphes pointés, qui nécessite de représenter chaque ensemble par trois termes). On trouvera dans la figure 9.3 à la page 317 une traduction des axiomes de Zermelo (en forme skolémisée) dans le type U.

Bien entendu, cette nouvelle formulation des axiomes est équivalente à l'ancienne, puisque les constructeurs d'ensembles pair, fold, power, union, void, succ et omega sont définis à partir des constructeurs de relations correspondants, dont nous avons déjà établi les propriétés dans la section 9.4. Il est donc clair que :

Proposition 9.5.13 (Axiomes de Zermelo) — *Les propositions* 1 *à* 15 *de la figure 9.3 sont prouvables dans le système* $F\omega.3$.

Axiomes de relativisation

(i)	$\forall u, v : \mathsf{set}\,.$	$\mathsf{set}(\mathsf{pair}(u, v))$
(ii)	$\forall u : \mathsf{set}\,. \forall p : (\mathsf{U} \to \mathsf{Prop})\,.$	$\mathsf{set}(\mathsf{fold}(u, p))$
(iii)	$\forall u : \mathsf{set}\,.$	$\mathsf{set}(\mathsf{power}(u))$
(iv)	$\forall u : \mathsf{set}\,.$	$\mathsf{set}(\mathsf{union}(u))$
(v)		$\mathsf{set}(\mathsf{void})$
(vi)	$\forall u : \mathsf{set}\,.$	$\mathsf{set}(\mathsf{succ}(u))$
(vii)		$\mathsf{set}(\mathsf{omega})$

Axiomes d'égalité et de compatibilité

1.	$\forall u : \mathsf{set}\,.$	$u \approx u$
2.	$\forall u, v : \mathsf{set}\,.$	$u \approx v \quad \Rightarrow \quad v \approx u$
3.	$\forall u, v, w : \mathsf{set}\,.$	$u \approx v \quad \Rightarrow \quad v \approx w \quad \Rightarrow \quad u \approx w$
4.	$\forall u, v, w : \mathsf{set}\,.$	$u \approx v \quad \Rightarrow \quad v \in w \quad \Rightarrow \quad u \in w$
5.	$\forall u, v, w : \mathsf{set}\,.$	$u \in v \quad \Rightarrow \quad v \approx w \quad \Rightarrow \quad u \in w$

Axiomes de Zermelo

6.	$\forall u, v : \mathsf{set}\,.$	$\big(\forall w : \mathsf{set}\,. \quad w \in u \quad \Leftrightarrow \quad w \in v\big) \quad \Rightarrow \quad u \approx v$
7.	$\forall u, v : \mathsf{set}\,.$	$\forall w : \mathsf{set}\,. \quad w \in \mathsf{pair}(u, v) \quad \Leftrightarrow \quad w \approx u \quad \vee \quad w \approx v$
8.	$\forall u : \mathsf{set}\,. \forall p : \mathsf{compat}\,.$	$\forall w : \mathsf{set}\,. \quad w \in \mathsf{fold}(u, p) \quad \Leftrightarrow \quad w \in u \quad \wedge \quad p(w)$
9.	$\forall u : \mathsf{set}\,.$	$\forall w : \mathsf{set}\,. \quad w \in \mathsf{power}(u) \quad \Leftrightarrow \quad w \subset u$
10.	$\forall u : \mathsf{set}\,.$	$\forall w : \mathsf{set}\,. \quad w \in \mathsf{union}(u) \quad \Leftrightarrow \quad \big(\exists v : \mathsf{set}\,. \quad w \in v \ \wedge \ v \in u\big)$
11.		$\forall w : \mathsf{set}\,. \quad \neg w \in \mathsf{void}$
12.	$\forall u : \mathsf{set}\,.$	$\forall w : \mathsf{set}\,. \quad w \in \mathsf{succ}(u) \quad \Leftrightarrow \quad w \in u \quad \vee \quad w \approx u$
13.		$\mathsf{void} \in \mathsf{omega}$
14.		$\forall w : \mathsf{set}\,. \quad w \in \mathsf{omega} \quad \Rightarrow \quad \mathsf{succ}(w) \in \mathsf{omega}$

Principe de récurrence

15.	$\forall p : \mathsf{compat}\,.$	$p(\mathsf{void}) \quad \Rightarrow$ $\big(\forall u : \mathsf{set}\,. \quad u \in \mathsf{omega} \quad \Rightarrow \quad p(u) \quad \Rightarrow \quad p(\mathsf{succ}(u))\big) \quad \Rightarrow$ $\forall u : \mathsf{set}\,. \quad u \in \mathsf{omega} \quad \Rightarrow \quad p(u)$

FIG. 9.3 – Présentation skolémisée de la théorie des ensembles de Zermelo dans le type U

Cette dernière proposition nous permet d'envisager une nouvelle traduction des propositions de la théorie des ensembles dans les propositions du système $F\omega.3$, dans laquelle chaque variable du langage de la théorie des ensembles est traduite par une unique variable du système $F\omega.3$, dont nous conviendrons qu'elle porte le même nom afin de simplifier l'exposé. Ainsi, on associe à chaque formule de la théorie des ensembles ϕ un terme $\phi^\ddagger$ du système $F\omega.3$ défini par récurrence sur la structure de la formule ϕ par

$$\begin{array}{lcl}
(x = y)^\ddagger & \equiv & \mathsf{eqv}(x, y) \\
(x \in y)^\ddagger & \equiv & \mathsf{elt}(x, y) \\
(\phi \Rightarrow \psi)^\ddagger & \equiv & \phi^\ddagger \Rightarrow \psi^\ddagger \\
(\phi \wedge \psi)^\ddagger & \equiv & \phi^\ddagger \wedge \psi^\ddagger \\
(\phi \vee \psi)^\ddagger & \equiv & \phi^\ddagger \vee \psi^\ddagger \\
(\forall x\ \phi)^\ddagger & \equiv & \forall x : \mathsf{U}\,.\,\mathsf{set}(x) \Rightarrow \phi^\ddagger \\
(\exists x\ \phi)^\ddagger & \equiv & \exists x : \mathsf{U}\,.\,\mathsf{set}(x) \wedge \phi^\ddagger
\end{array}$$

Notons qu'ici, nous ne traduisons que les formules "pures" de la théorie des ensembles, c'est-à-dire les formules qui ne font intervenir aucun symbole de constante ni symbole de fonction, et dont les seuls termes sont les variables. Bien entendu, nous pourrions étendre le langage de la théorie des ensembles avec des symboles de constantes et de fonctions analogues à ceux que nous avons présentés dans la figure 9.2, que nous traduirions dans le système $F\omega.3$ par les termes correspondants. Dans un souci de simplicité, nous nous en tiendrons uniquement à la traduction du langage de base, ce qui suffit amplement à dériver les résultats d'expressivité qui nous intéressent dans ce chapitre.[16]

Lemme 9.5.14 (Correction) — *Si ϕ est une formule de la théorie des ensembles dont les variables libres sont $x_1, \ldots, x_n$, alors :*

1. *$FV(\phi^\ddagger) \equiv \{x_1; \ldots; x_n\}$;*
2. *dans la signature $\Sigma \equiv [x_1 : \mathsf{U}; \ldots; x_n : \mathsf{U}]$, le jugement $\Sigma \vdash \phi^\ddagger : \mathsf{Prop}$ est dérivable.*

Preuve. Par récurrence sur la structure de la formule ϕ.

Proposition 9.5.15 (Validité) — *Soit ϕ une formule de la théorie des ensembles dont les variables sont $x_1, \ldots, x_n$. Si ϕ est prouvable dans la théorie des ensembles de Zermelo intuitionniste, alors la proposition $\phi^\ddagger$ est prouvable dans la signature $\Sigma = [x_1 : \mathsf{U}; \ldots; x_n : \mathsf{U}]$ et sous le contexte logique $\Gamma = [\xi_1 : \mathsf{set}(x_1); \ldots; \xi_n : \mathsf{set}(x_n)]$.*

Preuve. Par récurrence sur la structure de la preuve de la formule ϕ, que l'on suppose effectuée dans un calcul des prédicats intuitionniste dans le style de Hilbert. □

Remarque 9.5.16 — Un inconvénient de la traduction $\phi \mapsto \phi^\ddagger$ (par rapport à la traduction $\phi \mapsto \phi^\dagger$ introduite au paragraphe 9.5.1) est qu'elle nécessite de relativiser chaque quantification intervenant dans la formule ϕ par le prédicat $\mathsf{set}(x)$. Ceci est dû au fait que la fonction i n'est

[16] On remarquera que la skolémisation du *schéma* d'axiomes de compréhension nécessite d'introduire un nouveau symbole de constante pour chaque formule $\phi_{x_1,\ldots,x_n}(x)$ telle que $FV(\phi) \subset \{x_1; \ldots; x_n; x\}$ et *dans laquelle ne figure aucun symbole de constante ni de fonction.* On se reportera à [22] pour une discussion au sujet des problèmes posés par la skolémisation des axiomes de compréhension.

pas surjective, et que les relations $\mathsf{eqv}(u,v)$ et $\mathsf{elt}(u,v)$ ne font du sens que lorsque u et v sont des images de graphes pointés par l'injection i.

Dans un système de types disposant de sommes dépendantes prédicatives de manière primitive (comme par exemple ECC), il est possible de se passer de cette relativisation en définissant le type des ensembles $\mathsf{U} : \mathsf{Type}_3$ par

$$\mathsf{U} \quad := \quad \Sigma X : \mathsf{Type}_2 \,.\, ((X \to X \to \mathsf{Prop}) \times X)$$

et en posant $i(X,A,a) \equiv \langle X, \langle A, a\rangle\rangle : \mathsf{U}$ pour tout graphe pointé (X,A,a) (où $\langle M,N\rangle$ désigne la paire dépendante formée dans le type $\Sigma x : T \,.\, U$ par les termes M et N dont les types respectifs sont T et $U\{x := N\}$). Dans ce cadre, il est facile de vérifier que la fonction i est surjective (en utilisant les fonctions de projection permettant d'éliminer les paires dépendantes), d'où il ressort que tout objet $x : \mathsf{U}$ satisfait la condition $\mathsf{set}(x)$, qui devient donc superflue.

9.6 Le tiers-exclu

9.6.1 Le système $F\omega.3 + \mathsf{cl}$

Dans ce qui précède, nous avons montré que le paradigme "ensemble = graphe pointé" permet non seulement de traduire les énoncés de la théorie des ensembles dans le système $F\omega.3$, mais également de prouver la traduction de chacun des axiomes de la théorie des ensembles de Zermelo. Cependant, comme la logique du système $F\omega.3$ est intuitionniste (le principe du tiers-exclu n'est pas prouvable dans ce système), les traductions que nous avons présentées ne permettent pas de transporter dans le système $F\omega.3$ toutes les preuves de la théorie des ensembles de Zermelo, mais uniquement celles de son fragment intuitionniste IZ.

Pour passer du fragment intuitionniste à la théorie des ensembles de Zermelo complète, il est naturel d'étendre le système $F\omega.3$ en ajoutant à la syntaxe des termes de preuves une nouvelle constante

$$\mathsf{cl} \quad : \quad \forall a : \mathsf{Prop} \,.\, \neg\neg a \Rightarrow a$$

(où $\neg a$ désigne la proposition $a \Rightarrow \bot$, avec $\bot \equiv \forall b : \mathsf{Prop} \,.\, b$) dont le type est une proposition qui est équivalente (de manière intuitionniste) au principe du tiers-exclu. Dans ce qui suit, on notera $F\omega.3 + \mathsf{cl}$ cette extension du système $F\omega.3$.

Dans le système $F\omega.3 + \mathsf{cl}$, tous les schémas de déduction de la logique classique sont maintenant dérivables. Comme toutes les démonstrations que nous avons effectuées jusqu'ici dans $F\omega.3$ restent valables dans $F\omega.3 + \mathsf{cl}$, il est clair que :

Proposition 9.6.1 (Validité) — *Pour toute formule ϕ prouvable dans la théorie des ensembles de Zermelo (classique), ses traductions $\phi^\dagger$ et $\phi^\ddagger$ sont prouvables dans le système $F\omega.3 + \mathsf{cl}$ (dans les signatures et contextes donnés respectivement par les propositions 9.5.3 et 9.5.15).*

Corollaire 9.6.2 (Cohérence) — *Si le système $F\omega.3 + \mathsf{cl}$ est cohérent, alors la théorie des ensembles de Zermelo est cohérente également :*

$$\mathrm{Z} \quad \leq \quad F\omega.3 + \mathsf{cl}.$$

9.6.2 La A-traduction de Coquand-Herbelin

Pour terminer la preuve de cohérence relative $\mathrm{Z} \leq F\omega.3$, il suffit à présent de montrer que les systèmes $F\omega.3$ et $F\omega.3 + \mathsf{cl}$ sont des théories équiconsistantes. Pour cela, nous allons nous appuyer sur la A-traduction de Coquand-Herbelin [19] qui va nous permettre de traduire toutes les preuves de $F\omega.3 + \mathsf{cl}$ dans $F\omega.3$ de manière à transformer toute preuve de la proposition fausse $\bot$ effectuée dans le système $F\omega.3 + \mathsf{cl}$ en une preuve d'une proposition (équivalente à la proposition) fausse dans le système $F\omega.3$.

La A-traduction de Coquand-Herbelin est définie dans une large classe de systèmes de types purs — les *systèmes de types purs non-dépendants* — dans laquelle figure le système $F\omega.3$,[17] et qui est elle-même une sous-classe de la classe des *PTS logiques* (en reprenant la terminologie de [19]).

Définition 9.6.3 (PTS logique) — On appelle *PTS logique* tout PTS L muni de deux sortes particulières notées Prop et Type tel que :

- L est un PTS fonctionnel [26] ;
- $\mathsf{Prop} : \mathsf{Type}$ est un axiome de L ;
- $(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop})$ est une règle de L ;
- il n'existe dans L aucune sorte de type Prop.

Les systèmes du cube de Barendregt sont tous des PTS logiques, de même que les systèmes (incohérents) U et U^-. Le système $F\omega.3$ a également une structure de PTS logique, dans la mesure où l'on pose $\mathsf{Type} \equiv \mathsf{Type}_1$.

Définition 9.6.4 (PTS non-dépendant) — On appelle *PTS non-dépendant* tout PTS logique L dans lequel les seules règles de formation de produits dépendants où figure la sorte Prop sont de la forme $(s, \mathsf{Prop}, \mathsf{Prop})$, où s est une sorte quelconque de L.

Les systèmes du cube de Barendregt sans types dépendants (*i.e.* la face gauche du cube) sont tous des systèmes non-dépendants, de même que les systèmes U, U^- ou le système $F\omega.3$. En revanche, le Calcul des Constructions n'est pas un PTS non-dépendant, puisqu'il dispose d'une règle de formation de produits dépendants $(\mathsf{Prop}, \mathsf{Type}, \mathsf{Type})$ qui permet de construire des types dépendant de termes de preuves.

L'intérêt de cette notion est que dans la classe de PTS qu'elle définit, les types — qu'il s'agisse de propositions ou non — ne dépendent pas des termes de preuves, lesquels n'interviennent donc jamais dans le test de β-conversion. Ce dernier point est important car la A-traduction (ainsi que les non-non traductions plus généralement) ne préserve pas les coupures dans les termes de preuves.

Dans les deux paragraphes qui suivent, nous définissons successivement la A-traduction de Coquand-Herbelin sur les termes fonctionnels puis sur les termes de preuves du système $F\omega.3$. On remarquera que la présentation stratifiée du système simplifie grandement la définition de

[17] Ici, l'expression "système de type pur" est à prendre au sens strict du terme [26]. En particulier, elle ne recouvre ni les systèmes disposant de types sommes dépendantes (tels que ECC), ni les systèmes disposant d'une notion de cumulativité (tels que ECC ou CCω). C'est pour cette raison que dans notre présentation du système $F\omega.3$, nous avons volontairement omis de faire figurer une règle de cumulativité entre les univers prédicatifs qu'il serait pourtant très naturel de considérer.

la A-traduction dans la couche prédicative du formalisme, puisque cette définition s'effectue simplement par récurrence structurelle sur les termes sans utiliser les jugements de typage correspondants.

9.6.3 Traduction des termes fonctionnels

Comme son nom l'indique, la A-traduction est paramétrée par une proposition A. Soit A un terme clos de type Prop dans la signature vide. Pour tout terme M, on pose

$$\Diamond M \quad \equiv \quad (M \Rightarrow A) \Rightarrow A,$$

sachant que cette notation n'a un sens que dans le cas où M est une proposition. À chaque terme (non-typé) on associe un terme M^+ défini par récurrence sur la structure de M par :

$$\begin{array}{lcl}
x^+ & \equiv & x \\
\mathsf{Prop}^+ & \equiv & \mathsf{Prop} \\
\mathsf{Type}_i^+ & \equiv & \mathsf{Type}_i \\
(\Pi x:T\,.\,U)^+ & \equiv & \Pi x:T^+\,.\,U^+ \\
(\lambda x:T\,.\,M)^+ & \equiv & \lambda x:T^+\,.\,M^+ \\
(M\ N)^+ & \equiv & M^+\ N^+ \\
(\phi \Rightarrow \psi)^+ & \equiv & \Diamond\phi^+ \Rightarrow \Diamond\psi^+ \\
(\forall x:T\,.\,\phi)^+ & \equiv & \forall x:T^+\,.\,\Diamond\phi^+
\end{array}$$

On remarquera que cette traduction s'appuie sur la distinction syntaxique effectuée entre les produits dépendants $\Pi x:T\,.\,U$ d'une part, et l'implication $\phi \Rightarrow \psi$ et la quantification universelle $\forall x:T\,.\,\phi$ d'autre part. La définition ci-dessus est étendue de manière naturelle aux signatures en posant :

$$\begin{array}{lcl}
[]^+ & \equiv & [] \\
(\Sigma; [x:T])^+ & \equiv & \Sigma^+; [x:T^+]
\end{array}$$

La traduction $M \mapsto M^+$ a de bonnes propriétés, puisqu'elle préserve les substitutions, les séquences de réduction, les égalités intensionnelles, et finalement les jugements de typage :

Lemme 9.6.5 (Substitutivité) — *Si M et N sont des termes, et x une variable libre, alors :*

$$(M\{x := N\})^+ \quad \equiv \quad M^+\{x := N^+\}.$$

Preuve. Par récurrence sur la structure de M. □

Lemme 9.6.6 (β-réduction et β-conversion) — *Soient M_1 et M_2 deux termes fonctionnels. Si $M_1 \to_\beta M_2$ (resp. $M_1 =_\beta M_2$), alors $M_1^+ \to_\beta M_2^+$ (resp. $M_1^+ =_\beta M_2^+$).*

Preuve. On vérifie que le résultat est vrai sur un β-radical

$$((\lambda x:T\,.\,M)N)^+ \quad \equiv \quad (\lambda x:T^+\,.\,M^+)N^+ \quad \to_\beta \quad M^+\{x := N^+\} \quad \equiv \quad (M\{x := N\})^+$$

et on conclut par passage au contexte que $M_1 \to_\beta M_2$ entraîne $M_1^+ \to_\beta M_2^+$. La traduction de l'égalité intensionnelle $M_1 =_\beta M_2$ s'en déduit par une récurrence immédiate. □

Proposition 9.6.7 (Validité) — *Soient Σ une signature, M et T des termes. On a les implications suivantes :*

1. *si* $\Sigma \vdash$, *alors* $\Sigma^+ \vdash$;
2. *si* $\Sigma \vdash M : T$, *alors* $\Sigma^+ \vdash M^+ : T^+$.

Preuve. Par récurrence mutuelle sur les dérivations des jugements $\Sigma \vdash$ et $\Sigma \vdash M : T$. On utilise le lemme précédent pour traiter le cas de la règle de conversion. □

9.6.4 Traduction des termes de preuves

Pour tout terme M, on note M^* le terme défini par $M^* \equiv \Diamond M^+$, sachant que cette écriture n'a un sens que dans le cas où M est une proposition. La notation M^* est étendue aux contextes logiques en posant

$$\begin{array}{lcl} []^* & \equiv & [] \\ (\Gamma; [\xi : \phi])^* & \equiv & \Gamma^*; [\xi : \phi^*] \end{array}$$

Il est immédiat que les images par $M \mapsto M^*$ de deux termes β-convertibles sont des termes β-convertibles (y compris si les termes en question sont mal typés ou ne sont pas des propositions). On vérifie alors aisément que la traduction $\Gamma \mapsto \Gamma^*$ transforme chaque contexte logique bien formé en un autre contexte logique bien formé :

Lemme 9.6.8 (Correction)
1. *si* $\Sigma \vdash \phi : \mathsf{Prop}$, *alors* $\Sigma^+ \vdash \phi^* : \mathsf{Prop}$;
2. *si* $\langle\Sigma\rangle\Gamma \vdash$, *alors* $\langle\Sigma^+\rangle\Gamma^* \vdash$.

Preuve. Le premier point est immédiat, et le second point se déduit du premier par une récurrence immédiate. □

À toute preuve t d'une proposition ϕ, nous allons maintenant associer une preuve t^* de la proposition ϕ^*. Contrairement aux traductions $M \mapsto M^+$ et $\phi \mapsto \phi^*$, la A-traduction des termes de preuves est une traduction typée, qui ne fait de sens que dans une signature Σ et un contexte logique Γ donnés. En toute rigueur, on devrait donc écrire $(\langle\Sigma\rangle\Gamma \vdash t)^*$ pour désigner la A-traduction de la preuve t dans le contexte $\langle\Sigma\rangle\Gamma$, sachant que cette écriture n'est définie que lorsque le terme t est un terme de preuve bien formé dans le contexte $\langle\Sigma\rangle\Gamma$ (c'est-à-dire s'il existe une proposition ϕ tel que le jugement $\langle\Sigma\rangle\Gamma \vdash t : \phi$ est dérivable).

Dans une signature Σ et un contexte Γ donnés, le terme t^* est défini par récurrence sur la structure du terme t à l'aide des lemmes d'inversion standards de la théorie des PTS [26] :

$$\begin{array}{lcll} \xi^* & \equiv & \lambda k^{\phi^+ \Rightarrow A}.\, \xi\ k & \text{avec} \quad (\xi : \phi) \in \Gamma \\ (\lambda \xi^{\phi}.\, t)^* & \equiv & \lambda k^{(\phi^* \Rightarrow \psi^*) \Rightarrow A}.\, k\ (\lambda \xi^{\phi^*}.\, \lambda k'^{\psi^+ \Rightarrow A}.\, t^*\ k') & \text{avec} \quad \langle\Sigma\rangle\Gamma; [\xi : \phi] \vdash t : \psi \\ (t\ u)^* & \equiv & \lambda k^{\psi^+ \Rightarrow A}.\, t^*\ (\lambda f^{\phi^* \Rightarrow \psi^*}.\, f\ u^*\ k) & \begin{array}{l}\text{avec} \quad \langle\Sigma\rangle\Gamma \vdash t : \phi \Rightarrow \psi \\ \text{et} \quad \langle\Sigma\rangle\Gamma \vdash u : \phi\end{array} \\ (\lambda x : T\,.\, t)^* & \equiv & \lambda k^{(\forall x : T^+\,.\, \phi^*) \Rightarrow A}.\, k\ (\lambda x : T^+\,.\, \lambda k'^{\phi^+ \Rightarrow A}.\, t^*\ k') & \text{avec} \quad \langle\Sigma; [x : T]\rangle\Gamma \vdash t : \phi \\ (t\ M)^* & \equiv & \lambda k^{\phi^+\{x := M^+\} \Rightarrow A}.\, t^*\ (\lambda f^{\forall x : T^+\,.\, \phi^*}.\, f\ M^+\ k) & \begin{array}{l}\text{avec} \quad \langle\Sigma\rangle\Gamma \vdash t : \forall x : T\,.\, \phi \\ \text{et} \quad \Sigma \vdash M : T\end{array} \end{array}$$

Proposition 9.6.9 (Validité) — *Si* $\langle\Sigma\rangle\Gamma \vdash t : \phi$, *alors* $\langle\Sigma^+\rangle\Gamma^* \vdash t^* : \phi^*$.

Preuve. Par récurrence sur la structure du terme t, en utilisant à chaque étape le lemme d'inversion correspondant. □

9.6.5 Traduction de la constante cl

Dans ce paragraphe, on se place dans le cas particulier où A désigne la proposition fausse, c'est-à-dire $A \equiv \bot$. Dans ce cadre, la A-traduction de Coquand-Herbelin devient une non-non traduction, puisque pour toute proposition ϕ on a

$$\Diamond\phi \quad \equiv \quad (\phi \Rightarrow \bot) \Rightarrow \bot \quad \equiv \quad \neg\neg\phi.$$

Par la correspondance $\phi \mapsto \phi^*$, la proposition fausse $\bot$ se traduit en la proposition

$$\bot^* \quad \equiv \quad (\forall a : \mathsf{Prop}\,.\,a)^* \quad \equiv \quad \neg\neg\forall a : \mathsf{Prop}\,.\,\neg\neg a,$$

dont on montre aisément qu'elle est équivalente à la proposition fausse $\bot$ dans le système $F\omega.3$. Par ailleurs, la proposition $\forall a : \mathsf{Prop}\,.\,\neg\neg a \Rightarrow a$ se traduit en

$$(\forall a : \mathsf{Prop}\,.\,\neg\neg a \Rightarrow a)^* \quad \equiv \quad \neg\neg\big(\forall a : \mathsf{Prop}\,.\,\neg\neg(\neg\neg(\neg\neg a \Rightarrow \bot^*) \Rightarrow \bot^*) \Rightarrow \neg\neg a\big),$$

qui est une proposition prouvable dans le système $F\omega.3$.

Grâce à cette remarque, il est possible d'étendre la correspondance $\phi \mapsto \phi^*$ à tous les termes de preuves du système $F\omega.3 + \mathsf{cl}$ en posant $\mathsf{cl}^* \equiv q$, où q désigne une preuve de la proposition $(\forall a : \mathsf{Prop}\,.\,\neg\neg a \Rightarrow a)^*$ dans le système $F\omega.3$. De cette manière, la correspondance $\phi \mapsto \phi^*$ transforme à présent n'importe quel terme de preuve du système $F\omega.3 + \mathsf{cl}$ en un terme de preuve du système $F\omega.3$, et on a trivialement la propriété de validité suivante :

Proposition 9.6.10 (Validité de la non-non traduction) — *Pour tous Σ, Γ, ϕ et t définis dans le système $F\omega.3 + \mathsf{cl}$, on a :*

$$\langle\Sigma\rangle\Gamma \vdash_{F\omega.3+\mathsf{cl}} t : \phi \qquad \Rightarrow \qquad \langle\Sigma^+\rangle\Gamma^* \vdash_{F\omega.3} t^* : \phi^*.$$

De ceci découle immédiatement le résultat de cohérence relative suivant :

Lemme 9.6.11 (Équiconsistance) — *Les systèmes $F\omega.3$ et $F\omega.3 + \mathsf{cl}$ sont des théories équiconsistantes :*

$$F\omega.3 + \mathsf{cl} \quad \approx \quad F\omega.3.$$

Preuve. Immédiat d'après la proposition précédente et le fait que les propositions $\bot$ et $\bot^*$ sont logiquement équivalentes dans le système $F\omega.3$. □

En combinant ce dernier résultat avec la proposition 9.6.2, nous pouvons donc conclure que le système $F\omega.3$ est une théorie mathématique au moins aussi forte que la théorie des ensembles de Zermelo :

Proposition 9.6.12 (Cohérence relative) — *Si le système $F\omega.3$ (intuitionniste) est cohérent, alors la théorie des ensembles de Zermelo (classique) est cohérente également :*

$$\mathrm{Z} \quad \leq \quad F\omega.3.$$

9.7 Conclusion

Dans la section précédente, nous avons montré la cohérence relative de la théorie des ensembles de Zermelo par rapport à celle de la théorie des types de Church avec deux univers. Par ailleurs, il est clair que dans CCω il est possible de construire un modèle de $F\omega.3$, et de démontrer la cohérence de cette théorie (ce que nous ne ferons pas ici). De ceci, il résulte que le Calcul des Constructions avec univers a une puissance théorique qui dépasse celle de la théorie des ensembles de Zermelo classique, ce qui répond à la conjecture posée dans [17] :

Proposition 9.7.1 (Cohérence relative) — *Dans le Calcul des Constructions avec univers, il est possible d'exprimer et de démontrer la cohérence de la théorie des ensembles de Zermelo :*

$$\mathrm{Z} \quad < \quad \mathrm{CC}\omega.$$

Preuve. Résulte des inégalités $\mathrm{Z} \leq F\omega.3$ et $F\omega.3 < \mathrm{CC}\omega$. □

Dans le paragraphe qui suit, nous allons donner une inégalité stricte plus fine, en montrant que le système $F\omega.3$ est en réalité strictement plus fort que la théorie des ensembles de Zermelo.

9.7.1 La puissance théorique de $F\omega.3$

Au paragraphe 9.4.2, nous avons remarqué que la traduction du schéma d'axiomes de compréhension de la théorie des ensembles de Zermelo s'exprimait à l'aide d'une seule proposition du système $F\omega.3$, ce qui semble indiquer que le système $F\omega.3$ ne contient pas seulement la théorie des ensembles de Zermelo, mais la théorie des ensembles de Zermelo du second ordre.[18] En fait, il est facile de se convaincre que le système $F\omega.3$ contient en réalité la théorie des ensembles de Zermelo d'ordre supérieur.

Nous adoptons ici une formulation très expressive de la théorie des ensembles de Zermelo d'ordre supérieur, qui est basée sur une variante de la théorie des types simples de Church dans laquelle le type de base ι désigne le type des ensembles. Formellement, cette théorie est construite autour d'un λ-calcul simplement typé dont les seuls types de base sont les types o (le type des propositions) et ι (le type des ensembles).

Les termes du formalisme sont les termes du λ-calcul simplement typé auxquels on a ajouté des constantes $\Rightarrow$ (de type $o \to o \to o$) et $\forall^\tau$ (de type $(\tau \to o) \to o$) pour représenter l'implication et la quantification universelle (sur chaque type τ), ainsi que deux constantes $=$ et $\in$ de type $\iota \to \iota \to o$ pour représenter les relations d'égalité et d'appartenance.

La partie logique du formalisme est basée sur un système de déduction à la Hilbert (en logique classique) muni de tous les axiomes de la théorie des ensembles de Zermelo. Notons que dans ce cadre, les axiomes de compréhension sont remplacés par un unique axiome de compréhension exprimé à l'aide de la quantification universelle du second ordre

$$\forall x : \iota\,. \quad \forall p : \iota \to o\,. \quad \exists y : \iota\,. \quad \forall z : \iota\,. \qquad z \in y \quad \Leftrightarrow \quad z \in x \quad \wedge \quad p(z)$$

et dans lequel le prédicat $p : \iota \to o$ peut faire référence à n'importe quel objet dans la hiérarchie des types simples (axiome de compréhension imprédicatif).

[18] C'est-à-dire la théorie de Morse-Kelley [49] privée du schéma d'axiomes de remplacement.

Proposition 9.7.2 (Cohérence relative) — *Si le système $F\omega.3$ est cohérent, alors la théorie des ensembles de Zermelo d'ordre supérieur est cohérente également :*

$$\mathrm{Z}\omega \quad \leq \quad F\omega.3$$

Le schéma de la preuve est le suivant. À chaque type simple τ on associe un type U_τ défini dans le système $F\omega.3$ par

$$\mathsf{U}_o \;\equiv\; \mathsf{Prop}; \qquad \mathsf{U}_\iota \;\equiv\; \mathsf{U} \qquad \text{et} \qquad \mathsf{U}_{\tau\to\sigma} \;\equiv\; \mathsf{U}_\tau \to \mathsf{U}_\sigma$$

(où $\mathsf{U} : \mathsf{Type}_3$ est le type des ensembles qui a été introduit au paragraphe 9.5.2), et on vérifie que chaque type U_τ est un habitant de la sorte Type_1 (si τ ne contient aucune occurrence du type de base ι) ou de la sorte Type_3 (si τ contient au moins une occurrence du type ι). Chacun des types U_τ est muni d'une relation binaire $\mathsf{eqv}_\tau : \mathsf{U}_\tau \to \mathsf{U}_\tau \to \mathsf{Prop}$ définie par

$$\begin{array}{lcll}
\mathsf{eqv}_o & \equiv & \lambda a, b : \mathsf{U}_o \,.\quad a \Leftrightarrow b & \\
\mathsf{eqv}_\iota & \equiv & \mathsf{eqv} & (cf \text{ paragraphe } 9.5.2) \\
\mathsf{eqv}_{\tau\to\sigma} & \equiv & \lambda f, g : \mathsf{U}_{\tau\to\sigma} \,.\quad \forall x, y : \mathsf{U}_\tau \,.\quad \mathsf{eqv}_\tau(x, y) \;\Rightarrow\; \mathsf{eqv}_\sigma(f(x), g(y)). &
\end{array}$$

La relation binaire eqv_τ est clairement une relation d'équivalence partielle sur le type U_τ, et son domaine est donné par le prédicat $\mathsf{wf}_\tau : \mathsf{U}_\tau \to \mathsf{Prop}$ défini par

$$\mathsf{wf}_\tau \quad \equiv \quad \lambda x : \mathsf{U}_\tau \,.\quad \mathsf{eqv}_\tau(x, x),$$

(lequel va nous servir plus loin à relativiser toutes nos quantifications sur le type U_τ).[19]

Chaque terme $M : \tau$ du λ-calcul simplement typé est interprété par un terme $|M| : \mathsf{U}_\tau$ dans le système $F\omega.3$. La correspondance $M \mapsto |M|$ est définie de manière standard, en interprétant les variables par elles-mêmes, l'abstraction par l'abstraction et l'application par l'application, chaque annotation de type $x : \tau$ étant bien entendu remplacée par une annotation de la forme $x : \mathsf{U}_\tau$. Les constantes primitives sont quant à elles interprétées par :

$$\begin{array}{lcll}
|\Rightarrow| & \equiv & \lambda a, b : \mathsf{Prop} \,.\, a \Rightarrow b & \\
|\forall^\tau| & \equiv & \lambda p : (\mathsf{U}_\tau \to \mathsf{Prop}) \,.\quad \forall x : \mathsf{U}_\tau \,.\quad \mathsf{wf}_\tau(x) \;\Rightarrow\; p(x) & \\
|=| & \equiv & \mathsf{eqv} & \\
|\in| & \equiv & \mathsf{elt} & (cf \text{ paragraphe } 9.5.2)
\end{array}$$

On démontre alors le lemme de correction suivant :

Lemme 9.7.3 (Correction) — *Soit $M : \tau$ un terme du λ-calcul simplement typé dont les variables libres sont $x_1 : \sigma_1, \ldots, x_n : \sigma_n$. Alors :*

1. *le jugement $x_1 : \mathsf{U}_{\sigma_1}; \ldots; x_n : \mathsf{U}_{\sigma_n} \vdash |M| : \mathsf{U}_\tau$ est dérivable ;*
2. *la proposition $\mathsf{wf}_{\sigma_1\to\cdots\to\sigma_n\to\tau}\big(\lambda x_1 : \mathsf{U}_{\sigma_1} \ldots \lambda x_n : \mathsf{U}_{\sigma_n} \,.\, |M|\big)$ est prouvable dans le système $F\omega.3$ (dans le contexte vide).*

[19] On remarquera en particulier que les prédicats wf_ι et set définissent la même classe d'objets sur le type U, de même que les prédicats $\mathsf{wf}_{\iota\to o}$ et compat définissent la même classe d'objets sur le type $\mathsf{U} \to \mathsf{Prop}$.

Ce lemme de correction est important, car c'est lui qui permet d'interpréter les règles d'introduction et d'élimination de la quantification universelle dans Zω (qu'on a interprétée par une quantification universelle relativisée dans $F\omega.3$), mais aussi de montrer que les interprétations de tous les termes de type $\iota \to o$ constituent des prédicats compatibles (ce qui est nécessaire pour dériver dans $F\omega.3$ l'axiome de compréhension de Zω). De ceci on tire le résultat de validité suivant :

Lemme 9.7.4 (Validité) — *Soit ϕ un terme de type o (dans Zω) dont les variables libres sont $x_1 : \sigma_1, \ldots, x_n : \sigma_n$. Si ϕ est prouvable dans Zω, alors il existe dans $F\omega.3 + \mathsf{cl}$ un terme de preuve t (dépendant de variables de preuves $\xi_1, \ldots, \xi_n$) tel que le jugement*

$$\langle x_1 : \mathsf{U}_{\sigma_1}; \ldots; x_n : \mathsf{U}_{\sigma_n}\rangle[\xi_1 : \mathsf{wf}_{\sigma_1}(x_1); \ldots; \xi_n : \mathsf{wf}_{\sigma_n}(x_n)] \quad \vdash \quad t \quad : \quad |\phi|$$

soit dérivable dans $F\omega.3 + \mathsf{cl}$.

Ce dernier résultat nous donne immédiatement un algorithme permettant de transformer toute preuve de la proposition $\bot$ dans la théorie des ensembles de Zermelo d'ordre supérieur en une preuve de la proposition $\bot$ dans $F\omega.3 + \mathsf{cl}$, dont nous avons vu au paragraphe 9.6.5 qu'elle est elle-même réductible à une preuve de la proposition $\bot$ dans le système $F\omega.3$. Il est par ailleurs clair que Zω est une théorie strictement plus forte que la théorie des ensembles de Zermelo (*i.e.* Z $<$ Zω), d'où il ressort que :

Corollaire 9.7.5 — *Dans le système $F\omega.3$, il est possible d'exprimer et de démontrer la cohérence de la théorie des ensembles de Zermelo :*

$$\mathrm{Z} \quad < \quad F\omega.3.$$

D'après les différentes remarques effectuées depuis le début de ce chapitre, et notamment celle que nous avons faite au début de la section 9.2 concernant la "minimalité" de la sorte Type_2 vis-à-vis de la construction d'un type infini, il semble que la traduction des formules de Zω dans $F\omega.3$ épuise toute l'expressivité mathématique du système $F\omega.3$. Pour cette raison, nous terminerons ce paragraphe en posant la conjecture suivante :

Conjecture 9.7.6 — *La théorie des ensembles de Zermelo classique d'ordre supérieur et le système $F\omega.3$ sont des théories équiconsistantes :*

$$F\omega.3 \quad \approx \quad \mathrm{Z}\omega.$$

9.7.2 La puissance théorique de Z

Dans le système $F\omega.2$, il n'est plus possible de définir le type U des ensembles, mais il est toujours possible de construire le type $\mathsf{nat} : \mathsf{Type}_2$ des entiers de Church, et d'y exprimer plus généralement toutes les constructions de l'arithmétique d'ordre supérieur (*i.e.* HAω, et même PAω en passant par la non-non traduction de Coquand-Herbelin). Par analogie avec la conjecture précédente, il est raisonnable de penser que $F\omega.2$ et l'arithmétique d'ordre supérieur ont la même puissance théorique :

Conjecture 9.7.7 — *L'arithmétique d'ordre supérieur et le système $F\omega.2$ sont des théories équiconsistantes :*

$$F\omega.2 \quad \approx \quad \mathrm{PA}\omega.$$

Remarque 9.7.8 — En termes de puissance théorique, l'arithmétique d'ordre supérieur[20] a la même force que la théorie des types simples, qui est essentiellement le système $F\omega$ (à travers l'isomorphisme de Curry-Howard) que l'on a étendu en axiomatisant un type nat dans la sorte Type_1. Le résultat ci-dessus semble indiquer que du point de vue de la puissance théorique, l'ajout d'un univers a le même effet que l'ajout d'un type des entiers primitifs dans la sorte Type_1. On retrouve ici une idée standard de la théorie des ensembles, qui est que l'axiome de l'infini équivaut à supposer l'existence d'un ensemble non-vide stable par toutes les "opérations" de la théorie des ensembles (excepté l'existence d'un ensemble infini). Plus généralement, nous conjecturons que $F\omega.(n+1)$ a la même force théorique que $F\omega.n$ dans lequel on a axiomatisé un type des entiers naturels dans la sorte Type_1.

Si les deux conjectures précédentes sont vraies, il est clair que la puissance théorique de la théorie des ensembles de Zermelo se situe entre celle du système $F\omega.2$ et celle du système $F\omega.3$. Il est donc naturel de rechercher entre ces deux systèmes un PTS dont la puissance théorique serait exactement celle de la théorie des ensembles de Zermelo,[21] ce qui motive la définition suivante :

Définition 9.7.9 (λZ) — On note λZ le système de types purs formé à partir des sortes Prop, Type_1, Type_2, Type_3 et dont les axiomes et règles de formation de produits dépendants sont donnés par :

$$\begin{aligned}
\textbf{Axiom} &= \{\,(\mathsf{Prop} : \mathsf{Type}_1);\ (\mathsf{Type}_1 : \mathsf{Type}_2);\ (\mathsf{Type}_2 : \mathsf{Type}_3)\,\}\\
\textbf{Rule} &= \{\,(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop});\ (\mathsf{Type}_1, \mathsf{Prop}, \mathsf{Prop});\\
&\quad\ (\mathsf{Type}_2, \mathsf{Prop}, \mathsf{Prop});\ (\mathsf{Type}_3, \mathsf{Prop}, \mathsf{Prop});\\
&\quad\ (\mathsf{Type}_1, \mathsf{Type}_1, \mathsf{Type}_1);\ (\mathsf{Type}_1, \mathsf{Type}_2, \mathsf{Type}_2);\\
&\quad\ (\mathsf{Type}_2, \mathsf{Type}_1, \mathsf{Type}_2);\ (\mathsf{Type}_2, \mathsf{Type}_2, \mathsf{Type}_2)\,\}
\end{aligned}$$

D'après la définition ci-dessus, on a clairement l'inclusion de PTS

$$F\omega.2 \quad\subset\quad \lambda Z \quad\subset\quad F\omega.3.$$

Plus précisément, λZ est le PTS obtenu en ajoutant à $F\omega.2$ l'axiome $\mathsf{Type}_2 : \mathsf{Type}_3$ et la règle $(\mathsf{Type}_3, \mathsf{Prop}, \mathsf{Prop})$ tout en conservant les mêmes règles de formation de produits dépendants prédicatifs que dans le système $F\omega.2$. Comme dans λZ il n'existe aucune règle de la forme $(s_1, s_2, \mathsf{Type}_3)$, la sorte Type_2 est le seul habitant de la sorte Type_3.[22] Concrètement, le passage de $F\omega.2$ à λZ consiste à ajouter une quantification universelle $\forall X : \mathsf{Type}_2\,.\,\phi$ sur tous les types de la sorte Type_2.[23]

Dans le système λZ, il est toujours possible de parler des graphes pointés (X, A, a) dont le type de base X est dans la sorte Type_2, et d'exprimer les quantifications $\forall(X, A, a)\,.\,\phi$ et

[20] Ici, nous ne faisons pas de différence entre l'arithmétique d'ordre supérieur classique (PAω) et son fragment intuitionniste (HAω), puisque ces deux théories sont équiconsistantes.

[21] Notons que la théorie des ensembles de Zermelo est plus "proche" de l'arithmétique d'ordre supérieur que de la théorie des ensembles de Zermelo d'ordre supérieur : le passage de PAω à Z consiste essentiellement à ajouter une quantification universelle sur tous les ordres, qui correspond à la quantification non-bornée de la théorie des ensembles.

[22] De la même façon que dans le système F (resp. les systèmes U et U^-), Prop est le seul habitant de la sorte Type (resp. Type est le seul habitant de la sorte Kind).

[23] Il est facile de vérifier que dans λZ, la sorte Type_2 n'apparaît que dans le jugement $\Gamma \vdash \mathsf{Type}_2 : \mathsf{Type}_3$ et dans les produits dépendants $\Pi X : \mathsf{Type}_2\,.\,\phi$, qui sont formés à l'aide de la règle $(\mathsf{Type}_3, \mathsf{Prop}, \mathsf{Prop})$.

$\exists(X, A, a)\,.\,\phi$ introduites au début de la section 9.3. Bien que les termes EQV et ELT ne soient plus définissables — leur type ne peut plus être formé dans λZ — il est toujours possible de les remplacer par des abréviations $\mathsf{EQV}(X, A, a, Y, B, b)$ et $\mathsf{ELT}(X, A, a, Y, B, b)$ exprimant la bissimilarité et l'appartenance spécialisées aux graphes pointés (X, A, a) et (Y, B, b). D'après cette remarque, il semble clair que la traduction $\phi \mapsto \phi^\dagger$ introduite au paragraphe 9.5.1 (et dans laquelle chaque variable de la théorie des ensembles est traduite par trois variables de la théorie des types) peut être définie à nouveau dans le système λZ, à quelques changements de notations près.

On notera par ailleurs que dans λZ, il n'existe pas de type des prédicats définis sur la collection des graphes pointés.[24] Par conséquent, chaque instance du schéma de compréhension doit être traduite séparément, de même que la compatibilité des prédicats correspondants (qu'il n'est plus possible d'exprimer par un terme générique) doit être vérifiée au cas par cas. Autrement dit, le schéma d'axiomes de compréhension de la théorie des ensembles de Zermelo reste un schéma de propositions (prouvables) dans le système λZ.

Ces diverses constatations nous laissent penser que le système λZ capture exactement la puissance théorique de la théorie de Zermelo :

Conjecture 9.7.10 — *Le système de types purs λZ et la théorie des ensembles de Zermelo sont des théories équiconsistantes :*

$$\lambda Z \quad \approx \quad \text{Z}.$$

9.7.3 La puissance théorique de CCω

Dans le système $F\omega.4$ (obtenu en ajoutant au système $F\omega.3$ une sorte Type_4 ainsi qu'un axiome et des règles évidentes), il est possible de refaire toutes les constructions de ce chapitre en plaçant le type des entiers de Church et le type des ensembles non pas dans les sortes Type_2 et Type_3, mais dans les sortes Type_3 et Type_4 respectivement, c'est-à-dire en effectuant toutes nos constructions un univers plus haut.

Dans ce système, les types de base des graphes pointés se situent dans la sorte Type_3, c'est-à-dire au même niveau que le type $\mathsf{U} : \mathsf{Type}_3$ que nous avons défini au paragraphe 9.5.2. Dans un tel cadre, nous pourrions non seulement prouver les traductions (en termes de graphes pointés) de tous les axiomes de Zermelo comme précédemment, mais nous pourrions également construire un graphe pointé basé sur le type $\mathsf{opt}(\mathsf{U})$ représentant un ensemble stable par toutes les opérations de la théorie des ensembles de Zermelo (en reprenant toutes les constructions effectuées dans $F\omega.3$). Cette remarque motive la définition suivante :

Définition 9.7.11 (Z-univers) — En théorie des ensembles, on appelle *Z-univers* tout ensemble U satisfaisant les critères suivants :

1. U est un ensemble transitif ;
2. si $x, y \in U$, alors $\{x; y\} \in U$;
3. si $x \in U$, alors $\mathfrak{P}(x) \in \mathsf{U}$;
4. si $x \in U$, alors $(\bigcup x) \in U$;
5. $\omega \in U$.

[24] Puisque le terme $\Pi X : \mathsf{Type}_2\,.\,(X \to X \to \mathsf{Prop}) \to X \to \mathsf{Prop}$ est mal typé dans λZ.

D'après cette définition, il est clair qu'un Z-univers est essentiellement un ensemble transitif stable par toutes les opérations de la théorie des ensembles de Zermelo (la stabilité par le schéma de compréhension s'obtenant à l'aide des points 1 et 3).[25] Dans le système $F\omega.4$, nous pouvons donc non seulement traduire toutes les preuves de la théorie des ensembles de Zermelo, mais également celles de la théorie des ensembles de Zermelo étendue par l'axiome "il existe un Z-univers", que nous noterons $\mathrm{Z}^{1\mathsf{u}}$, et qui est clairement strictement plus forte que la théorie des ensembles de Zermelo.

Ce procédé inspiré par le principe de réflexion est incrémental, et il est raisonnable de penser que chaque sorte prédicative Type_i ajoutée au formalisme (ainsi que l'axiome et les règles qui conviennent) permet de représenter à chaque étape un Z-univers de plus (par un graphe pointé) dont un des éléments est le Z-univers introduit à l'étape précédente.

Soit $F\omega^2$ la théorie des types de Church avec univers (sans entiers primitifs), c'est-à-dire le PTS muni des mêmes sortes et axiomes que CCω, et dont les règles sont $(\mathsf{Prop},\mathsf{Prop},\mathsf{Prop})$, $(\mathsf{Type}_i,\mathsf{Prop},\mathsf{Prop})$, $(\mathsf{Type}_i,\mathsf{Type}_j,\mathsf{Type}_{\max(i,j)})$ (avec $i,j,k>0$). De manière analogue, on note $Z^{(\omega)\mathsf{u}}$ la théorie des ensembles de Zermelo munie d'une infinité de symboles de constantes u_i ($i\in\omega$) et étendue par les axiomes suivants

$$\text{“}\mathsf{u}_i\text{ est un Z-univers”}\qquad\text{et}\qquad\text{“}\mathsf{u}_i\in\mathsf{u}_{i+1}\text{”}\qquad\qquad(i\geq 0).$$

La remarque précédente semble indiquer que toute preuve de $\mathrm{Z}^{(\omega)\mathsf{u}}$ — qui n'utilise qu'un nombre fini de constantes u_i — peut être traduite dans un sous-système $F\omega.n\subset F\omega^2$ (avec n suffisamment grand) de manière à envoyer chaque preuve de la formule $\bot$ (dans $\mathrm{Z}^{(\omega)\mathsf{u}}$) sur une preuve de la proposition $\bot$ (dans $F\omega^2$), en passant bien entendu par la non-non traduction décrite au paragraphe 9.6.5.

Par ailleurs, l'ensemble V_{ω^2} est clairement un modèle de $\mathrm{Z}^{(\omega)\mathsf{u}}$ dans ZF, et un examen attentif des techniques introduites dans [48] — qui permettent d'interpréter les univers de CCω sans recourir à la notion d'ensemble inaccessible[26] — montrent qu'il est possible de construire un modèle de CCω (et sans doute aussi de ECC) dans l'ensemble V_{ω^2} également. Ceci nous laisse donc penser que le système F_{ω^2} et la théorie des ensembles $\mathrm{Z}^{(\omega)\mathsf{u}}$ ont la même puissance théorique, de même que CCω et ECC :

Conjecture 9.7.12 — *La théorie des ensembles de Zermelo avec univers, la théorie des types de Church avec univers, le Calcul des Constructions avec univers et le Calcul des Constructions étendu sont des théories équiconsistantes :*

$$\mathrm{Z}^{(\omega)\mathsf{u}}\quad\approx\quad F\omega^2\quad\approx\quad\mathrm{CC}\omega\quad\approx\quad\mathrm{ECC}.$$

On trouvera dans la figure 9.4 un tableau récapitulatif des différentes conjectures concernant la puissance théorique comparée des PTS imprédicatifs courants et de diverses variantes de la théorie des ensembles de Zermelo (du premier ordre ou d'ordre supérieur).

[25] Un Z-univers est à la théorie des ensembles de Zermelo ce qu'un ensemble inaccessible est à la théorie des ensembles de Zermelo-Fränkel. Pour des raisons évidentes liées au second théorème d'incomplétude de Gödel, il n'est pas possible de montrer l'existence d'un Z-univers dans la théorie des ensembles de Zermelo (à moins que celle-ci ne soit incohérente). Dans ZF en revanche, il est facile de vérifier que pour tout ordinal limite $\lambda\geq 2\omega$, l'ensemble V_λ est un Z-univers.

[26] On remarquera en particulier que les dénotations des univers prédicatifs de CCω telles qu'elles sont définies dans [48] sont à peu de choses près des Z-univers (en faisant abstraction de toutes les constructions liées à l'information de réductibilité).

PTS	**Axiom**	**Rule**	**FO/HO**
F_{ω^2}/CCω/ECC $\uparrow$ ...	...	...	$\mathrm{Z}^{(\omega)\mathsf{u}}$ $\uparrow$ ...
$F\omega.n$ (CC_n)		$(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_n)_{\max(i,j)=n}$	$\mathrm{Z}^{(n-3)\mathsf{u}}\omega$
	$\mathsf{Type}_{n-1} : \mathsf{Type}_n$	$(\mathsf{Type}_n, \mathsf{Prop}, \mathsf{Prop})$	$\mathrm{Z}^{(n-3)\mathsf{u}}$
...	...	...	...
$F\omega.5$ (CC_5)		$(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_5)_{\max(i,j)=5}$	$\mathrm{Z}^{2\mathsf{u}}\omega$
	$\mathsf{Type}_4 : \mathsf{Type}_5$	$(\mathsf{Type}_5, \mathsf{Prop}, \mathsf{Prop})$	$\mathrm{Z}^{2\mathsf{u}}$
$F\omega.4$ (CC_4)		$(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_4)_{\max(i,j)=4}$	$\mathrm{Z}^{1\mathsf{u}}\omega$
	$\mathsf{Type}_3 : \mathsf{Type}_4$	$(\mathsf{Type}_4, \mathsf{Prop}, \mathsf{Prop})$	$\mathrm{Z}^{1\mathsf{u}}$
$F\omega.3$ (CC_3)		$(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_3)_{\max(i,j)=3}$	$\mathrm{Z}\omega$
λZ	$\mathsf{Type}_2 : \mathsf{Type}_3$	$(\mathsf{Type}_3, \mathsf{Prop}, \mathsf{Prop})$	Z
$F\omega.2$ (CC_2)		$(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_2)_{\max(i,j)=2}$	PAω
	$\mathsf{Type}_1 : \mathsf{Type}_2$	$(\mathsf{Type}_2, \mathsf{Prop}, \mathsf{Prop})$	PA (HF)
$F\omega$ ($\mathrm{CC}_\subset$)		$(\mathsf{Type}_1, \mathsf{Type}_1, \mathsf{Type}_1)$	
F	$\mathsf{Prop} : \mathsf{Type}_1$	$(\mathsf{Type}_1, \mathsf{Prop}, \mathsf{Prop})$ $(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop})$	

Dans chacune des lignes du tableau ci-dessus, nous avons établi une correspondance entre un PTS et une variante de la théorie des ensembles de Zermelo du premier ordre (**FO**) ou d'ordre supérieur (**HO**), en conjecturant que les théories mises en regard (dans les première et quatrième colonnes) sont équiconsistantes.

Le tableau se lit du bas vers le haut, et les axiomes et règles données dans les deuxième et troisième colonnes du tableau s'ajoutent les uns aux autres de manière incrémentale.

Dans la colonne de gauche, CC_i désigne le Calcul des Constructions avec i univers, qui correspond au système $F\omega.i$ auquel on a ajouté des règles de cumulativité $\mathsf{Prop} \subset \mathsf{Type}_1 \subset \cdots \subset \mathsf{Type}_{i-1} \subset \mathsf{Type}_i$, et dont nous conjecturons qu'il a la même puissance théorique.

HF désigne la théorie des ensembles héréditairement finis — c'est-à-dire la théorie des ensembles sans axiome de l'infini — qui est équiconsistante à l'arithmétique de Peano [39].

FIG. 9.4 – PTS imprédicatifs et théories de Zermelo

Bibliographie

[1] P. Aczel. The type theoretic interpretation of constructive set theory. In *Logic Colloqium '77*, 1978.

[2] P. Aczel. The type theoretic interpretation of constructive set theory : Choice principles. In *The L.E.J. Brouwer Centenary Symposium*, 1982.

[3] P. Aczel. The type theoretic interpretation of constructive set theory : Inductive definitions. In *Proceedings of the 7th International Congress on Logic, Methodology and Philosophy of Science*, 1986.

[4] P. Aczel. Non well-founded sets. *Center for the Study of Language and Information*, 1988.

[5] P. Aczel. On relating type theories and set theories. In Altenkirch, Naraschewski, and Reus, editors, *Proceedings of Types'98*, 1999.

[6] T. Altenkirch. *Constructions, Inductive types and Strong Normalization*. PhD thesis, University of Edinburgh, 1993.

[7] L. Augustsson. Cayenne – a language with dependent types. In *International Conference of Functional Programming*, 1998.

[8] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics*. North-Holland, 1984.

[9] H. Barendregt. Introduction to generalized type systems. Technical Report 90-8, University of Nijmegen, Department of Informatics, May 1990.

[10] B. Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université Paris VII, 1999.

[11] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997.

[12] G. Barthe and M.H. Søorensen. Domain-free pure type systems. *Journal of functional programming*, 10(5) :412–452, 2000.

[13] L.E.J. Brouwer. Intuïtionistische splitsing van mathematische grondbegrippen. *Nederl. Akad. Wetensch. Verslagen*, 32 :877–880, 1923.

[14] A. Church. A formulation of the theory of simple types. *The Journal of Symbolic Logic*, 5, 1940.

[15] A. Church. The calculi of lambda-conversion, 1941.

[16] T. Coquand. *Une théorie des constructions*. PhD thesis, Université Paris 7, 1985.

[17] T. Coquand. An analysis of Girard's paradox. In *Proceedings, Symposium on Logic in Computer Science (LICS'86)*, pages 227–236. IEEE Computer Society, 1986.

[18] T. Coquand. A new paradox in type theory. In *Proceedings of the 9th International Congress of Logic, Methodology and Philosophy of Science*, 1994.

[19] T. Coquand and H. Herbelin. A-translation and looping combinators in pure type systems. *Journal of Functional Programming*, 4, 1994.

[20] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 120(76) :95, 1988.

[21] G. Dowek. The undecidability of typability in the lambda-pi calculus. In *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 139–145. Springer, 1993.

[22] G. Dowek. A type-free formalization of mathematics where proofs are objects. In E. Giménez and Ch. Paulin-Mohring, editors, *Types for Proofs and Programs*, volume 1512 of *Lecture Notes in Computer Science*, pages 88–111. Springer Verlag, 1998.

[23] G. Frege. *Grundgesetze der Arithmetik, begriffsschriftlich abgeleitet*, volume I. Jena, 1893.

[24] D. Fridlender. A proof-irrelevant model of martin-löf's logical framework. *Mathematical Structures in Computer Science (MSCS)*, 2001.

[25] J. H. Geuvers. The Church-Rosser property for $\beta\eta$-reduction in typed lambda calculi. In IEEE, editor, *Proceedings of the seventh annual symposium on Logic in Computer Science, Santa Cruz, Cal.*, pages 453–460, 1992.

[26] J.H. Geuvers and M.J. Nederhof. A modular proof of strong normalization for the calculus of constructions. In *Journal of Functional Programming*, volume 1,2(1991), pages 155–189, 1991.

[27] P. Giannini, F. Honsell, and S. Ronchi della Rocca. Type inference : some results, some problems. In *Fundamenta Informaticæ*, volume 19(1,2), pages 87–126, 1993.

[28] J.-Y. Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Doctorat d'état, Université Paris VII, Juin 1972.

[29] J.-Y. Girard. The system F of variable types : Fifteen years later. *Theoretical Computer Science*, 45 :159–192, 1986.

[30] J.-Y. Girard. *Locus Solum. Mathematical Structures in Computer Science (MSCS)*, 2001.

[31] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[32] M. Hagiya and Y. Toda. On implicit arguments. Technical report, Department of Information Science, Faculty of Science, University of Tokyo, 1995.

[33] A. Heyting. Mathematische grundlagenforschung. Intuitionismus. Beweisteorie. 1934.

[34] D.J. Howe. The computational behaviour of Girard's paradox. In *Proceedings of the Second Symposium of Logic in Computer Science*, pages 205–214. IEEE, 1987.

[35] A.J. Hurkens. A simplification of Girard's paradox. In *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications (TLCA'95)*, 1995.

[36] P. Jackson. The Nuprl proof development system, version 4.1 reference manual and user's guide. Technical report, Cornell University, 1994.

[37] G. Kahn and G. Plotkin. Concrete domains. *Theoretical Computer Science*, 121(1–2) :187–278, 1993.

[38] A. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35 :58–65, 1932.

[39] J.-L. Krivine. *Théorie des ensembles*. Cassini, 1998.

[40] D. Leivant. Polymorphic type inference. In *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*, pages 88–98, 1983.

[41] X. Leroy and P. Weis. *Le langage Caml*. Dunod, 1999.

[42] Z. Luo. *Computation and Reasoning : A Type Theory for Computer Science*. Oxford University Press, 1994.

[43] Z. Luo and R. Pollack. Lego proof development system : User's manual. Technical Report 92-228, LFCS, 1992.

[44] L. Magnusson. Introduction to ALF — an interactive proof editor. In Uffe H. Engberg, Kim G. Larsen, and Peter D. Mosses, editors, *Proceedings of the 6th Nordic Workshop on Programming Theory* (Aarhus, Denmark, 17–19 October, 1994), number NS-94-6 in Notes Series, page 269, Department of Computer Science, University of Aarhus, December 1994. BRICS. vi+483.

[45] P. Martin-Löf. A theory of types. Manuscrit non publié, 1971.

[46] P. Martin-Löf. An intuitionistic type theory : Predicative part. *Logic Colloquium*, pages 73–118, 1975.

[47] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[48] P.-A. Melliès and B. Werner. A generic normalization proof for pure type systems. In *Proceedings of TYPES'96*, 1997.

[49] E. Mendelson. *Introduction to Mathematical Logic (Third Edition)*. Chapman & Hall, 1987.

[50] A. Miquel. A model for impredicative type systems with universes, intersection types and subtyping. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS'00)*, 2000.

[51] A. Miquel. The Implicit Calculus of Constructions. In Samson Abramsky, editor, *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*, volume 2044 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 2001.

[52] B. Nordström, K. Peterson, and J. Smith. *Programming in Martin-Löf's Type Theory. An Introduction*. Oxford University Press, 1990.

[53] C. Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation à diriger les recherches, Université Claude Bernard Lyon I, December 1996.

[54] R. Pollack. *The Theory of LEGO : A Proof Checker for the Extended Calculus of Constructions*. Phd thesis, University of Edinburgh, 1994.

[55] J.C. Reynolds. Polymorphism is not set-theoretic. *Semantics of Data types*, pages 145–156, 1984.

[56] A. Saïbi. *Algèbre Constructive en Théorie des Types, Outils génériques pour la modélisation et la démonstration, Application à la théorie des Catégories*. PhD thesis, Université Paris VI, 1998.

[57] D. Scott. Data types as lattices. *SIAM Journal on Computing*, 5 :522–587, 1976.

[58] D. Scott. Domains for denotational semantics. In M. Nielson and E. M. Schmidt, editors, *Automata, Languages, and Programming : 9th Colloquium, Aarhus, Denmark*, number 140, pages 577–613. Springer-Verlag, 1982.

[59] W.W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32, 1967.

[60] S. van Bakel, L. Liquori, S. Ronchi della Rocca, and P. Urzyczyn. Comparing Cubes. In A. Nerode and Yu. V. Matiyasevich, editors, *Proceedings of LFCS '94. Third International Symposium on Logical Foundations of Computer Science,* St. Petersburg, Russia, volume 813 of *Lecture Notes in Computer Science*, pages 353–365. Springer-Verlag, 1994.

[61] J.B. Wells. *Type Inference for System F with and without the Eta Rule.* PhD thesis, Boston University, 1996.

[62] J.B. Wells. Typability and type checking in system F are equivalent and undecidable. In *Annals of Pure and Applied Logic*, volume 98(1-3), pages 111–156, 1999.

[63] B. Werner. *Une théorie des Constructions Inductives.* PhD thesis, Université Paris VII, 1994.

[64] B. Werner. Sets in types, types in sets. In *Proceedings of TACS'97*, 1997.

[65] H. Xi. Imperative Programming with Dependent Types. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science*, pages 375–387, Santa Barbara, June 2000.

Table des figures

RÉSUMÉ en français :

Cette thèse a pour principal objet l'étude du Calcul des Constructions implicite, une variante du Calcul des Constructions avec univers dans laquelle les fonctions sont dépourvues de domaine de définition et dont le système de types permet de définir des types par intersection. Dans la première partie, nous établissons les propriétés syntaxiques du calcul et étudions la relation de sous-typage induite par la présence des types intersection.

Dans la deuxième partie, nous étudions les modèles du calcul implicite et montrons que les espaces cohérents permettent d'interpréter la relation de sous-typage du formalisme sans aucune coercition. Après avoir construit un premier modèle classique dans lequel les preuves sont indiscernables, nous construisons ensuite un modèle de réalisabilité en montrant que cette notion s'adapte au cadre des espaces cohérents de telle sorte que l'équivalence de catégories usuelle entre PER et ω-sets modestes est remplacée par une identité. Enfin, nous prouvons le théorème de normalisation forte du calcul implicite, qui entraîne la cohérence logique du système.

Dans la troisième partie, nous définissons une traduction de la théorie des ensembles incohérente de Frege dans le système U^- en représentant les ensembles par des graphes pointés, ce qui nous donne une traduction immédiate du paradoxe de Russell dans ce système. En utilisant les mêmes idées, nous définissons ensuite une traduction de la théorie des ensembles de Zermelo classique d'ordre supérieur dans le système $F\omega$ avec deux univers, et nous montrons que la puissance théorique du Calcul des Constructions avec univers dépasse celle de la théorie de Zermelo, répondant ainsi à une conjecture posée par T. Coquand en 1986.

TITRE en anglais : The Implicit Calculus of Constructions : Syntax and Semantics

MOTS-CLÉS en français : Logique, Lambda-calcul, Types intersection, Calcul des Constructions, Espaces cohérents, Réalisabilité, Normalisation, Hyper-ensembles

RÉSUMÉ en anglais :

The main topic of this thesis is the study of the Implicit Calculus of Constructions, a variant of the Calculus of Constructions with universes in which functions are domain-free and whose type system allows the definition of types by intersection. In the first part, we establish the syntactic properties of the calculus and we study the subtyping relation induced by the presence of intersection types.

In the second part, we study the models of the implicit calculus and show that coherence spaces are well-suited to interpret the subtyping relation of the formalism without any coercions. After building a classical proof-irrelevant model of the implicit calculus, we then build a realisability model and show that this notion can be used in the framework of coherence spaces in such a way that the usual category equivalence between PER and modest ω-sets is replaced by an identity. Finally, we prove the strong normalisation theorem for the implicit calculus, which entails the logical consistency of the system.

In the third part, we define a translation of Frege's inconsistent set theory into system U^- by representing sets as pointed graphs, and we obtain an immediate translation of Russell's paradox in that system. Using the same ideas, we define a translation of higher-order Zermelo classical set theory into system $F\omega$ with two universes and we show that the proof-theoretical strength of the Calculus of Constructions with universes exceeds the one of Zermelo's theory, thus giving an answer to a conjecture posed by T. Coquand in 1986.

MOTS-CLÉS en anglais : Logic, Lambda-calculus, Intersection Types, Calculus of Constructions, Coherence spaces, Realisability, Normalisation, Hypersets

ADRESSE DU LABORATOIRE : Projet LogiCal — INRIA Rocquencourt
Domaine de Voluceau — B.P. 105 — 78153 Le Chesnay Cedex — FRANCE