Une Théorie des Constructions Inductives

Benjamin Werner

THESE DE DOCTORAT

présentée

A L'UNIVERSITE PARIS 7

Spécialité : Informatique Fondamentale

par

Benjamin WERNER

Sujet de la thèse :

Une Théorie

des Constructions Inductives

Soutenue le 2 Mai 1994 devant la Commission d'examen composée de

MM.	**Serge GRIGORIEFF**	**Président**
	Jean GALLIER	**Rapporteurs**
	Jean-Pierre JOUANNAUD	
Mme.	**Christine PAULIN-MOHRING**	**Directeur**
MM.	**Gérard BERRY**	**Examinateurs**
	Gérard HUET	
	Jean-Louis KRIVINE	

Der Einfall also, eine Sache von drei vier Takten, nicht wahr, mehr nicht.
Alles Ubrige ist Elaboration, ist Sitzfleisch. Oder nicht?

Le Diable, dans *Dr. Faustus*, Thomas Mann.

Je remercie chaleureusement Jean Gallier et Jean-Pierre Jouannaud de bien avoir voulu être les rapporteurs de cette thèse. J'ai été particulièrement sensible à leurs encouragements et leurs conseils. Je leur exprime ma gratitude, ainsi qu'à Serge Grigorieff, Gérard Berry et Jean-Louis Krivine qui ont bien voulu faire partie de ce jury.

J'ai eu la très grande chance de faire ce travail de thèse sous la direction de Christine Paulin-Mohring. Sa gentillesse, sa patience et sa rigueur scientifique resteront longtemps un modèle pour moi.

En m'accueillant dans son projet, Gérard Huet m'a permis de découvrir ce qu'était la recherche à travers un domaine passionnant. Grace à sa confiance et son soutien, de nombreuses angoisses et difficultés habituelles du jeune chercheur m'ont été épargnées.

L'environnement de travail dont j'ai profité à l'INRIA a été exceptionnel. Je voudrais remercier tous les chercheurs des projets Coq, Cristal, Para et autres pour les discussions, les idées, l'aide technique et plus généralement les bons moments qu'ils m'ont fait partager.

Un grand merci aussi aux professeurs Constable et Hayashi, ainsi qu'aux membres de leurs équipes, pour m'avoir accueilli avec énormément de gentillesse. Mon passage aux universités de Cornell et Ryukoku a été riche d'enseignements et d'expériences agréables.

Au cours de ces années, j'ai pu côtoyer de nombreux chercheurs, en particulier du BRA Types. J'ai puisé dans ces discussions de nombreuses idées.

Introduction

Le principal objet de ce travail est la présentation et l'étude du Calcul de Constructions Inductives. Il s'agit d'un formalisme qui permet un raisonnement de type mathématique et pouvant être mis en œuvre sur ordinateur. C'est-à-dire qu'il est possible de vérifier *mécaniquement* la validité d'une construction faite à l'intérieur du système.

Les premiers travaux de logique mathématique cherchaient avant tout à définir l'ensemble des règles formelles nécessaire pour permettre des raisonnements non-triviaux. On cite en général Boole, Frege [49] et Peano [126]. La découverte en 1912 par Russell [134] d'un *paradoxe* dans un système logique de Frege a montré la nécessité d'une étude mathématique de ces formalismes eux-mêmes ; on appelle ce domaine la *méta-mathématique*.

Au départ, la logique mathématique ne se voulait certainement pas un domaine appliqué et les préocupations des logiciens souvent plus proches de celles des philosophes que des ingénieurs. L'arrivée de l'ordinateur a changé cet état de fait : la capacité de la machine à manipuler des objets symboliques lui permet de représenter effectivement des preuves formelles, et de vérifier leur validité dans un formalisme bien défini. Au delà de la fascination un peu naïve que peut susciter l'impression de "faire naitre à la vie" et de matérialiser des concepts mathématiques, la vérification de preuves formelles constitue évidemment un enjeu pratique important :

- En vérifiant une preuve mécaniquement, on élimine pratiquement les risques habituels d'erreur. Cette perspective est évidemment séduisante, non seulement en mathématiques, mais dans tous les domaines où la rigueur est importante, et en particulier la validation de programmes.
- Les preuves formelles sont des objets suffisamment bien définis pour que l'on puisse à terme espérer développer des outils qui permettront à l'ordinateur d'aider le mathématicien. C'est le domaine de la démonstration automatique, ou synthèse de preuve.

À plus long terme encore, certains espèrent que ce dévelopement conduira à la mise en place des véritables bases de données de preuves, qui regrouperont l'essentiel du savoir mathématique, pourraient être interfacées avec les descriptions informelles des démonstrations, etc.

Nous sommes toutefois encore loin de ces objectifs ambitieux. La mise en œuvre de systèmes de preuves plus faciles d'utilisation nécessite des efforts technologiques, en particulier de programmation, considérables. Mais la nécessité de savoir produire des preuves formelles a également changé notre manière de voir et de juger les formalismes eux-mêmes : ces-derniers sont maintenant jugés à l'aune de leur facilité d'utilisation.

On compte maintenant un certain nombre d'équipes de recherche de part le monde qui developpent, ou ont developpé, des systèmes permettant la vérification par ordinateur de preuves formelles. Les pionniers ont été les concepteurs du système Automath, autour de N. G. de Bruijn, dans les années 1960 [35]. Aujourd'hui on peut citer le projet NuPRL de R. C. Constable à l'université de Cornell, HOL de M. Gordon et Isabelle de L. Paulson à Cambridge, le système NQTHM de R. Boyer et J. Moore à Austin, OYSTER d'Alan Bundy à Edinburgh, ALF à Göteborg et les

implémentations de variantes du Calcul des Constructions : LEGO à Edinburgh et Coq à l'INRIA et l'ENS-Lyon.

Un point commun remarquable est que les formalismes sous-jacents à tous ces systèmes disposent chacun d'un mécanisme permettant la représentation *intentionnelle* de fonctions. Plus précisément :

- les fonctions ont un statut singulier (contrairement à la théorie des ensembles où elle sont un cas particulier de prédicats)
- une fonction peut être définie par la donnée d'un *algorithme* la calculant.

Le premier formalisme présentant ces caractéristiques a été la logique d'ordre supérieur de Church. Pour permettre la représentation intentionnelle des fonctions, Church a introduit le λ-calcul typé, dont diverses évolutions forment toujours la base des systèmes nommés ci-dessus. La raison essentielle de ce choix semble être la difficulté qu'il y aurait à utiliser un formalisme de type Théorie des Ensembles sur ordinateur : les multiples étapes de codage rendrait le système très lourd à utiliser, à moins d'un énorme travail de l'implémenteur.

De plus, l'utilisation de systèmes de types plus expressifs d'un point de vue algorithmique simplifie les preuves. Un exemple très simple est donné par Dowek [38] : dans un système où l'on peut effectivement définir le comportement calculatoire de l'addition, la preuve d'une proposition comme $2 + 2 = 4$ est véritablement triviale, puisque le mécanisme de calcul est capable d'identifier syntaxiquement les termes $2 + 2$ et 4. Dans des formalismes plus simples, on doit par contre construire une véritable preuve en utilisant les propriétés de l'addition, données sous forme de lemmes ou d'axiomes. Plus le système de types est puissant, plus on pourra identifier de termes ; autant de preuves s'en verront simplifiées.

De plus, on s'est rendu compte par la suite, que la structure des λ-termes typés était isomorphe à la représentation des preuves formelles dans la notation connue sous le nom de *Déduction Naturelle*, due à Dag Prawitz [130]. De même, les types sont alors isomorphes aux propositions. En d'autres termes, le λ-calcul typé permet de représenter non seulement des programmes mais aussi des preuves. On peut donc décrire un formalisme logique complet comme un système de types. Cette correspondance est connue sous le nom d'isomorphisme de Curry-Howard [71]. Cette approche simplifie la méta-théorie : la cohérence logique d'un système se ramène géneralement à la propriété de *normalisation* ou de terminaison des calculs. D'un point de vue plus pratique, le traitement uniforme des preuves et des programmes simplifie bien sûr l'implémentation. On dispose également souvent de proprietés agréables comme la décidabilité du typage : la vérification de la correction d'une preuve est automatique.

Bien sûr, plus un système est performant, plus on se rapproche du paradoxe ; pour citer Jean-Yves Girard, "la roche Tarpéiene est proche du Capitole". En d'autres termes, l'étude du système risque d'être plus difficile et dangereuse. Plus généralement, on peut essayer de préciser les caractéristiques souhaitables du système de types :

- Il doit permettre de définir une classe aussi large que possibles de fonctions calculables, ces dernières étant vue d'un point de vue extentionnel. Ceci correpond grossièrement à la notion de *complexité logique* du formalisme.
- D'un point de vue intentionnel, ces fonctions peuvent être définies de différentes manières ; on demande au système d'autoriser l'utilisation d'algorithmes *efficaces*. Les travaux de Loïc Colson ont souligné les subtiles différences entre ce point et le précédent.
- A l'intérieur de cette classe d'algorithmes, on veut être capable d'identifier autant de termes que possible.

Pour ce qui est des deux premiers points, relativement peu de progrès ont été accomplis depuis

l'introduction par Jean-Yves Girard des systèmes F et F_ω au début des années 1970 [58, 59]. De fait, l'expressivité de ces systèmes est telle, qu'il apparaît comme à peu près inutile de chercher à l'étendre [1]. En revanche, de nombreuses recherches ces dernières années ont porté sur le troisième point, et en particulier sur l'ajout de règles de réécriture aux règles de conversion habituelles du λ-calcul. De fait, le problème d'identifier syntaxiquement des programmes équivalents n'intéresse pas seulement les logiciens, mais rejoint le domaine de la sémantique des langages de programmation. Un exemple particulièrement frappant est le lien entre opérateurs de contrôle et logique classique (voir par entre autres [65, 107, 106]) qui a pu conduire des chercheurs se considérant plus logiciens qu'informaticiens à se pencher sur les travaux de Matthias Felleisen.

L'évolution des différentes variantes du Calcul des Constructions illustre également bien cette tendance. Le formalisme originel a été introduit en 1985 par Thierry Coquand et Gérard Huet [24, 30, 32]. Ce système combine la puissance expressive du système F_ω avec la représentation des preuves par l'isomorphisme de Curry-Howard. Ce système, remarquablement concis et élégant dans sa présentation, pouvait, alors, sembler achevé : le polymorphisme permet d'y coder les structures de données inductives et de définir sur ces derniers tous les algorithmes définissables dans F_ω et l'on dispose de la logique intuitioniste d'ordre ω pour raisonner sur ces objets. Là encore, c'est la mise en œuvre de ce système sur ordinateur qui a montré les limites du mode de représentation des données, et ainsi motivé l'ajout de *types inductifs primitifs* en 1989 par Thierry Coquand et Christine Paulin-Mohring [33] qui est étudié ici. La principale nouveauté, qui rend délicate l'étude du formalisme résultant est la possibilité de définir de nouveaux types (c'est-à-dire aussi de nouvelle propositions) par récurrence primitive sur des objets inductifs : on rajoute de nouvelles règles de conversion non seulement au niveau des preuves et programmes, mais également au niveau des types et des propositions. Dans la preuve de normalisation, il nous faut alors tenir compte de ces nouvelles identifications. Une autre extension, sans doute moins importante mais également utile, est la η-conversion, qui correspond à une forme faible d'égalité extensionnelle. L'ajout de la η-règle complique également l'étude du formalisme, car il fait dépendre la propriété de confluence de la propriété de normalisation.

On peut résumer cette thèse en disant qu'elle démontre que l'on peut faire cohabiter au sein d'un système cohérent un certain nombre de caractéristiques qui avaient déjà été étudiées et utilisées séparemment, à savoir :

- Le Calcul des Constructions ; c'est-à-dire les types dépendants et surtout le polymorphisme.
- Des types inductifs primitifs, permettant de construire des types par récurrence primitive (élimination forte).
- Une règle de conversion utilisant la η-réduction.

Il faut souligner qu'un effort particulier a été fait pour étudier un formalisme aussi proche que possible de ce qui peut être mis en œuvre sur ordinateur. En particulier, nous avons choisi d'utiliser une version de la règle de conversion (ou d'égalité de types) purement syntaxique. Ce choix, discuté plus en détail à la fin de cette thèse complique sensiblement certaines parties de la preuve de normalisation, mais nous semble indispensable, compte tenu des motivations de cette étude.

Le reste de ce travail est organisé comme suit :

Le premier chapitre commence par une rapide présentation du Calcul des Constructions à tra-

[1] Il faut modérer un peu cette affirmation : dans le cadre, par exemple, de l'extraction de programmes, on trouve souvent des algorithmes faisant appel à une récurrence non-structurelle et qui ne peuvent donc pas être programmés directement tels quels dans des systèmes tel que F_ω étendu par des types inductifs. Pourtant, là aussi, il s'agit plus d'un problème de "souplesse" du langage de programmation que d'un manque de "force brute" du système de types. Voir par exemple [125].

vers quelques exemples simples. On met ensuite en valeur le rôle des définitions inductives et on motive l'ajout d'un mécanisme primitif pour celles-ci. On présente informellement les contraintes de positivité. Enfin la présentation de la dérivation de quelques résultats formels sur le λ-calcul simplement typé permet d'illustrer l'utilisation des types inductifs.

Le deuxième chapitre commence par la présentation formelle et détaillée du Calcul des Constructions inductives (CCI). On y introduit également un certain nombre de notations, et on y démontre un certain nombre de propriétés importantes, dont en particulier la correction de la réduction vis-à-vis du typage.

La normalisation de CCI est démontrée dans le chapitre 4. On se sert d'une adaptation de la méthode de réductibilité due à Tait et perfectionnée par Girard. La preuve de normalisation de CCI est longue et technique ; or une preuve de normalisation ne peut être que réellement convaincante si le lecteur en a une compréhension intuitive. Pour cette raison, le chapitre 3 présente les idées essentielles de la technique de réductibilité à travers divers systèmes de types de complexité croissante.

On termine l'étude méta-théorique dans le chapitre 5, où l'on se sert du résultat de normalisation pour vérifier des propriétés comme la cohérence logique du système, sa confluence et la décidabilité du typage. En y discutant également de présentations alternatives du système, nous indiquons également pourquoi il semble inévitable de faire un choix entre un système plus facile à étudier mais mal-adapté à une implémentation, et un système de mise en œuvre plus efficace mais plus ardu à étudier.

En appendice on présente un résultat qui relie deux problèmes ouverts : la confluence et la possibilité de construire un point-fixe dans des systèmes non-normalisants. On présente également une formalisation du paradoxe de Russell qui permet la construction d'un point fixe dans une variante incohérente de CCI.

Il est à peu près indispensable de respecter l'ordre des chapitre à la lecture. Seul le chapitre 3 peut être lu indépendamment (au moins dans ses premières parties). Le lecteur familier avec le Calcul des Constructions et l'isomorphisme de Curry-Howard pourra sauter tout ou partie du chapitre 1.

Chapitre 1

Une Tentative de Présentation par l'Exemple

Cette présentation du Calcul des Constructions et de son extension par des types inductifs, est délibérément modeste. Nous chercherons simplement à illustrer l'utilisation de ce calcul comme formalisme logique à travers quelques exemples simples. On insistera plus particulièrement sur la description des types et prédicats inductifs pour permettre une compréhension plus intuitive des définitions qui seront détaillées dans le chapitre 2.

De manière générale, il semble que le principe de l'isomorphisme de Curry-Howard, qui est au cœur du Calcul des Constructions, ne soit pas toujours facile ni à assimiler, ni à expliquer. Parmi diverses introductions, il nous faut mentionner celle du premier chapitre de la thèse de Gilles Dowek [38], remarquable de clarté et de concision ; nous ne pouvons que recommander sa lecture au néophyte.

1.1 Un langage de programmation – rappels de λ-calcul typé

L'isomorphisme de Curry-Howard, c'est l'idée que des preuves et des algorithmes peuvent être représentés de manière uniforme. Pour le comprendre, il faut donc commencer par maîtriser ces deux notions. Nous avons déjà mentionné dans l'introduction le rôle crucial qu'est appelé à jouer la définition de fonctions sous une forme *intentionnelle*, c'est-à-dire par la donnée d'un algorithme. En d'autre termes, le formalisme permet la description formelle d'algorithmes et contient une méthode de *calcul* ou d'évaluation de ces algorithmes. Ces deux caractéristiques correspondent à la définition abstraite d'un langage de programmation.

Le λ-calcul a été introduit par Alonzo Church, précisément pour décrire les objets de son formalisme logique, appelé "Logique d'Ordre Supérieur" (Higher-Order Logic) [16]. Il reste l'un des moyens les plus élégants de décrire des calculs de manière abstraite. Sans rentrer dans les détails historiques, on peut rappeler qu'une première version du formalisme de Church s'est révélée incohérente, parce qu'utilisant le λ-calcul non-typé, qui est trop expressif algorithmiquement.

En λ-calcul typé, tout objet bien-formé possède un type. On écrira toujours $t : T$ pour dire que l'objet t est de type T. A l'image de ce qui se passe dans un langage de programmation comme ML, les principales structures de données (entiers naturels, listes) correspondent chacune à un type. Par exemple l'objet 0 est bien-formé de type Nat (le type des entiers naturels) :

$0 : \mathrm{Nat}$

Par ailleurs, à tout entier naturel, nous pouvons associer son successeur. Le successeur est donc une application des entiers naturels vers eux-mêmes. On utilise la flèche pour désigner le type des applications : un objet de type $A \to B$ est une application (ou plus exactement une description intentionnelle d'une application) des objets de type A vers les objets de type B. Dans le cas du successeur S, on a donc :

$S : \text{Nat} \to \text{Nat}$

Une fonction peut, bien sûr, être appliquée à un argument, pourvu que celui-ci soit du type demandé. L'application de la fonction S à l'argument O est notée $(S\ 0)$. Comme S est de type Nat $\to$ Nat, on a :

$(S\ 0)$: Nat
$(S\ (S\ 0))$: Nat
...

Convention d'écriture Une simplification usuelle est de ne pas écrire toutes les parenthèses lorsqu'une fonction est appliquée successivement à plusieurs arguments : $(\ldots(f\ a_1)\ldots\ a_n)$ sera noté $(f\ a_1\ \ldots\ a_n)$. Par ailleurs, on utilisera la notation traditionnelle $f(x)$ pour l'application pour les raisonnements méta-mathématiques, lorsque f est une fonction au sens de la théorie des ensembles (c'est-à-dire lorsque f est en fait un prédicat). La notation "λ-calcul" étant réservée aux objets du formalisme.

La définition de nouvelles fonctions est possible grâce à l'abstraction, ou λ-notation. La fonction qui à un objet x de type A associe un objet M est notée $\lambda x : A.M$ ou $[x : A]M$. Bien sûr, on peut utiliser la *variable* x pour construire l'objet M. L'exemple le plus simple est la fonction identité sur un type A :

$[x : A]x : A \to A$

Règles

On peut utiliser ces quelques termes très simples pour illustrer le principe des règles de typage. Pour cela, il nous faut introduire la notion de *contexte* : comme un terme peut contenir des variables non-définies (ou "libres"), on ne peut le typer que si l'on précise d'abord le type de ces variables. Un contexte, généralement désigné par une majuscule grecque (Γ ou Δ) est donc une liste de paires composées chacune d'une variable et d'un type. On désigne par [] le contexte vide, et par $\Gamma :: (x, T)$ le contexte Γ auquel on a ajouté la paire (x, T). Un jugement de typage est donc décrit par un triplet (contexte, terme, type). On écrit

$$\Gamma \vdash t : T$$

pour dire que t est de type T dans le contexte Γ. Dans ce chapitre, on se permettra d'omettre le contexte lorsque celui-ci sera vide ou qu'il sera facile de le deviner.

Voici les règles d'inférence du λ-calcul simplement typé, qui est un fragment de notre système :

$$\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash (t_1\ t_2) : B} \qquad \frac{\Gamma :: (x, A) \vdash t : B}{\Gamma \vdash [x : A]t : A \to B}$$

$$\frac{\Gamma\ wf \qquad (x, A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash A : \text{Set} \qquad x \notin \Gamma}{\Gamma :: (x, A)\ wf}$$

Les deux premières règles, permettant de typer l'abstraction et l'application sont faciles à comprendre. Le jugement Γ *wf* signifie simplement que Γ est *bien-formé* ; $\Gamma \vdash A$: Set signifie que A

est un type bien-formé dans le contexte Γ. La nécessité de ces jugements apparaîtra dans la section suivante.

Remarque Les règles ci-dessus correspondent au fragment simplement typé du Calcul des Constructions. Elles sont admissibles dans ce Calcul, mais non primitives. En fait on verra que l'opérateur $\rightarrow$, de construction de type, n'est qu'un cas particulier des types de fonctions dans ce calcul. Il en va de même du jugement Γ *wf*.

Réduction

La règle de calcul essentielle est la *β-réduction*. Elle correspond à l'élimination conjointe d'une abstraction et d'une application : on dit que le terme $([x : A]M\ N)$ se β-réduit en $M[x \setminus N]$, c'est-à-dire le terme M dans lequel la variable x a été remplacée par N (cette notion de *substitution* est définie précisément dans le chapitre suivant.) On écrit :

$$([x : A]M\ N) \rhd_\beta M[x \setminus N].$$

On s'intéressera également à la *η-réduction* :

$$[x : A](M\ x) \rhd_\eta M \quad \text{si } x \text{ n'est pas libre dans } M.$$

La η-réduction permet d'identifier certaines fonctions qui sont extentionnellement égales ; par exemple :

$$[x : \mathrm{Nat}](f\ x) \rhd_\eta f.$$

Notons que la distinction entre f et $[x : \mathrm{Nat}](f\ x)$ a d'ailleurs peu de sens en mathématiques usuelles.

On verra plus loin que le calcul est enrichi par une autre notion de réduction. On verra surtout comment la notion de réduction peut intervenir dans le typage.

1.2 Prédicats, propositions et types dépendants

L'autre volet de l'isomorphisme de Curry-Howard est que les λ-termes peuvent également être utilisés pour représenter des preuves de manière naturelle. Dans ce paradigme, le langage des types est enrichi pour permettre à ceux-ci de représenter des propositions. Si un type P correspond à une proposition, un terme t de type P est alors une preuve de cette proposition. Les propositions prouvables sont donc les *types habités*. Pour s'habituer à cette idée, il est utile de rappeler la sémantique proposée par Heyting pour interpréter les preuves. En voici la version typée :

- Une preuve de la conjonction $A \wedge B$ est une paire formée d'une preuve de A et d'une preuve de B
- une preuve de la disjonction $A \vee B$ est une paire dont la seconde composante est soit une preuve de A soit une preuve de B, la première composante étant un drapeau (*flag*) indiquant s'il s'agit d'une preuve de A ou de B
- une preuve de l'implication $A \Rightarrow B$ est une application construisant une preuve de B à partir de toute preuve de A
- une preuve de la proposition existentielle $\exists x : A.P(x)$ est une paire composée d'un objet x_0 de type A et d'une preuve de la proposition $P(x_0)$

– une preuve de la proposition universelle $\forall x : A.P(x)$ est une fonction associant à tout objet x_0 de type A une preuve de $P(x_0)$.

Dans cette sémantique, les preuves sont donc soit des paires (typées) soit des fonctions totales, deux classes d'objets à la base de la programmation fonctionnelle, c'est-à-dire du λ-calcul typé. Les trois premiers cas de l'énumération ci-dessus, c'est-à-dire le fragment qui correspond au *calcul propositionnel*, peuvent être représentés dans le λ-calcul simplement typé :

– La conjonction logique correspond au produit du λ-calcul : si les type $\overline{A}$ et $\overline{B}$ représentent respectivement les propositions A et B, alors le type produit (souvent noté $\overline{A} * \overline{B}$) représente bien les preuves de $A \wedge B$.
– La disjonction est représentée par le type somme : les objets de type $\overline{A} + \overline{B}$ sont bien les preuves de $A \vee B$.
– L'implication enfin est naturellement interprétée par le type flèche. Un objet de type $\overline{A} \to \overline{B}$ associe bien un objet de type $\overline{B}$ (preuve de B) à tout objet de type $\overline{A}$ (preuve de A). Cet objet a donc bien la sémantique d'une preuve de $A \Rightarrow B$.

Informellement, ceci signifie que le λ-calcul simplement typé représente la logique propositionnelle intuitionniste. Les règles d'inférence du λ-calcul correspondent très exactement aux règles de prouvabilité du calcul propositionnel. C'est le passage au calcul des prédicats qui nécessite une extension du système de type, et plus précisément du langage des types.

Considérons le cas d'un prédicat sur les entiers naturels comme *Even*, le prédicat de parité. Un tel prédicat est un objet qui à tout entier n associe une proposition (n est pair.) Une proposition étant un type, c'est-à-dire encore un objet de type Set, le plus facile est de le typer comme une fonction des entiers vers les types/propositions. On aura donc :

$\mathit{Even} : \mathrm{Nat} \to \mathrm{Set}$

et étant donné un terme $n : \mathrm{Nat}$, on pourra construire :

$(\mathit{Even}\ n) : \mathrm{Set}$

L'objet $(\mathit{Even}\ n)$ est donc un type qui contient un terme n ; on dit qu'il *dépend* de n. L'extension qui permet la représentation d'un calcul des prédicats dans un λ-calcul typé est donc ce qu'on appelle la possibilité de construire des *types dépendants*. C'est elle, bien sûr, qui donne toute sa portée à l'isomorphisme de Curry-Howard.

Formellement, cette extension correspond à de nouvelles règles de typage permettant la construction de ces types dépendants. Elle nécessite également un changement dans la notation du type des fonctions : considérons un type fonctionnel $A \to B$; dans un calcul avec types dépendants, le type du résultat B doit pouvoir dépendre de la valeur de l'argument $x : A$. La notation $\to$ ne permet pas de faire apparaître cette dépendance. On la remplace donc la notation $(x : A)B$, où x est liée, et peut donc apparaître, dans B. On lit en général cette notation "pour tout x de type A, B". En effet, on retrouve bien la sémantique de Heyting du quantificateur universel : puisqu'une preuve de $\forall x : A.B(x)$ est une fonction qui à tout x_0 de type A associe une preuve de $B(x_0)$, une preuve de cette proposition sera simplement un terme de type

$$(x : A)B.$$

Les règles de typage de ces types, aussi appelés *produits dépendants*, sont similaires à celle des types flèches, mais en faisant apparaître la dépendance lors de l'application :

$$\frac{\Gamma :: (x, A) \vdash B : \mathrm{Set}}{\Gamma \vdash (x : A)B : \mathrm{Set}} \qquad \frac{\Gamma \vdash t_1 : (x : A)B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash (t_1\ t_2) : B[x \setminus t_2]}$$

Il est facile de voir que le produit dépendant généralise strictement la flèche. On n'a donc plus besoin de celle-ci dans le calcul. Plus exactement, on va conserver la notation $A \to B$ comme une abréviation de $(x : A)B$ dans le cas où x n'apparaît pas (n'a pas d'occurrences libres) dans B. Les règles de typage que nous avons données pour la flèche deviennent donc des règles admissibles, puisqu'elles sont un cas particulier des règles ci-dessus.

Nous ne présenterons pas le formalisme dit de "déduction naturelle" du à Prawitz [130]. Remarquons tout de même que les deux règles ci-dessus correspondent exactement à l'introduction et l'élimination du quantificateur universel. La remarque importante est :

Dans un λ-calcul avec types dépendants, on unifie le mécanisme d'abstraction/application de la programmation fonctionnelle et le mécanisme logique d'introduction/élimination du quantificateur universel.

Notons aussi que les règles ci-dessus ne suffisent pas à construire le type Nat $\to$ Set (en fait $(x : \text{Nat})\text{Set}$) du prédicat *Even*. On a pour cela besoin de la règle :

$$\frac{\Gamma :: (x, A) \vdash K : \text{Type}}{\Gamma \vdash (x : A)K : \text{Type}}$$

qui permet de dériver Nat $\to$ Set : Type. L'objet Type est une constante particulière comme Set, avec la règle Set : Type. De manière générale, Type est donc le type des prédicats (et en particulier des types.)

1.3 La règle de conversion

Nous avons indiqué comment les règles de réduction du λ-calcul permettent d'effectuer des calculs sur les objets du formalisme. Ces calculs permettent d'identifier un certain nombre de termes. Par exemple, on voudrait que $([x : \text{Nat}]x\ 0)$ et 0 désignent effectivement le même objet. Pour cela, on introduit la règle de *conversion*. On définit la relation d'équivalence $=_{\beta\eta}$ sur les termes comme la cloture reflexive, symétrique et transitive de $\rhd_\beta$ et $\rhd_\eta$. La règle de conversion est alors :

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash T' : s \qquad T =_{\beta\eta} T'}{\Gamma \vdash t : T'}$$

On comprend que cette règle permette d'identifier deux termes convertibles. Pour reprendre l'exemple ci-dessus, étant donné un prédicat P sur les entiers, et une preuve p de $(P\ 0)$, alors p est aussi une preuve de $(P\ ([x : \text{Nat}]x\ 0))$. On a donc "gratuitement" la preuve que 0 est égal à $([x : \text{Nat}]x\ 0)$. Ceci signifie aussi que plus on ajoute de règles de réductions au système, plus on pourra identifier de termes mécaniquement, ce qui permet de prouver plus de propositions. De là l'interêt de considérer la η-réduction, et aussi les réductions liées aux types inductifs primitifs qui seront présentées plus loin.

Si l'on peut parler de gratuité pour certaines preuves, c'est bien sûr que la relation de conversion reste décidable, c'est-à-dire peut toujours être vérifiée par la machine. Plus précisément, on verra que la propriété essentielle de ce système est la décidabilité du typage : il existe un algorithme qui vérifie si un jugement $\Gamma \vdash t : T$ est dérivable ou non. La clé de voute de cet algorithme est précisément la décidabilité de la conversion, qui repose sur les deux propriétés de *normalisation* et de *confluence* qui sont les résultats principaux de ce travail.

Le choix fait de disposer d'une relation de typage décidable interdit de trop étendre la règle de conversion. Certains systèmes, comme celui implémenté dans NuPRL [21] renoncent à la décidabilité du typage en faveur d'une plus grande souplesse de la règle de conversion.

1.4 Les règles du Calcul des Constructions

On peut maintenant résumer les diverses idées présentées jusqu'ici en donnant les règles de typage du Calcul des Constructions. On simplifie grandement la présentation formelle de ce système en identifiant syntaxiquement les différentes classes d'objets (termes, types, etc.) Les objets de ce calcul sont donc obtenus à partir des quatre constructeurs vus jusqu'ici : variables, abstraction, application et produit ainsi que les constantes Set et Type. La grammaire correspondante est :

$$t \ := \ x \mid (t\ t) \mid [x:t]t \mid (x:t)t \mid s$$

où s désigne Set ou Type. Les règles du calcul sont alors :

$$(\text{Ax}_1)\ \ [] \vdash \text{Set} : \text{Type}$$

$$(\text{Prod-}s)\ \frac{\Gamma :: (x:t_1) \vdash t_2 : s}{\Gamma \vdash (x:t_1)t_2 : s} \qquad (\text{Lam})\ \frac{\Gamma \vdash (x:t_1)t_2 : s \qquad \Gamma :: (x:t_1) \vdash t : t_2}{\Gamma \vdash [x:t_1]t : (x:t_1)t_2}$$

$$(\text{W})\ \frac{\Gamma \vdash t : s \qquad \Gamma \vdash A : B \qquad x \notin \Gamma}{\Gamma :: (x:t) \vdash A : B}$$

$$(\text{Var})\ \frac{\Gamma \vdash t_1 : t_2 \qquad (x,t) \in \Gamma}{\Gamma \vdash x : t} \qquad (\text{App})\ \frac{\Gamma \vdash t_2 : (x:T_1)T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash (t_2\ t_1) : T_2[x \setminus t_1]}$$

$$(\text{Conv})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_1 : s \qquad \Gamma \vdash T_2 : s \qquad T_1 =_{\beta\eta} T_2}{\Gamma \vdash t : T_2}$$

1.5 Le rôle des définitions inductives

Habituellement, on définit un ensemble défini inductivement comme le plus petit ensemble vérifiant un certain nombre de prédicats. Par exemple l'ensemble des entiers naturels peut être vu comme le plus petit ensemble $\mathbb{N}$ qui vérifie :

- $0 \in \mathbb{N}$ (0 est un entier naturel)
- Si $n \in \mathbb{N}$ alors $n+1 \in \mathbb{N}$ (l'ensemble est clos par successeur).

Ce type de définition joue un rôle important en mathématiques. Il devient essentiel lorsque l'on travaille dans une théorie des types, où il s'utilise de manière très naturelle. Dans Coq, on dispose d'une instruction dédiée ; par exemple :

```
Inductive Definition nat : Set =
0 : nat | S : nat -> nat.
```

qui construira effectivement les objets Nat : *Set*, 0 : Nat, S : Nat $\rightarrow$ Nat. Mais surtout, le fait que Nat est le *plus petit* type construit à partir des deux *constructeurs* 0 et S est illustré par l'existence de schémas de récurrence :

Nat_*rec* : $(P : \text{Nat} \rightarrow \mathit{Set})(P\ 0) \rightarrow ((m : \text{Nat})(P\ m) \rightarrow (P\ (S\ m))) \rightarrow (n : \text{Nat})(P\ n)$

c'est-à-dire que pour tout prédicat P sur les entiers, si $(P\ 0)$ est vérifié, et si $(P\ (S\ m))$ est vérifié sous la condition $(P\ m)$, alors tout entier vérifie aussi P. On reconnaît bien sûr le neuvième axiome

de Peano [126]. Il est utile de remarquer dès maintenant que ces schémas de récurrence ne sont pas simplement des axiomes logiques, mais des opérateurs munis d'un comportement calculatoire particulier. Toujours dans le cas des entiers naturels, on aura les règles de réduction suivantes :

$$(\mathrm{Nat\_}rec\ P\ p_0\ p_S\ 0) \rhd p_0$$
$$(\mathrm{Nat\_}rec\ P\ p_0\ p_S\ (S\ n)) \rhd (p_S\ n\ (nat\_rec\ P\ p_0\ p_S\ n)).$$

Intuitivement, on comprend bien que ces règles de réductions préservent le typage : on aura P : Nat $\rightarrow$ Set, $p_0 : (P\ 0)$, $p_S : (m : \mathrm{Nat})(P\ m) \rightarrow (P\ (S\ m))$, et donc :

$$(\mathrm{Nat\_}rec\ P\ p_0\ p_S\ 0) : (P\ 0)$$
$$(\mathrm{Nat\_}rec\ P\ p_0\ p_S\ (S\ n)) : (P\ (S\ n)).$$

On peut également remarquer dès à présent que Nat_*rec* se comporte comme un opérateur de filtrage vis-à-vis de leur quatrième argument. De plus, lorsque cet argument est de la forme $(S\ m)$, on observe un *appel récursif*, puisque la fonction $(nat\_rec\ P\ p_0\ p_S)$ est rappelée sur l'argument n. On reconnaît donc la l'opérateur de *récurrence structurelle* sur les entiers naturels, tel qu'il existe dans le système T de Göedel, décrit dans le chapitre 3, mais ici l'utilisation des types dépendants permet de le typer de manière plus fine. On peut toutefois retrouver facilement la version sans type dépendants ; soit la définition :

$$\mathrm{Nat\_}rec' = [P : \mathrm{Set}](\mathrm{Nat\_}rec[n : \mathrm{Nat}]P) : (P : \mathrm{Set})P \rightarrow (\mathrm{Nat} \rightarrow P \rightarrow P) \rightarrow \mathrm{Nat} \rightarrow P.$$

Le comportement calculatoire de Nat_*rec*′ sera le même que celui de Nat_*rec*. De manière générale, on parlera de *schéma d'élimination dépendant* pour Nat_*rec* et de *schéma d'élimination non-dépendant* pour Nat_*rec*′. Cette distinction existe pour tous les types inductifs que nous considérons dans la suite. Le système génère également d'autres schémas, que nous considérerons plus tard.

Dans l'exemple ci-dessus, ainsi que dans la suite de cette partie, nous utilisons les caractères `verbatim` *pour les instructions Coq. En effet* `Inductive Définition ...` *est une* instruction *et ne désigne pas de* termes *du système formel proprement dit. Nous verrons par la suite quelle est la structure effective des termes construits par cette instruction.*

1.5.1 Types de données

La définition ci-dessus rappelle bien sur les types concrets de ML [47, 99, 103, 34, 4, 92]. De fait, une première utilisation des types inductifs dans un système tel que Coq est la représentation de structures de données classiques.

Sommes et Produits

Parmi les types les plus utilisés en programmation, et en particulier en programmation fonctionnelle, on trouve les sommes et les produits. Donnons-nous deux types A et B, que nous supposerons déjà construits par ailleurs. On définit le produit de A et B (souvent noté $A * B$) comme le plus petit type obtenu par le constructeur de paire :

```
Inductive Definition ProdAB : Set =
  pairAB : A -> B -> ProdAB.
```

Le schéma d'élimination correspondant étant :

ProdAB_*rec* : $(P : \text{ProdAB} \to \text{Set})((a : A)(b : B)(P\ (pairAB\ a\ b))) \to (p : \text{ProdAB})(P\ p)$

La somme de A et B, souvent notée $A + B$, désigne un type dont les habitants sont soit des objets de type A, soit des objets de type B, accompagnés d'une marque indiquant dans quel cas on se trouve. C'est donc le plus petit type obtenu à partir de deux constructeurs, qui sont les injections de A et B respectivement, vers le type somme :

```
Inductive Definition SumAB : Set =
  left : A -> SumAB
| right : B -> SumAB.
```

On obtient alors :

SumAB_*rec* : $(P : SumAB \to \text{Set})((a : A)(P\ (left\ a))) \to ((b : B)(P\ (right\ b))) \to (s : SumAB)(Ps)$

Types récursifs

Une caractéristique qui distingue les entiers naturels de la somme et du produit est que ce type est *récursif* : le constructeur S prend comme argument un objet qui est lui-même de type Nat. La représentation des entiers naturels est l'exemple le plus simple de type inductif récursif habité. En informatique on utilise couramment d'autres structures inductives récursives. Nous donnons l'exemple des listes homogènes et des arbres binaires :

```
Inductive Definition list : Set =
  nil : list
| cons : A -> list -> list.

Inductive Definition tree : Set =
  leaf : tree
| node : A -> tree -> tree -> tree.
```

Dans les deux cas, la récursivité de la définition est reflété par les schémas d'élimination. Ces derniers correspondent au principes de récurrences dit "structurels" sur les structures définies :

$$\begin{aligned} tree\_rec : \ & (P : tree \to \text{Set}) \\ & (P\ leaf) \to \\ & ((a : A)(T_1 : tree)(P\ T_1) \to (T_2 : tree)(P\ T_2) \to (P\ node\ a\ T_1\ T_2)) \to \\ & (T : tree)(P\ T) \end{aligned}$$

Représentation des types simples

Les définitions inductives sont on ne peut plus courantes en logique formelle. Par exemple lorsque, dans le chapitre suivant, nous construirons formellement les objets du formalisme, nous les utiliserons de manière essentielle. Plus simplement, voici les formalisations des objets du λ-calcul simplement typé ; tout d'abord les types :

```
Inductive Definition ST : Set =
  iota : ST
| Arr : ST -> ST -> ST.
```

On utilise les indices de de Bruijn pour les λ-termes :

```
Inductive Definition term : Set =
  var : nat -> term
| app : term -> term -> term
| abs : term -> term.
```

1.5.2 Définitions inductives parametrées

La définition précédente semble un peu restrictive : on aimerait pouvoir définir les listes, arbres, sommes et produits de manière générique et pas seulement pour des types A et B donnés. La possibilité, dans le Calcul des Constructions, de définir des fonctions des types vers les types, ou plus généralement des prédicats vers les prédicats, nous permet de paramétrer ces définitions, en construisant, par exemple, *list* :Set→Set. La syntaxe Coq est :

```
Inductive Definition list [A:Set] : Set =
  nil : (list A)
| cons : A->(list A)->(list A).
```

où :

```
Inductive Definition Sum [A,B:Set] : Set =
  inl : A->(Sum A B)
| inr : B->(Sum A B).
```

Les schémas d'élimination obtenus sont identiques au précédents, à cela près, qu'ils sont eux aussi aussi parametrés :

$$\begin{aligned} \mathit{list\_rec}: \quad & (A:\mathrm{Set})(P:(\mathit{list}\ A) \quad \to \mathrm{Set}) \\ & (P\ (\mathit{nil}\ A)) \to \\ & ((a:A)(l:(\mathit{list}\ A))(P\ l) \to (P\ (\mathit{cons}\ A\ a\ l))) \to \\ & (l:(\mathit{list}\ A))(P\ l) \\ \mathrm{SumAB\_rec}: & (A,B:\mathrm{Set})(P:(\mathit{Sum}\ A\ B) \to \mathrm{Set}) \\ & ((a:A)(P\ (\mathit{left}\ A\ B\ a))) \to ((b:B)(P\ (\mathit{right}\ A\ B\ b))) \to (s:(\mathit{Sum}\ A\ B))(P\ s) \end{aligned}$$

1.5.3 Prédicats inductifs

Les types définis précédemment sont tous des types de donnés. Ils ne diffèrent pas de ce qui peut être fait à l'aide des types concrets d'un langage comme ML. Dans un système comme le Calcul des Constructions, le schéma de définitions inductives est compatible avec les types dépendants : on peut définir des *prédicats inductifs*. Un exemple typique est la parité des entiers naturels : *Even* est le plus petit prédicat vérifié par 0 et vérifié par $(S\ (S\ n))$ s'il est vérifié par n. En Coq, cela donne :

```
Inductive Definition Even : nat -> Set =
  Even0 : (Even 0)
| EvenS : (n:nat)(Even n)->(Even (S (S n))).
```

En plus des objets *Even, Even0* et *EvenS* attendus, on construit le schéma d'élimination :

$$\begin{aligned} \mathit{Even\_rec} : \quad & (P : \mathrm{Nat} \to \mathrm{Set}) \\ & (P\ 0) \to \\ & ((n : \mathrm{Nat})(\mathit{Even}\ n) \to (P\ n) \to (P\ (S\ (S\ n)))) \to \\ & (n : \mathrm{Nat})(\mathit{Even}\ n) \to (P\ n) \end{aligned}$$

qui permet de prouver récursivement des propriétés sur les entiers pairs.

Égalité

Un autre exemple, non-récursif celui-là, est la définition de l'égalité. Il suffit de définir l'égalité comme le plus petit prédicat réflexif ! Étant donnés un type $A : \mathrm{Set}$ et un élément $a : A$ de ce type, on définit donc le prédicat "être égal à a" comme :

```
Inductive Definition Eqa : A -> Set =
 refla : (Eqa a).
```

c'est-à-dire aussi comme le prédicat vérifié uniquement par a. L'avantage essentiel de cette définition est que le schéma d'élimination associé nous permet de retrouver l'égalité de Leibniz :
$\mathit{Eqa\_rec} : (P : A \to \mathrm{Set})(Pa) \to (x : A)(\mathit{Eqa}\ x) \to (P\ x)$

On a bien sûr intérêt à paramétrer cette définition par A et a, en obtenant ainsi une définition générique de l'égalité :

```
Inductive Definition Eq [a:A;a:A] : A -> Set =
 refl : (x:A)(Eq A a x)
```

Le quantificateur existentiel

Nous avons vu comment formaliser un certain nombre de concepts mathématiques courants dans le Calcul des Constructions. En particulier le quantificateur universel se traduit immédiatement grâce à la quantification dépendante et les prédicats grâce aux types dépendants. Nous avons vu également comment formaliser les connecteurs logiques de conjonction et de disjonction par les types produits et sommes décrits ci-dessus. Nous pouvons donc plonger le calcul des prédicats intuitionniste dans les Constructions, à condition de définir le quantificateur existentiel.

```
Inductive Definition sig [A:Set;P:A->Set] =
  exist : (a:A)(P a)->(sig A P).
```

Remarquons que cette définition est une généralisation stricte du produit : la définition est la même, mais le type de la seconde composante de la paire peut, ici, dépendre de la valeur de la première. On retrouve ainsi la sémantique de Heyting du connecteur existentiel. Le schéma d'élimination correspond bien sûr à la règle d'élimination de ce quantificateur en déduction naturelle :

$\mathit{sig\_rec} : (A : \mathrm{Set})(P : A \to \mathrm{Set})(Q : \mathrm{Set})((x : A)(P\ x) \to Q) \to (\mathit{sig}\ A\ P) \to Q$

Le λ-calcul simplement typé

Les règles de typage du λ-calcul simplement typé définissent inductivement un prédicat ternaire sur les contextes, les types simples et les termes. Pour le formaliser, il nous faut d'abord construire les contextes comme listes de types :

```
Inductive Definition context : Set =
  nil : context
| cons : ST -> context -> context.
```

Un prédicat inductif permet de préciser qu'un type se trouve à une profondeur n dans un contexte :

```
Inductive Definition inpos [T:ST] : context -> nat -> Set =
  inposcar : (G:context)(inpos T (cons T G) O)
| inposcdr : (G:context)(n:nat)(T1 : ST)
                  (inpos T G n)->(inpos T (cons T1 G)(S n)).
```

A chaque règle de typage correspond alors une clause inductive :

```
Inductive Definition WTt : context -> term -> ST -> Set =
  Vart : (n:nat)(G:context)(T:ST)(inpos T G n)->(WTt G (var n) T)
| Appt : (G:context)(A,B:ST)(t1,t2:term)
                (WTt G t1 (Arr A B))->(WTt G t2 A)->(WTt G (app t1 t2) B).
| Lamt  : (G:context)(A,B:ST)(t:term)
               (WTt (cons A G) t B)-> (WTt G (lam t) (Arr A B)).
```

Une autre possibilité est de remarquer que l'arbre de dérivation étant isomorphe au terme typé, ce dernier est en quelque sorte superflu. On obtient alors une définition alternative plus simple[1] :

```
Inductive Definition in [T:ST] : context -> Set =
  incar : (G:context)(in T (cons T G))
| incdr : (G:context)(T1:ST)(in T G)->(in T (cons T1 G)).

Inductive Definition WT : context -> ST -> Set =
  Var : (T:ST)(G:context)(in T G)->(WT G T)
| App : (G:context)(A,B:ST)
          (WT G (Arr A B))->(WT G A)->(WT G B)
| Lam : (G:context)(A,B:ST)(WT (cons A G) B)->(WT G (Arr A B)).
```

1.5.4 La condition de stricte positivité

Même si nous ne nous sommes pas encore intéressés aux termes correspondant aux définitions inductives, nous avons vu que tout type inductif est entièrement décrit par les types de ses constructeurs. Toutefois, il est nécessaire d'imposer une restriction sur la forme que peuvent prendre ces types si l'on veut préserver la propriété de normalisation et la cohérence logique du système. Dans ce paragraphe, nous expliquons informellement la condition de *stricte positivité* sur les arguments des constructeurs. Aussi ce paragraphe est-il essentiel pour comprendre la définition formelle du système donné dans le chapitre 2.

[1] Cette deuxième version peut-être surprenante, car les variables ne semblent pas identifiées. En fait l'indice de de Bruijn de chaque variable est contenu implicitement dans la preuve de `(in T (cons T G))` qui permet effectivement de distinguer deux variables du même type. Le terme de type `(in ...)` doit donc être vue plus comme un terme qu'une preuve. C'est un bon exemple d'utilisation des types dépendants, à mi-chemin entre propositions et types de données.

Le cas général

Considérons un type inductif I, décrit par les types $C_1 \ \ldots \ C_n$ de ses n constructeurs. Pour simplifier, on considérera dans un premier temps que I n'est pas un prédicat (n'est pas un type dépendant) ; c'est-à-dire que I : Set. Chaque constructeur attend un certain nombre d'arguments pour construire un objet de type I. On aura donc pour tout i, avec $1 \leq i \leq n$:

$$C_i = (x_1^i : U_1^i)(x_2^i : U_2^i) \ldots (x_{n_i}^i : T_{n_i}^i)I$$

où n_i est donc *le nombre d'arguments du n-ième constructeur de* I. Les types U_j^i des arguments peuvent dépendre de I. Par exemple dans le cas des entiers naturels, avec $I = \text{Nat}$, le deuxième constructeur S est de type $\text{Nat} \rightarrow \text{Nat}$, et donc $n_2 = 1$ avec $U_1^2 = \text{Nat}$.

La restriction porte précisément sur cette dépendance des arguments des constructeurs vis-à-vis du type inductif ; elle est la suivante : pour tout U_i^j,

1. soit I n'apparaît pas dans U_i^j, et on dit alors que l'argument est non-récursif,
2. soit U_i^j est de la forme $(y_1 : V_1) \ldots (y_m : V_m)I$, où I n'apparaît pas dans les V_k, et l'argument est alors récursif.

On voit bien que pour les types non-récursifs décrits précédemment (produit, somme) on n'utilise que la clause 1. La clause 2 est utilisée pour les types récursifs comme les entiers naturels ou les listes. Le type de l'argument du constructeur S est Nat, qui est bien un sous-cas du schéma $(y_1 : V_1) \ldots (y_m : V_m)\text{Nat}$.

Une représentation des ordinaux

On peut remarquer que dans tous les exemples mentionnés jusqu'ici, on n'utilise pas le schéma dans toute sa généralité : on ne trouve pas ci-dessus d'arguments récursifs fonctionnels, c'est-à-dire de la forme $(y_1 : V_1) \ldots (y_m : V_m)I$ avec des séquences $y_1 \ \ldots \ y_m$ et $V_1 \ \ldots \ V_m$ non-vides. La définition inductive la plus connue avec argument récursif fonctionnel est une représentation des notations ordinales en théorie des types. Sa définition dans Coq est :

```
Inductive Definition Ord : Set =
  Oo : Ord
| So : Ord -> Ord
| lim : (nat -> Ord) -> Ord.
```

Bien entendu, `Oo` et `So` correspondent respectivement au 0 (zéro) et au successeur des ordinaux. Le constructeur `lim` appliqué à une suite f d'ordinaux représente la borne supérieure (réunion) de cette suite [2].

Indiquons le type et le comportement calculatoire du schéma d'élimination correspondant, qui correspond effectivement à une formulation du principe d'induction transfinie :

[2] Bien sûr il s'agit là d'une *représentation* des ordinaux, et celle-ci est forcément incomplète : tous les ordinaux ne peuvent pas être représentés. Plus encore, la collection des ordinaux représentés en théorie des types est un ensemble, et est donc majoré.

$$
\begin{array}{l}
Ord\_rec\ :\ (P : Ord \to \mathrm{Set}) \\
\quad (P\ O_o) \to \\
\quad ((y : Ord)(P\ y) \to (P\ (S_o\ y))) \to \\
\quad\ ((y : \mathrm{Nat} \to Ord)((n : \mathrm{Nat})(P\ (y\ n))) \to (P\ (lim\ y))) \to (o : Ord)(P\ o)
\end{array}
$$

$$
\begin{array}{l}
(Ord\_rec\ P\ p_0\ p_S\ p_l\ O_o)\ \triangleright\ p_O \\
(Ord\_rec\ P\ p_0\ p_S\ p_l\ (S_o x))\ \triangleright\ (p_S\ x\ (Ord\_rec\ P\ p_0\ p_S\ p_l\ x)) \\
(Ord\_rec\ P\ p_0\ p_S\ p_l\ (lim\ f))\ \triangleright\ (p_l\ f\ [n : \mathrm{Nat}](Ord\_rec\ P\ p_0\ p_S\ p_l\ (f\ n)))
\end{array}
$$

Définitions incorrectes

Un exemple de type inductif interdit est une définition comportant un cas $U_i^j = I \to I$; aussi la définition suivante est refusée par le système :

```
Inductive Definition absurd : Set =
 C : (absurd -> absurd) -> absurd.
```

De manière pragmatique, on peut expliquer ceci en montrant comment la même définition dans un langage de programmation fonctionnel fortement typé, par exemple en CAML [47], conduit à un programme non normalisable, sans utilisation du point-fixe. En CAML la construction correspondante est :

```
# type absurd = C of absurd -> absurd;;
```

En utilisant simplement le filtrage de CAML, on peut alors typer la fonction `f` suivante du type `absurd -> absurd` :

```
# let f = function (C g) -> g (C g);;
f : absurd -> absurd = <fun>
```

et il est alors facile de voir que le programme `f (C f)` va boucler. On peut même, en utilisant uniquement le filtrage sur les objets de type `absurd` construire le point fixe usuel. Ceci montre qu'il ne peut pas y avoir, dans le Calcul des Constructions Inductives, de schéma d'élimination associé au type `absurd`. Un peu plus loin, nous essayons d'expliciter ce point de manière plus théorique.

1.5.5 La forme des schémas d'élimination

Nous avons dit précédemment que le type et le comportement calculatoire des schémas d'élimination étaient inférés automatiquement à partir de la description des types inductifs, c'est-à-dire de la donnée des types de leurs constructeurs ; il nous reste à expliquer ce point. La description formelle du mécanisme présidant à la construction du type du schéma d'élimination est relativement peu parlante, alors que son principe sous-jacent est relativement simple. Aussi est-il, là encore, conseillé de bien comprendre ce paragraphe avant d'entamer la lecture du chapitre 2. Nous détaillons d'abord le cas des schémas *non-dépendants*.

Le cas non-dépendant

Comme dans le paragraphe précédent, on considère un type inductif I, et les types $T_1\ \ldots\ T_n$ de ses n constructeurs. On appellera $C_1\ \ldots\ C_n$ ces derniers. On a donc : $\forall i.C_i : T_i$. Une bonne

manière de comprendre le schéma d'élimination est de le voir d'abord comme un opérateur de filtrage (*pattern-matching*) exhaustif et très simple, voir minimal. Ce dernier est une fonction I_{rec_nodep} qui attend trois sortes d'arguments :

- Tout d'abord le type de l'objet que nous cherchons à construire ; dans un premier temps nous considérerons qu'il ne dépend pas de l'objet à destructurer. Nous l'appellerons $P : Set$.
- Ensuite viennent n termes $f_1 \ \ldots \ f_n$ qui forment les n branches du filtrage, chacune correspondant à un des n constructeurs de I.
- Enfin l'objet t de type I que l'on cherche à déstructurer.

En d'autres termes, pourvu que les $f_1 \ \ldots \ f_n$ soient correctement typés, on aura :

$$(I_{rec\_nodep}\ P\ f_1\ \ldots\ f_n\ t) : P.$$

Nous appellerons $\Delta\{T_i, P\}$ le type respectif de f_i, qui est défini ci-dessous. Le type de I_{rec} est alors :

$$I_{rec\_nodep} : (P : Set)(f_1 : \Delta\{T_1, P\}) \ldots (f_n : \Delta\{T_n, P\})I \to P.$$

Le fait que ce schéma soit dit non-dépendant tient simplement au fait que P ne dépend pas de t ; sinon on aurait $P : I \to \mathrm{Set}$.

Pour comprendre la construction des $\Delta\{T_i, P\}$, il est utile de considérer d'abord les règles de réduction associées au schéma d'élimination. Ce dernier correspondant à une étape de filtrage, il y a possibilité de réduction si t est de la forme $(C_i\ u_1\ \ldots\ u_{n_i})$. On a alors $u_j : U_j^i$. Supposons dans un premier temps que C_i soit un constructeur non-récursif, c'est-à-dire que I n'apparaisse pas dans les U_j^i ; on a alors la réduction suivante :

$$(I_{rec\_nodep}\ P\ f_1 \ldots f_n\ (C_i\ u_1\ \ldots\ u_{n_i})) \rhd (f_i\ u_1\ \ldots\ u_{n_i}).$$

Il est facile de comprendre que, dans ce cas, on aura

$$f_i : (x_1^i : U_1^i)\ \ldots\ (x_{n_i}^i : U_{n_i}^i)P = \Delta\{T_i, P\}.$$

Nous retrouvons ainsi les types des schémas d'élimination non-dépendants des types non récursifs présentés ci-dessus.

Si maintenant au moins l'un des arguments de C_i est récursif, le comportement calculatoire de I_{rec_nodep} est plus complexe, puisque, comme indiqué, par exemple, au paragraphe de présentation des entiers naturels, l'aspect "pattern-matching" du schéma d'élimination se double alors d'un aspect "récursion". Plus exactement, chaque fois que l'argument x_j^i d'un constructeur C_i est récursif, la branche correspondante du filtrage f_i attendra un argument supplémentaire, à savoir l'appel récursif de la fonction définie appliquée à x_j^i. Supposons plus précisément que $U_j^i = I$; alors là où on avait précédemment :

$$f_i : (x_1^i : U_1^i)\ \ldots\ (x_j^i : U_j^i)\ \ldots\ (x_{n_i}^i : U_{n_i}^i)P$$

on aura maintenant :

$$f_i : (x_1^i : U_1^i)\ \ldots\ (x_j^i : U_j^i)P \to\ \ldots\ (x_{n_i}^i : U_{n_i}^i)P$$

ce qui se reflète dans la règle de réduction :

$$(I_{rec\_nodep}\ P\ f_1 \ldots f_n\ (C_i\ u_1\ \ldots\ u_{n_i})) \rhd (f_i\ u_1\ \ldots\ u_j\ (I_{rec\_nodep}\ P\ f_1 \ldots f_n\ x_j^i)\ \ldots\ x_{n_i}^i).$$

À partir de là, il est facile de comprendre comment ceci se généralise aux constructeurs admettant plusieurs arguments récursifs. Remarquons également que ceci est suffisant pour retrouver les schémas d'élimination des entiers et des listes définis précédemment dans leurs versions non-dépendantes. On comprend aisément comment ceci se généralise aux cas de constructeurs admettant plusieurs arguments récursifs, comme les arbres binaires. En revanche, on voit déjà que la définition formelle de la forme générale de ces schémas ne sera pas très parlante au premier abord.

Le cas d'arguments récursifs plus complexe n'est pas très difficile. Prenons maintenant

$$U_j^i = (y_1 : V_1)\ \ldots\ (y_m : V_m)I.$$

On aura alors :

$$f_i : (x_1^i : U_1^i)\ \ldots (x_j : U_j^i)((y_1 : V_1)\ \ldots\ (y_m : V_m)P) \rightarrow\ \ldots\ (x_{n_i}^i : U_{n_i}^i)P$$

et comme règle de réduction :

$$(I_{rec\_nodep}\ P\ f_1 \ldots f_n\ (C_i\ u_1\ \ldots\ u_{n_i})) \triangleright$$
$$(f_i\ u_1\ \ldots\ u_j\ [y_1 : V_1]\ \ldots\ [y_m : V_m](I_{rec\_nodep}\ P\ f_1 \ldots f_n\ (x_j^i\ y_1 \ldots y_m))\ \ldots\ x_{n_i}^i).$$

Le cas dépendant

Une fois compris la formation et le typage des schémas d'élimination non-dépendants, il n'est pas très difficile de passer à leur homologue dépendant. Comme indiqué précédemment, les règles de réduction sont les mêmes. Il suffit donc de définir leur type. Le premier argument de I_{rec} sera un prédicat sur I : $P : I \rightarrow \mathrm{Set}$ (au lieu d'un type $P : \mathrm{Set}$ dans le cas de I_{rec_nodep}). Les types des branches f_i du filtrage seront appelés $\Delta\{T_i, P, C_i\}$, et définis par :

– Si tous les arguments de C_i sont non-récursifs :

$$\Delta\{T_i, P, c\} \equiv (x_1^i : U_1^i)\ \ldots\ (x_{n_i}^i : U_{n_i}^i)(P\ (c\ x_1^i\ \ldots\ x_{n_i}^i)).$$

– Si le j-ème argument de C_i est récursif de type

$$U_j^i = (y_1 : V_1)\ \ldots\ (y_m : V_m)I$$

on aura :

$$\Delta\{T_i, P, c\} \equiv (x_1^i : U_1^i)\ \ldots\ (x_j : U_j^i)((y_1 : V_1)\ \ldots\ (y_m : V_m)(P\ (x_j^i\ y_1\ \ldots\ y_m))))$$
$$\ldots\ (x_{n_i}^i : U_{n_i}^i)(P\ (c\ x_1^i\ \ldots\ x_{n_i}^i)).$$

On peut vérifier que l'on retrouve ainsi les schémas Nat_*rec*, *tree_rec*, etc, vus précédemment.

1.6 Représentation des Types Inductifs

Dans ce qui précède, nous avons décrit un certain nombre de propriétés essentielles (types, réductions) des objets correspondant à un mécanisme de définitions inductives. Nous avons mentionné que dans une implémentation telle que Coq, ce mécanisme correspondait à une instruction dédiée (`Inductive Definition`...). Il nous reste maintenant à étudier la structure de ces objets.

1.6.1 Le codage imprédicatif

Alonzo Church, l'inventeur du λ-calcul, avait très tôt remarqué qu'une classe de λ-termes pouvait être utilisée pour représenter les entiers naturels. C'est la représentation connue sous le nom "d'entiers de Church" :

$$\begin{aligned} 0 &\equiv \lambda x.\lambda f.x \\ 1 &\equiv \lambda x.\lambda f.(f\ x) \\ 2 &\equiv \lambda x.\lambda f.(f\ (f\ x)) \\ &\dots \\ n &\equiv \lambda x.\lambda f.(f^n\ x) \end{aligned}$$

Cette représentation n'est pas utilisable en λ-calcul simplement typé. En effet, on peut certes assigner un même type aux termes ci-dessus, mais il sera impossible de typer correctement le terme qui effectuera l'exponentiation de deux entiers de Church[3]. En revanche, lorsque Jean-Yves Girard a introduit le λ-calcul typé du second ordre, ou système F au début des années 1970 (voir [58, 59]), il a remarqué que grâce au polymorphisme on pouvait, dans son système, typer uniformément les entiers de Church et les fonctions prouvablement totales dans l'arithmétique du second ordre. C'est pour cette raison que le système F, conçu à l'origine pour prouver la cohérence logique de l'arithmétique fonctionnelle du second ordre, n'a pas besoin d'entiers primitifs, contrairement au système T, introduit pour prouver la cohérence de l'arithmétique du premier ordre. On peut trouver les définitions et les preuves de normalisation de ces systèmes dans le chapitre 3 ; voir aussi [61, 83].

Comme le Calcul des Constructions est une extension du système F, on peut également y définir les entiers de Church ; il suffit de prendre :

$$\text{Nat} \equiv (A : Set)A \to (A \to A) \to A.$$

Il est possible de généraliser cette opération aux autres types inductifs ; il existe ainsi des *codages imprédicatifs* des listes, sommes, produits, ordinaux, etc. L'étude précise de ces codages a été faite par Corrado Böhm et Alessandro Berarducci [12], puis dans le cadre du Calcul des Constructions par Christine Paulin-Mohring et Franck Pfenning [119, 120, 127]. Il faut noter que cette possibilité semble avoir été l'une des motivations principales de l'introduction du polymorphisme dans le Calcul des Constructions.

Si cette technique de représentation est très utile pour, par exemple, donner des preuves de cohérence de formalismes logiques, il présente aussi un certain nombre de défauts, qui se sont révélés particulièrement gênants avec la mise en œuvre de preuves formelles effectivement vérifiées par l'ordinateur et la quête de programmes efficaces extraits de preuves :

- On ne peut, sans extension du système, construire que les schémas d'élimination non-dépendants.
- Le comportement calculatoire n'est pas exactement celui souhaité. En particulier la règle de réduction
$$(I_{rec}\ P\ f_1\ \dots\ f_n\ (C_i\ a_1\ \dots a_m)) \rhd (f_i\ a_1\ \dots)$$
n'est en fait vérifiée en général que si le terme $(C_i\ a_1\ \dots a_m)$ est *clos*. Ceci peut être gênant car on souhaite utiliser l'égalité correspondante.
- La représentation imprédicative est mauvaise d'un point de vue informatique : elle est relativement gourmande en place mémoire, et surtout les calculs sont longs ; le calcul du prédécesseur d'un entier de Church n nécessite un temps au moins proportionnel à n.

[3] En fait, comme cela a été montré par Helmut Schwichtenberg, les seules fonctions définissables sur les entiers de Church en λ-calcul simplement typé sont les polynômes (à coefficients constants), étendus par le test à zéro.

- Comme indiqué par Christine Paulin-Mohring, la représentation par le codage imprédicatif n'est fidèle que pour les types de données : dès que l'un des arguments de l'un des constructeurs est fonctionnel, il devient possible d'habiter le type imprédicatif par des termes clos qui ne peuvent pas être obtenus à partir des constructeurs.
- Le codage imprédicatif est mal adapté à l'extraction de programmes vers des compilateurs de type ML : on souhaite en effet utiliser les types concrets et le filtrage de ces compilateurs.
- Il est impossible de prouver que $0 \neq 1$ dans le Calcul des Constructions pur (voir par exemple [150]). Plus généralement il est exclu de prouver que deux constructeurs d'un type inductif sont différents, sans étendre le système.

1.6.2 Les types inductifs primitifs

Ces différentes raisons ont conduits Thierry Coquand et Christine Paulin-Mohring à proposer une extension au Calcul des Constructions, précisément destinée à permettre un traitement plus satisfaisant des définitions inductives. L'idée est d'ajouter trois nouveaux objets syntaxiques au Calcul : un pour représenter les types inductifs proprement dits, un pour les constructeurs et un pour les schémas d'élimination. Les notations sont les suivantes :

- Les types (ou les prédicats) inductifs sont représentés par le terme

$$\mathsf{Ind}(X : A)\{T_1(X);\ \ldots\ ;T_n(X)\}$$

qui doit être compris ainsi : A est le type de l'objet défini, $T_1(X)\ \ldots\ T_n(X)$ sont les types des n constructeurs. On souligne que la variable X est liée dans les $T_i(X)$ pour permettre des arguments récursifs. Par exemple le type Nat sera simplement une abréviation de :

$$\mathrm{Nat} \equiv \mathsf{Ind}(X : \mathrm{Set})\{X; X \to X\} : \mathrm{Set}$$

de même on aura :

$$\mathit{list} \equiv [A : \mathrm{Set}]\mathsf{Ind}(X : \mathrm{Set})\{X; A \to X \to X\} : \mathrm{Set} \to \mathrm{Set}$$

$$\mathit{Even} \equiv \mathsf{Ind}(X : \mathrm{Nat} \to \mathrm{Set})\{(X\ 0); (n : \mathrm{Nat})(X\ n) \to (X\ (S\ (S\ n)))\} : \mathrm{Nat} \to \mathrm{Set}.$$

- Pour les constructeurs, le terme $\mathsf{Constr}(i, I)$ désigne le i-ème constructeur du type inductif I. On aura ainsi simplement :

$$0 \equiv \mathsf{Constr}(1, \mathrm{Nat}) : \mathrm{Nat}$$

c'est-à-dire dans sa forme expansée :

$$0 \equiv \mathsf{Constr}(1, \mathsf{Ind}(X : \mathrm{Set})\{X; X \to X\}) : \mathsf{Ind}(X : \mathrm{Set})\{X; X \to X\}$$

ainsi que :

$$S \equiv \mathsf{Constr}(2, \mathrm{Nat})$$

$$\mathit{nil} \equiv \mathsf{Constr}(1, \mathit{list})$$

etc.

– Le terme correspondant aux schémas d'élimination est un peu plus compliqué :

$$\mathsf{Elim}(I, Q, [u_1;\ \ldots\ ; u_p], t)\{f_1;\ \ldots\ ; f_n\}$$

qui doit être compris ainsi : I est le type inductif sur lequel on fait l'élimination, t est l'objet que l'on élimine et les f_i sont les "branches" du filtrage. La liste de termes $[u_1;\ \ldots\ ; u_p]$ sert a préciser le type de t dans le cas où I est un prédicat inductif. De manière générale on aura :

$$t : (I\ u_1\ \ldots\ u_n)$$

et bien sûr si I : Set ou I : Prop, la liste $[u_1;\ \ldots\ ; u_p]$ sera vide. Enfin Q sert simplement à préciser le type de l'objet fabriqué :

$$\mathsf{Elim}(I, Q, [u_1;\ \ldots\ ; u_p], t) : (Q\ u_1 \ldots u_p\ t).$$

Par exemple, les schémas d'élimination vus précédemment sont simplement des formes "curryfiées" de l'opérateur Elim :

$$\mathrm{Nat\_}rec \equiv [Q : \mathrm{Nat} \to \mathrm{Set}][f_1 : (Q\ 0)][f_2 : (m : \mathrm{Nat})(Q\ m) \to (Q\ (S\ m))][n : \mathrm{Nat}]$$
$$\mathsf{Elim}(\mathrm{Nat}, Q, [], n)\{f_1; f_2\}\ .$$

Il s'agit là bien sûr d'une description sommaire et intuitive. Les règles de typage précises, qui formalisent entre autres la contrainte de stricte positivité et les nouvelles règles de réduction, sont détaillées dans le chapitre 2.

1.6.3 L'élimination forte

Il nous reste à indiquer comment prouver $0 \neq 1$. Pour ce faire, il faut exhiber un prédicat sur les entiers P : Nat $\to$ Set, tel que $(P\ O)$ et la négation de $(P\ (S\ O))$ soient prouvables. Les outils que nous avons décrits jusqu'ici ne suffisent pas pour construire un tel prédicat. Il nous faut pour cela être capables de définir une proposition par filtrage sur un entier naturel. En d'autre termes, on doit disposer d'un schéma d'élimination sur les entiers permettant de construire des objets de type Set. C'est cette possibilité qu'on appelle l'élimination forte. La version la plus simple d'un tel schéma serait :

$\mathrm{Nat\_}recs$: Set $\to$ (Nat $\to$ Set $\to$ Set) $\to$ Nat $\to$ Set
avec :

$$\begin{aligned} (\mathrm{Nat\_}recs\ Q_0\ Q_s\ O) &\ \triangleright\ Q_0 \\ (\mathrm{Nat\_}recs\ Q_0\ Q_s\ (S\ n)) &\ \triangleright\ (Q_s\ n\ (\mathrm{Nat\_}recs\ Q_0\ Q_s\ n)) \end{aligned}$$

Pour prouver $0 \neq 1$, on peut alors construire :

$Disc \equiv (\mathrm{Nat\_}recs\ \mathrm{True}\ [n : \mathrm{Nat}][Q : Set]\mathrm{False}) : \mathrm{Nat} \to \mathrm{Set}$
On a alors $(Disc\ O)$ qui est convertible à True, et donc trivialement prouvable, et $(Disc\ (S\ O))$ qui est convertible à False. Si on suppose $(Eq\ \mathrm{Nat}\ O\ (S\ O))$, comme on a $(Disc\ O)$, on peut en déduire $(Disc\ (S\ O))$, c'est-à-dire False. On a donc $(Eq\ \mathrm{Nat}\ O\ (S\ O)) \to \mathrm{False}$ (en clair, $0 \neq 1$.)

Forme générale

Le type proposé ci-dessus pour Nat_*recs* correspond à une version non-dépendante de l'élimination forte, spécialisée à la seule construction d'objets de type Set. On peut donner un type plus général à Nat_*recs*, qui permet de construire un prédicat de toute arité par élimination forte :

Nat_$recs : (Q : \text{Type})Q \to (nat \to Q \to Q) \to \text{Nat} \to Q$

où, rapellons-le, Type est le type de Set. Ce type correspond exactement au schéma d'élimination non-dépendent des entiers où l'on remplace Set par Type. On peut généraliser encore un peu en prenant le schéma dépendant correspondant :

Nat_$recs : (Q : \text{Nat} \to \text{Type})(Q\ O) \to ((n : \text{Nat})(Q\ n) \to (Q\ (S\ n))) \to (n : \text{Nat})(Q\ n)$

De manière générale, on obtient donc le type de l'opérateur d'élimination forte d'un type inductif I en remplaçant la quantification $(P : I \to \text{Set})$ du type de l'élimination faible par une quantification $(P : I \to \text{Type})$. Les règles de réduction restent donc inchangées.

Il faut remarquer que pour pouvoir construire les types des schémas d'élimination forte tels qu'ils sont décrits ici, il faut légèrement étendre le système de type. En effet, dans le Calcul des Constructions, on n'autorise pas la constructions d'objets tels que Nat $\to$ Type (nous appellerons cette classe d'objets *schémas d'ordres.*) Il est toutefois assez facile de modifier les règles du système, pour l'étendre avec ce niveau supplémentaire. Là encore, on trouvera la description précise dans le chapitre suivant.

L'élimination forte permet donc d'utiliser le langage de programmation, avec toute son expressivité calculatoire, pour construire de nouveaux types. Il est clair que c'est par cette caractéristique que le Calcul des Constructions Inductives se démarque le plus nettement du Calcul des Constructions pur. Par exemple les schémas d'élimination (faibles) dépendants peuvent être facilement axiomatisés dans les Constructions, et ces axiomes sont même *réalisables* (voir [120]). La preuve de $0 \neq 1$ par contre, ainsi que les quelques autres exemples de cette section n'ont absolument pas de contre-partie dans le calcul sans types inductifs primitifs.

On peut noter que les schémas d'élimination faibles reviennent, d'une certaine façon, à rajouter un peu de réécriture au langage de programmation sous-jacent. Il existe à ce titre divers travaux étudiant le mélange réécriture et λ-calcul typé (par exemple [145, 6].) L'élimination forte en revanche revient à introduire ces règles de réécriture au niveau des types.

Petits constructeurs

A partir de là, on comprend bien qu'en introduisant une telle extension, il faut être extrêmement prudent et éviter de faire le pas qui sépare le système cohérent du paradoxe. En particulier, on sait comment Girard a mis en évidence l'incohérence du premier système de Martin-Löf en y formalisant une version typée du paradoxe de Burali-Forti. Coquand a montré comment ce paradoxe pouvait être adapté dans le Calcul des Constructions avec sommes fortes [25]. Sans rentrer dans le détail, on peut dire que ce paradoxe peut être construit dans tout système où il est possible d'encapsuler un type dans un terme (d'un type particulier), puis de récupérer ce type. Un peu plus précisément, il suffit de disposer d'un type U : Set et de termes *encap* et *decap* tels que :

$$\begin{aligned} encap &: \text{Set} \to U \\ decap &: U \to \text{Set} \\ (decap\ (encap\ A)) &\rhd A \end{aligned}$$

Or notre schéma de types inductifs permet facilement de construire un type pouvant encapsuler tous les autres :

```
Inductive Definition U : Set =
 encap : Set -> U.
```

Si l'on autorise l'élimination forte sur ce type, il est facile de construire un terme *decap* vérifiant les conditions ci-dessus, et donc de formaliser le paradoxe de Girard. Pour éviter ce problème, Coquand et Paulin-Mohring ont proposé de restreindre l'élimination forte aux types inductifs dont les constructeurs ne prennent pas de prédicats en arguments. On les appelle alors *petits constructeurs*. Plus précisément, un type de constructeur C est dit petit si :

$$C \equiv (x_1 : T_1)\ \ldots\ (x_n : T_n)(I\ u_1\ \ldots\ u_k)$$

et les T_i sont tous des types (c'est-à-dire de type Set dans leur contexte respectif).

On voit bien que l'unique constructeur du type U ci-dessus n'est pas petit. Sa construction est donc légale, mais il ne lui correspond pas de schéma d'élimination forte. Il est donc possible d'y encapsuler n'importe quel type, mais on ne pourra plus extraire le type A de l'objet (*encap* A).

Cette restriction peut parfois paraître un peu *ad hoc* à première vue. Notre travail montre qu'elle est en fait suffisante. Il est également instructif et rassurant d'essayer d'appliquer notre preuve de normalisation au système sans cette restriction, et de vérifier à quel moment elle ne s'applique plus.

Notons également qu'en annexe, nous proposons une formalisation du paradoxe de Russell, plus simple que le paradoxe de Girard, en utilisant également les sommes fortes. Toutefois, notre formalisation nécessite une autre extension au système (qui semble en revanche inoffensive si elle est utilisée seule.) [4]

L'inversion de prédicats inductifs

La preuve de $0 \neq 1$ donnée ci-dessus peut être vue comme un cas particulier de ce que l'on appelle l'inversion des prédicats inductifs. Il s'agit de l'une des applications les plus courantes de l'élimination forte. L'idée est qu'étant donné un prédicat inductif sur un type inductif, on peut en général définir un prédicat équivalent en utilisant l'élimination forte. Prenons par exemple la définition suivante du prédicat de parité sur les entiers naturels :

```
Inductive Definition Even : nat -> Set =
  E0 : (Even 0)
| ES : (n:nat)(Even n)->(Even (S (S n))).
```

Si nous regardons maintenant une preuve (close) de (*Even* n), on comprend bien qu'elle est à peu près isomorphe à l'entier n. Aussi, il est facile de construire un prédicat équivalent à *Even* par récurrence structurelle sur l'entier. En notation "à la ML" cela donne par exemple :

```
Function Even_inv : nat -> Set =
  0 => True
| (S 0) => False
| (S (S n)) => (Even_inv n).
```

[4] L'autre intérêt de notre construction est son comportement calculatoire plus intéressant que celui du paradoxe de Girard.

Il est assez facile de construire une fonction équivalente en utilisant l'opérateur Nat_*recs*. On peut ensuite prouver par récurrence sur n (c'est-à-dire en utilisant Nat_*rec*) que $(Even\ n)$ et $(Even\_inv\ n)$ sont équivalents. L'intérêt de l'opération est bien sûr que $(Even\_inv\ (S\ O))$ se réduit en False, ce qui permet de prouver que 1 n'est pas pair. Cette opération est ce qu'on appelle *l'inversion* du prédicat *Even*. La preuve de $0 \neq 1$ correspond elle à une inversion possible du prédicat inductif "être égal à 0."

On peut aussi remarquer que dans toute preuve intuitive que 1 ne vérifie pas le prédicat inductif *Even*, on doit utiliser le fait que, par exemple, 0 n'est pas égal à 1, puisque sinon on ne pourrait être sûr de ce que le constructeur $E0$ ne permet pas de construire de preuve de $(Even\ (S\ O))$. On ne peut donc nier la parité de 1 qu'en discriminant entre les constructeurs de Nat, donc, dans le cas présent, en utilisant l'élimination forte.

Il existe de très nombreux exemples de preuves dans lesquelles l'inversion est nécéssaire. Aussi serait-il très utile de comprendre comment l'automatiser au moins partiellement. Dans[29], Thierry Coquand propose de remplacer, dans les règles de typage, les schémas d'élimination des types inductifs par un mécanisme de filtrage primitif qui fait intervenir une forme d'unification. L'un des attraits de cette alternative est justement que l'opération d'inversion devient pour ainsi dire "gratuite."

Réflexion

On a indiqué plus haut qu'il était possible de représenter, par exemple, le λ-calcul simplement typé en utilisant des définitions inductives. Une idée intéressante est alors d'utiliser l'élimination forte pour passer de la représentation des types simples comme objets inductifs aux véritables types du calcul qui leurs correspondent. Par exemple, en reprenant le type ST donné plus haut :

$$\begin{array}{rcl} TR & \equiv & [A:\mathrm{Set}](ST\_recs\ [T:ST]\mathrm{Set}\ A\ [T:ST][X:\mathrm{Set}][U:ST][Y:\mathrm{Set}]X \to Y) \\ & : & \mathrm{Set} \to ST \to \mathrm{Set} \end{array}$$

qui a le comportement calculatoire suivant :

$$\begin{array}{rcl} (TR\ A\ iota) & \triangleright & A \\ (TR\ A\ (Arr\ T\ U)) & \triangleright & (TR\ A\ T) \to (TR\ A\ U). \end{array}$$

On dispose ainsi d'une représentation d'une classe de types au niveau des preuves à partir de laquelle on peut revenir vers les types représentés.

On avait également donné une représentation des λ-termes simplement typés sous la forme du prédicat inductif $WT : context \to ST \to \mathrm{Set}$. Il est alors assez facile d'étendre la traduction ci-dessus à cette représentation en construisant un terme

$$tr \equiv (A:\mathrm{Set})(T:ST)(WT\ nil\ T) \to (TR\ A\ T)$$

qui traduit la représentation d'un λ-terme simplement typé clos vers le terme correspondant du système lui-même. Cette opération s'appelle la *réflexion* d'un fragment du système (ici le λ-calcul simplement typé).

On peut refaire ce genre de construction pour des fragments plus large que les types simples. Il ne semble pas possible en revanche de réfléchir un fragment avec types dépendants. On a alors besoin d'univers comme ceux proposés par Martin-Löf [98]. Comme remarqué par Mendler [102],

ces univers peuvent être vus comme un type inductif particulier munis d'élimination forte. Aussi, Dybjer [45] propose une extension du schéma de définitions inductives qui inclus les univers. Il serait utile d'ajouter cette extension à notre formalisme, et il semble qu'elle soit cohérente avec le Calcul des Constructions Inductives.

Chapitre 2

Le Système

Ce chapitre est consacré à la définition formelle du Calcul des Constructions Inductives et à l'énoncé de quelques propriétés syntaxiques de base. Nous essayons de choisir une présentation favorisant autant que faire se peut les preuves des deux propriétés essentielles de *normalisation forte* et de *confluence* énoncés dans le chapitre 4. Cette présentation s'inspire des notations de Christine Paulin-Mohring [124].

Les sept premières parties de ce chapitre ne comportent que des définitions :

- Dans la partie 1 on définit l'ensemble des termes du calcul, indépendamment du typage.
- Les parties 2 à 6 servent essentiellement à définir les règles de réduction et à introduire un certain nombre de notations.
- En 7 on donne les règles de typage proprement dites.

Ensuite, en 8, on définit de manière syntaxique et élémentaire les distinctions entre les différentes classes de termes (preuves, prédicats, etc) et on montre que cette classification est implicitement présente dans les règles du système. La partie 9 est la plus complexe et la plus longue. Les résultats essentiels qui y sont démontrés sont : une forme faible de confluence et la préservation du type par réduction (subject-reduction).

2.1 Les termes

Comme c'est l'usage, nous définissons les différentes algèbres utilisées par la donnée des grammaires correspondantes.

Depuis la thèse de Coquand [24], l'habitude a été prise, d'identifier syntaxiquement les abstractions de variables de type et celles de variables de termes. De même on utilise une seule algèbre commune aux types et aux termes. Dans ce qui suit, nous ferons de même mais en introduisant au préalable deux classes de variables, comme c'est d'ailleurs fait dans la présentation des "PTS" [56]. Dans le cas présent, cela permettera de distinguer syntaxiquement entre les termes de preuves, les prédicats, etc, simplifiant notablement certaines preuves, ainsi que la définition des "petits" et "grands" constructeurs.

Nous supposons donnés deux ensembles dénombrables et distincts, Vt, VT. Nous appellerons leurs éléments respectivement *variables de terme et variables de prédicat.*

On utilisera :

- les premières minuscules latines (a, b, c, etc) pour les variables de terme.
- les minuscules grecques (α, β, γ, etc) pour les variables de prédicat.

Les dernières lettres latines (x, y, z, X, Y, etc) seront utilisées pour désigner indifféremment les deux classes de variables. On a donc l'algèbre des variables :

$$x \ := \ a \mid \alpha$$

On se donne trois constantes Set, Type, Extern qu'on appellera *sortes*. Elles seront génériquement désignées par la lettre s :

$$s \ := \ \text{Set} \mid \text{Type} \mid \text{Extern}$$

Nous définissons l'algèbre de termes $\mathcal{L}$ par :

$$t \ := \ s \mid x \mid (x : t)t \mid [x : t]t \mid (t\ t) \mid \mathsf{Ind}(\alpha : t)\{\overrightarrow{t}\} \mid \mathsf{Constr}(n, t) \mid \mathsf{Elim}(t, t, \vec{t}, t)\{\overrightarrow{t}\}$$

où n désigne un nombre entier et $\vec{t}$ désigne l'algèbre des séquences de termes :

$$\vec{t} \ := \ \{\} \mid t :: \vec{t}$$

2.2 Notation vectorielle

La donnée des règles d'inférence définissant le Calcul des Constructions Inductives nécessite de nombreuses utilisations de points de suspension comme $(x_1 : t_1) \ldots (x_n : t_n)t$ ce qui nuit fortement à la clarté et la concision de la présentation. Pour y remédier partiellement nous utilisons une notation vectorielle à l'image de [124]. Cette idée est également déjà présente dans les travaux de de Bruijn avec la notion de *télescope* [35].

On commence par définir les séquences de variables :

$$\vec{x} \ := \ \{\} \mid x :: \vec{x}$$

Soient $\vec{x}$ et $\vec{u}$ deux séquences, respectivement de variables et de termes, de même longueur. Pour tout terme t on définit :

- le cas des séquences vides :
  $(\{\} : \{\})t \ = \ t$
  $[\{\} : \{\}]t \ = \ t$
  $(t\ \{\}) \ = \ t$
- si $\vec{x} = x :: \vec{x'}$ et $\vec{u} = u :: \vec{u'}$:
  $(\vec{x} : \vec{u})t \ = \ (x : u)(\vec{x'} : \vec{u'})t$
  $[\vec{x} : \vec{u}]t \ = \ [x : u][\vec{x'} : \vec{u'}]t$
  $(t\ \vec{u}) \ = \ ((t\ u)\ \vec{u'})$

Séquences indicées et conventions d'écriture

En pratique, on aura souvent à désigner le i-ème élément d'une séquence de termes ou de variables, c'est-à-dire à indicer les éléments d'une séquence.

Définition 2.1 *Soit n un entier naturel. On appelle $\mathfrak{I}(n)$ l'ensemble $\{i \in \mathbb{N}, 0 < i \leq n\}$. Etant donnée une séquence de termes $\vec{t}$ (respectivement une séquence de variables $\vec{x}$), on désigne par $\mathfrak{I}(\vec{t})$ (respectivement $\mathfrak{I}(\vec{x})$) l'ensemble $\mathfrak{I}(n)$ où n est la longueur de $\vec{t}$ (respectivement de $\vec{x}$).*

Soit n un entier naturel, et une famille de termes $(t_i)_{i\in\mathfrak{I}(n)}$. On peut immédiatement lui associer la séquence de termes

$$\vec{t_{i_{i\in\mathfrak{I}(n)}}} = t_1 :: t_2 :: \ldots :: t_n :: \{\}.$$

Inversement, toute séquence $\vec{t}$ s'écrit évidemment sous la forme $\vec{t_{ii\in\mathfrak{I}(n)}}$, où n est la longueur de $\vec{t}$. On écrira pour cela :

$$\vec{t} \equiv \vec{t_{i_{i\in\mathfrak{I}(n)}}}.$$

En général, on se permettra d'omettre l'ensemble d'indices, qui est défini sans ambiguïtés pour une séquence donnée. On se permettra ainsi de dire "soit $\vec{t_i}$, une séquence de termes". De plus, pour dire, par exemple, que chaque t_i vérifie une propriété P, on s'autorisera "$\forall i.P(t_i)$", pour $\forall i \in \mathfrak{I}(n).P(t_i)$ où n est la longueur de la séquence. Enfin, quand on considérera deux séquences comme $\vec{t_i}$ et $\vec{x_i}$, on signifiera implicitement qu'elles ont même longueur, et donc même ensemble d'indices. Par exemple si un terme

$$\mathsf{Elim}(\mathsf{Ind}(X : A)\{\overrightarrow{C}\}, Q, \vec{u}, t)\{\overrightarrow{f}\}$$

est bien formé, les séquences $\vec{C}$ et $\vec{f}$ sont forcément de la même longueur, ce que l'on indiquera en l'écrivant

$$\mathsf{Elim}(\mathsf{Ind}(X : A)\{\overrightarrow{C_i}\}, Q, \vec{u}, t)\{\overrightarrow{f_i}\}.$$

Il s'agit là d'un abus de notation évident, mais qui allège sensiblement l'écriture et ne semble pas ambigu en pratique, pour peu que l'on évite d'utiliser plusieurs fois les mêmes identificateurs.

Tout terme t s'écrit de manière unique sous la forme $(t_0\ \vec{t_i})$, t_0 n'étant pas une application. On écrira pour cela :

$$t \equiv (t_0\ \vec{t_i})$$

de même on définit les écritures :

$$t \equiv (\vec{x_i} : \vec{t_i})t_0 \text{ où } t_0 \text{ n'est pas un produit}$$

et

$$t \equiv [\vec{x_i} : \vec{t_i}]t_0 \text{ où } t_0 \text{ n'est pas une abstraction.}$$

2.3 Variables libres et substitution

Définition 2.2 (Variables libres) *Étant donné un terme t, on définit l'ensemble $FV(t)$ des variables libres de t par :*

- $FV(s) = \{\}$ *si s est une sorte*
- $FV(x) \equiv \{x\}$
- $FV((t_1\ t_2)) \equiv FV(t_1) \cup FV(t_2)$
- $FV(([x : t_1]t_2) \equiv FV(t_1) \cup (FV(t_2) \setminus \{x\})$
- $FV((x : t_1)t_2)) \equiv FV(t_1) \cup (FV(t_2) \setminus \{x\})$
- $FV(\mathsf{Ind}(\alpha : t)\{\overrightarrow{t}\}) \equiv FV(t) \cup (FV(\vec{t}) \setminus \{\alpha\})$
- $FV(\mathsf{Constr}(n, t)) \equiv FV(t)$
- $FV(\mathsf{Elim}(t_1, t_2, \{\overrightarrow{v}\}, t)\{\overrightarrow{u}\}) \equiv FV(t_1) \cup FV(t_2) \cup FV(\vec{v}) \cup FV(t) \cup FV(\vec{u})$

l'ensemble $FV(\vec{t})$ des variables libres d'une séquence de termes étant bien sûr défini par :

- $FV(\{\}) \equiv \emptyset$
- $FV(t :: \vec{t}) \equiv FV(t) \cup FV(\vec{t})$.

Si une variable x n'est pas élément de $FV(t)$, on dira alors qu'elle n'a pas **d'occurrences libres** *dans t.*

Étant donnés une variable x et deux termes t_1 et t_2, si x n'a pas d'occurrences libres dans t_2, on pourra éventuellement écrire $t_1 \to t_2$ à la place de $(x : t_1)t_2$.

Définition 2.3 (Substitution) *Étant donnés deux termes t et t' et une variable x, on définit le terme $t[x \setminus t']$ de la manière usuelle.*

Définition 2.4 (Substitution séquentielle) *Soit t un terme, $\vec{x}$ et $\vec{u}$, respectivement une séquence de variables et de termes, de même longueur. On définit le terme $t[\vec{x} \setminus \vec{u}]$ par :*

$$\begin{array}{lcl} t[[] \setminus []] & \equiv & t \\ t[x :: \vec{x} \setminus u :: \vec{u}] & \equiv & t[x \setminus u][\vec{x} \setminus \vec{u}] \end{array}$$

2.4 Occurrences positives et constructeurs

Nous formalisons maintenant les contraintes de positivité, présentées dans le chapitre 1 et nécessaires pour spécifier la forme des définitions inductives.

La notion d'arité décrit les types que peuvent avoir les définitions inductives :

Définition 2.5 (Arité) *Étant donné une sorte s, on appelle arités de sorte s l'ensemble des termes décrits par :*

$$Ar \ := \ s \mid (x : t)Ar.$$

Exemple On reprend les exemples du type inductif Nat des entiers naturels et du prédicat *Even* de parité déjà vus au chapitre 1. Le type de Nat sera l'arité Set, le type du prédicat inductif *even* sera l'arité Nat $\to$ Set.

Définition 2.6 (Occurrences positives) *Soit X une variable de prédicat. On dira qu'un terme t est* **strictement positif** *en X si $t \equiv (\vec{x} : \vec{t})(X\ \vec{t'})$, si X n'a pas d'occurrences libres dans $\vec{t}$ et $\vec{t'}$, et aucune des composantes de $\vec{x}$ n'est égale à X. On notera aussi $Pos(X, t)$.*

Définition 2.7 (Type de constructeur) *Un terme C est un type de constructeur en X si l'une des conditions suivantes est vérifiée :*

- *$C = (X\ \vec{t})$ sous la condition que X n'est pas libre dans $\vec{t}$*
- *$C = (x : t)D$ sous les conditions : D est un type de constructeur en X, X n'a pas d'occurrences libres dans t et $X \neq x$.*
- *$C = P \to D$ sous les conditions : D est un type de constructeur en X et $Pos(X, P)$.*

On allègera le formalisme en disant simplement que $C(X)$ est un type de constructeur ; on notera aussi $constr(C(X))$.

Exemple Le type Nat est défini comme $\mathsf{Ind}(X : \text{Set})\{X; X \to X\}$. Les notations 0 et S sont juste des abréviations pour respectivement $\mathsf{Constr}(1, \text{Nat})$ et $\mathsf{Constr}(2, \text{Nat})$. Le prédicat inductif *even* est défini comme

$$\mathsf{Ind}(X : \text{Nat} \to \text{Set})\{(X\ 0); (a : \text{Nat})(X\ a) \to (X\ (S\ (S\ a)))\}.$$

On vérifie bien que $X, X \to X, (X\ 0)$ et $(a : \text{Nat})(X\ a) \to (X\ (S\ (S\ a)))$ sont tous des types de constructeurs en X.

Définition 2.8 (petits et grands constructeurs) *Soit $C(X)$ un type de constructeur. On a $C(X) \equiv (\vec{x} : \vec{t})(X\ \vec{x'})$. Si $\vec{x}$ est une séquence de variables de terme, c.a.d. $\vec{x} = \vec{a}$, alors on dira que $C(X)$ est un* petit constructeur. *On écrira $Small(C(X))$. Si un constructeur n'est pas petit, il est* grand *; noté $Big(C(X))$.*

2.5 Dérivation des constructeurs

Nous avons maintenant les définitions nécessaires aux règles d'introduction des types inductifs. Il nous reste à préparer le terrain pour les règles d'élimination. Le définition suivante correspond aux types du (des) schéma(s) d'élimination associé(s) au type inductif. Rappelons que le type de chaque branche du schéma d'élimination dépend du type du constructeur correspondant. Cette opération, présentée informellement dans le chapitre 1 est appelée *dérivation du constructeur.*

Définition 2.9 *Soit $C(X)$, un type de constructeur et deux termes Q et c. On définit le terme $\Delta\{C(X), Q, c\}$ par récurrence sur la preuve que $C(X)$ est un type de constructeur :*

$\Delta\{(X\ \vec{t}), Q, c\}$	$\equiv$	$(Q\ \vec{t}\ c)$
$\Delta\{(x : t)D(X), Q, c\}$	$\equiv$	$(x : t)\Delta\{D(X), Q, (c\ x)\}$
$\Delta\{((\vec{x} : \vec{t})(X\ \vec{t'})) \to D(X), Q, c\}$	$\equiv$	$(p : (\vec{x} : \vec{t})(X\ \vec{t'}))((\vec{x} : \vec{t})((Q\ \vec{t'})\ (p\ \vec{x}))) \to \Delta\{D(X), Q, (c\ p)\}$

Enfin on notera $\Delta\{C(t), Q, c\}$ pour $\Delta\{C(X), Q, c\}[X \setminus t]$.

Exemple Les dérivés des types X et $X \to X$ des constructeurs des entiers naturels sont :

$$\begin{aligned} \Delta\{X, Q, c\} &\equiv (Q\ c) \\ \Delta\{X \to X, Q, c\} &\equiv (n : X)(Q\ n) \to (Q\ (c\ n)) \end{aligned}$$

et si l'on y substitue X par Nat et c par respectivement les constructeurs 0 et S, on retrouve les deuxième et troisième arguments du schéma d'élimination dépendant :

$$\mathrm{Nat\_}rec : (Q : \mathrm{Nat} \to \mathrm{Set})(Q\ 0) \to ((n : \mathrm{Nat})(Q\ n) \to (Q\ (S\ n))) \to (n : \mathrm{Nat})(Q\ n).$$

Enfin les définitions suivantes nous permettront de définir les réductions correspondant aux-dits schémas d'élimination :

Définition 2.10 *Soit $C(X)$, un type de constructeur en X et deux termes f et F. On définit le terme $\Delta[C(X), f, F]$ par récurrence sur la preuve que $C(X)$ est un type de constructeur :*

$\Delta[(X\ \vec{t}), f, F]$	$\equiv$	f
$\Delta[(x : t)D(X), f, F]$	$\equiv$	$[x : t]\Delta[D(X), (f\ x), F]$
$\Delta[((\vec{x} : \vec{t})(X\ \vec{t'})) \to D(X), f, F]$	$\equiv$	$[p : (\vec{x} : \vec{t})(X\ \vec{t'})]\Delta[D(X), (f\ p\ [\vec{x} : \vec{t}](F\ \vec{t'}\ (p\ \vec{x}))), F]$

Définition 2.11 *Soit A une arité de sorte S, deux termes $I \equiv \mathsf{Ind}(X : A)\{\overrightarrow{C(X)}\}$ et Q et une séquence de termes $\vec{f}$. On définit :*

$$\mathsf{Fun\_Elim}(I, Q, \vec{f}) \equiv [\vec{x} : \vec{A}][c : (I\ \vec{x})]\mathsf{Elim}(I, Q, \vec{x}, c)\{\overrightarrow{f}\}$$

2.6 Règles de réduction

Définition 2.12 (réduction) *On se donne les relations suivantes sur les termes, définies sous forme de règles de réécriture :*

$$\begin{array}{lll} ([x:t]t_1\ t_2) & \to_\beta & t_1[x \setminus t_2] \\ [x:t_1](t\ x) & \to_\eta & t \qquad\qquad Si\ x \notin FV(t) \\ \mathsf{Elim}(I,Q,\vec{a},(\mathsf{Constr}(i,I')\ \vec{m}))\{\overrightarrow{f_j}\} & \to_\iota & (\Delta[C_i(I),f_i,\mathsf{Fun\_Elim}(I,Q,\vec{f_j})]\ \vec{m}) \\ \multicolumn{3}{c}{Avec\ I = \mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}} \end{array}$$

On désigne par $\rhd_\beta$, $\rhd_\eta$ et $\rhd_\iota$, les relations correspondant à la réécriture d'un sous-terme par respectivement $\to_\beta$, $\to_\eta$ et $\to_\iota$. On utilise $\to$ et $\rhd$ pour les réunions de ces relations. De même on ecrit $\rhd^*$ et $\rhd^+$ (respectivement $\rhd_\beta^*$ etc) pour les fermetures réflexive-transitive et transitive de $\rhd$ (respectivement $\rhd_\beta$ etc.) et $=_{\beta\eta\iota}$ pour la fermeture réflexive, symétrique et transitive. On appellera chaîne de réductions partant de t *toute séquence $t_1 \dots t_n$ de termes tels que $t \rhd t_1 \rhd t_2 \dots \rhd t_n$. On s'autorisera éventuellement à parler de chaîne de réduction infinie.*

Exemple

$$\begin{array}{lll} [x:\mathrm{Nat}](S\ x) & \to_\eta & S \\ ([\alpha:\mathrm{Set}][x:\alpha]x\ (\beta:\mathrm{Set})\beta \to \beta\ [\beta:\mathrm{Set}][y:\beta]y & \rhd_\beta^* & [\beta:\mathrm{Set}][y:\beta]y) \\ \mathsf{Elim}(\mathrm{Nat},Q,[],(S\ t))\{t_0;t_S\} & \rhd_\iota & \\ & & ([n:\mathrm{Nat}](t_S\ n\ ([m:\mathrm{Nat}]\mathsf{Elim}(\mathrm{Nat},Q,[],m)\{t_0;t_S\}\ n))\ t) \\ \mathsf{Elim}(\mathrm{Nat},Q,[],(S\ t))\{t_0;t_S\} & \rhd^* & (t_S\ t\ \mathsf{Elim}(\mathrm{Nat},Q,[],t)\{t_0;t_S\}) \end{array}$$

Remarque Pour tous termes t et u, toute variable x et toutes séquences de termes $\vec{u}$ et $\vec{x}$ (de même longueur), on a :

$$(\lambda x :: \vec{x}.t\ u :: \vec{u}) \equiv (\lambda x.\lambda \vec{x}.t\ u\ \vec{u})$$

dont on peut déduire :

$$(\lambda x :: \vec{x}.t\ u :: \vec{u}) \rhd_\beta (\lambda \vec{x}.t[x \setminus u]\ \vec{u})$$
$$(\lambda \vec{x}.t\ \vec{u}) \rhd_\beta^* t[\vec{x} \setminus \vec{u}].$$

Lemme 2.1 *Si $t \rhd t'$, alors $FV(t') \subset FV(t)$.*

Définition 2.13 (Radical) *Un radical (respectivement un β-radical, η-radical, etc) est un terme t tel que $t \to_\beta t'$ (respectivement $t \to_\eta t'$, etc). Si $t \rhd t'$, le* radical reduit *est le sous-terme de t réécrit par $\to$. Les radicaux d'un terme sont tous ses sous-termes susceptibles d'être réecrits par $\to$.*

Définition 2.14 (Réduits d'un terme) *Soit t un terme. On appellera réduit de t (respectivement β-réduit, $\beta\iota$-réduit, etc) un terme t' tels que $t \rhd^* t'$ (respectivement $t \rhd_\beta^* t'$, $\rhd_{\beta\iota}^* t'$, etc).*

Définition 2.15 (Termes normaux) *Un terme t est dit normal (respectivement β-normal, ι-normal, etc) si aucun de ses sous-termes n'est un radical (respectivement un β-radical, ι-radical, etc). On dira aussi que t est* en forme normale. *Une forme normale de t est tout terme t' normal tel que $t \rhd^* t'$.*

Définition 2.16 (Termes normalisables) *Un terme t est dit normalisable, ou plus précisément* faiblement normalisable, *s'il admet une forme normale. Il sera dit* fortement normalisable *s'il n'existe pas de chaîne de réduction infinie partant de t.*

2.7 Les règles d'inférence

Définition 2.17 (contexte) *Un contexte est une séquence de paires formées d'une variable et d'un terme. On les décrira généralement par la lettre* Γ *ou d'autres majuscules grecques :*

$$\Gamma \ := \ [] \mid \Gamma :: (x, t)$$

Par ailleurs on écrira $\Gamma_1\Gamma_2$ pour la concaténation de deux contextes, on dira $x \in \Gamma$ pour dire que la variable x est liée dans Γ. Dans la suite, on n'utilisera que des contextes où chaque variable sera liée au plus une fois (on interdit $\Gamma :: (a, T_1) :: (a, T_2)$). Dans ce cas, $\Gamma(x)$ désignera le terme auquel x est lié dans Γ : si $\Gamma = \Delta_1 :: (a, T)\Delta_2$, alors $\Gamma(a) = T$.

On peut maintenant définir inductivement la relation de typage. Cette relation est définie entre un contexte et deux termes. On notera $\Gamma \vdash t_1 : t_2$ pour "t_1 est bien formé de type t_2 dans le contexte Γ." Dans cette première présentation des règles de typage, destinée à faciliter le plus possible l'étude méta-théorique du système, on définit également la relation de typage pour les séquences de termes ; si $\vec{t}$, $\vec{T}$, et $\vec{x}$ sont respectivement deux séquences de termes et une séquence de variables de même longueurs, on pourra considérer le jugement $\Gamma \vdash \vec{t} : (\vec{x} : \vec{T})$. Les règles de dérivation sont données dans la figure 2.7. À noter que les règles Prod et Lam sont paramétrées par une sorte s. On parlera donc de Prod-Set, Prod-Type, Lam-Extern, etc.

2.8 Classification des Termes

Alors que dans notre présentation informelle du système, nous distinguions entre preuves, prédicats, types, etc, la définition du système telle que donnée jusqu'ici ne connaît qu'un seul ensemble de termes. Partant des règles de typage, un premier pas est de distinguer les différents niveaux des termes qu'on manipule en *stratifiant* ainsi le système.

Définition 2.18 (Les cinq classes des termes) *On partitionne l'ensemble des termes en cinq parties : le singleton* {Extern}*, l'ensemble des termes de preuve* p*, l'ensemble des prédicats* Pr*, l'ensemble des ordres* K *et l'ensemble des schémas d'ordres* Ex*. Chaque terme* t *appartient à la partie* $\mathsf{Cl}(t)$*, définie comme suit :*

$$\begin{array}{lcl}
\mathsf{Cl}(\mathrm{Set}) & \equiv & \mathsf{K} \\
\mathsf{Cl}(\mathrm{Type}) & \equiv & \mathsf{Ex} \\
\mathsf{Cl}(a) & \equiv & \mathsf{p} \\
\mathsf{Cl}(\alpha) & \equiv & \mathsf{Pr} \\
\mathsf{Cl}(t\ t') & \equiv & \mathsf{Cl}(t) \\
\mathsf{Cl}([x : t]t') & \equiv & \mathsf{Cl}(t') \\
\mathsf{Cl}((x : t)t') & \equiv & \mathsf{Cl}(t') \\
\mathsf{Cl}(\mathsf{Ind}(X : A)\{\overrightarrow{C}\}) & \equiv & \mathsf{Pr} \\
\mathsf{Cl}(\mathsf{Constr}(i, I)) & \equiv & \mathsf{p} \\
\mathsf{Cl}(\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{f}\}) & \equiv & \mathsf{p} \textit{ si } \mathsf{Cl}(Q) = \mathsf{Pr} \textit{ et } \mathsf{Pr} \textit{ sinon.}
\end{array}$$

On définit par ailleurs la fonction suivante sur les classes de termes : $\begin{cases} \mathsf{lift}(\mathsf{p}) & \equiv & \mathsf{Pr} \\ \mathsf{lift}(\mathsf{Pr}) & \equiv & \mathsf{K} \\ \mathsf{lift}(\mathsf{K}) & \equiv & \mathsf{Ex} \end{cases}$

Nous avons besoin des lemmes techniques suivants :

Règles du Calcul des Constructions

$$(\text{Ax}_1)\ [] \vdash \text{Set} : \text{Type} \qquad (\text{Ax}_2)\ [] \vdash \text{Type} : \text{Extern}$$

$$(\textsc{Prod}\text{-}s)\ \frac{\Gamma :: (x : t_1) \vdash t_2 : s}{\Gamma \vdash (x : t_1)t_2 : s} \qquad (\textsc{Lam}\text{-}s)\ \frac{\Gamma \vdash (x : t_1)t_2 : s \qquad \Gamma :: (x : t_1) \vdash t : t_2}{\Gamma \vdash [x : t_1]t : (x : t_1)t_2}$$

$$(\text{W-Set})\ \frac{\Gamma \vdash t : \text{Set} \qquad \Gamma \vdash A : B \qquad x \notin \Gamma}{\Gamma :: (a : t) \vdash A : B} \qquad (\text{W-Type})\ \frac{\Gamma \vdash t : \text{Type} \qquad \Gamma \vdash A : B \qquad \alpha \notin \Gamma}{\Gamma :: (\alpha : t) \vdash A : B}$$

$$(\textsc{Var})\ \frac{\Gamma \vdash t_1 : t_2 \qquad (x : t) \in \Gamma}{\Gamma \vdash x : t} \qquad (\textsc{App})\ \frac{\Gamma \vdash t_2 : (x : T_1)T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash (t_2\ t_1) : T_2[x \setminus t_1]}$$

$$(\textsc{Conv})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_1 : s \qquad \Gamma \vdash T_2 : s \qquad T_1 =_{\beta\eta\iota} T_2}{\Gamma \vdash t : T_2}$$

Règles pour les types inductifs

$$(\textsc{Ind})\ \frac{Ar(A, \text{Set}) \qquad \Gamma \vdash A : \text{Type} \qquad \forall i.(\Gamma :: (X : A) \vdash C_i(X) : \text{Set}) \qquad \forall i.constr(C_i(X))}{\Gamma \vdash \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} : A}$$

$$(\textsc{Intro})\ \frac{\Gamma \vdash \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} : T}{\forall i.(\Gamma \vdash \mathsf{Constr}(n, \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}) : C_i(\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}))}$$

$$(\text{W-Elim})\ \frac{\begin{array}{c}A \equiv (\vec{x} : \vec{A})\text{Set} \qquad I = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} \qquad \Gamma \vdash \vec{u} : (\vec{x} : \vec{A}) \qquad \Gamma \vdash t : (I\ \vec{u}) \\ \Gamma \vdash Q : (\vec{x} : \vec{A})(I\ \vec{x}) \to \text{Set} \qquad \forall i.(\Gamma \vdash f_i : \Delta\{C_i(I), Q, \mathsf{Constr}(i, I)\})\end{array}}{\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{f_i}\} : (Q\ \vec{u}\ t)}$$

$$(\text{S-Elim})\ \frac{\begin{array}{c}A \equiv (\vec{x} : \vec{A})\text{Set} \qquad I = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} \qquad \Gamma \vdash \vec{u} : (\vec{x} : \vec{A}) \qquad \Gamma \vdash t : (I\ \vec{u}) \\ \Gamma \vdash Q : (\vec{x} : \vec{A})(I\ \vec{x}) \to \text{Type} \qquad \forall i.(\Gamma \vdash f_i : \Delta\{C_i(I), Q, \mathsf{Constr}(i, I)\}) \qquad \forall i.Small(C_i(X))\end{array}}{\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{f_i}\} : (Q\ \vec{u}\ t)}$$

Typage de séquences de termes

$$(\textsc{nil})\ \Gamma \vdash [] : ([] : [])$$

$$(\textsc{cons})\ \frac{\Gamma \vdash \vec{t} : (\vec{x} : \vec{T}) \qquad \Gamma(\vec{x} : \vec{T}[\vec{x} \setminus \vec{t}]) \vdash t : T[\vec{x} \setminus \vec{t}]}{\Gamma \vdash t :: \vec{t} : (x :: \vec{x} : T :: \vec{T})}$$

FIG. 2.1 – Les règles de dérivation

Lemme 2.2 *Si* $\mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}$ *apparaît comme sous terme d'un jugement dérivable, alors tous les* $C_i(X)$ *sont des types de constructeurs.*

PREUVE Immédiate par récurrence sur la structure de la dérivation. ■

Lemme 2.3 *Quels que soient les termes* Q *et* c *et le type de constructeur* $C(X)$*, on a* $\mathsf{Cl}(Q) = \mathsf{Cl}(\Delta[C(X), Q, c]$.

PREUVE Par récurrence immédiate sur la structure de $C(X)$. ■

Lemme 2.4 *Soient* t_1 *et* t_2 *deux termes et* x *une variable. Si* $\mathsf{Cl}(t_2) = \mathsf{Cl}(x)$*, alors* $\mathsf{Cl}(t_1[x \setminus t_2]) = \mathsf{Cl}(t_1)$.

PREUVE Immédiate par récurrence sur la structure de t_1. ■

Nous pouvons maintenant établir notre théorème qui donne son sens à cette classification :

Théorème 1 *Si le jugement* $\Gamma \vdash t_0 : t_1$ *est dérivable, alors on a* $\mathsf{Cl}(t_1) = \mathsf{lift}(\mathsf{Cl}(t_0))$*. En particulier* $t_0 \neq$ Extern*. De même pour tout couple* (x,t) *de* Γ *on aura* $\mathsf{Cl}(t) = \mathsf{lift}(\mathsf{Cl}(x))$.

PREUVE Par récurrence sur la dérivation du jugement. On renforce l'hypothèse de récurrence par la proposition suivante : si $[x:t]t'$ ou $(x:t)t'$ apparaissent comme sous-terme d'un jugement dérivable, alors $\mathsf{Cl}(t) = \mathsf{lift}(\mathsf{Cl}(x))$.

- La règle APP : l'hypothèse de récurrence nous assure que $\mathsf{Cl}(x) = \mathsf{Cl}(t_1)$, puisque $\Gamma \vdash t_1 : T_1$ est une prémisse de la règle. Le lemme précédent permet alors de conclure que

$$\mathsf{lift}(\mathsf{Cl}(t_2\ t_1)) = \mathsf{lift}(\mathsf{Cl}(t_2)) = \mathsf{Cl}((x:T_1)T_2) = \mathsf{Cl}(T_2) = \mathsf{Cl}(T_2[x \setminus t_1]).$$

- Le cas de la règle CONV est facile aussi : Comme $\Gamma \vdash T_1 : s$ et $\Gamma \vdash T_2 : s$, par hypothèse de récurrence, T_1 et T_2 appartiennent à la même classe.
- La règle IND : Comme $Ar(A, \mathrm{Set})$, on a bien $\mathsf{Cl}(A) = \mathsf{K}$.
- La règle INTRO : on a vu que $C_i(X)$ était un type de constructeur ; aussi $\mathsf{Cl}(C_i(X)) = \mathsf{Pr}$, et d'après le lemme précédent $\mathsf{Cl}(C_i(\mathsf{Ind}(X:A)C_i(\vec{X}))) = \mathsf{Pr}$.
- La règle W-ELIM : on a $\mathsf{Cl}(Q\ \vec{u}\ t) = \mathsf{Cl}(Q) = \mathsf{Pr}$.
- Le cas de la règle S-ELIM est similaire au précédent.

Les cas des autres règles sont triviaux. ■

2.9 Propriétés fondamentales du système

On commence maintenant l'étude des propriétés du système. Cette dernière partie du chapitre s'articule comme suit :

1. On commence par prouver un certain nombre de propriétés de confluence faible. Cette partie porte presque uniquement sur les termes et l'on n'y considère pas les règles de typage.
2. On prouve le lemme-clé dit *d'inversion*, qui relie la structure d'une dérivation à celle du terme typé.
3. On utilise alors les résultats précédents pour montrer que la réduction est correcte vis-à-vis du typage.

4. On donne une description syntaxique des différentes classes de termes.
5. On montre que tout ordre se β-réduit vers une arité, que l'on appelle sa forme ordre-normale.
6. On adapte à notre système la caractérisation syntaxique habituelle des formes normales.

Le travail effectué dans ce chapitre s'adapte essentiellement à tout système de type pur PTS, c'est-à-dire sans types inductifs, pour peu qu'il soit possible d'y construire une stratification comme celle décrite dans la partie précédente. Aussi une bonne partie de ce qui suit est fortement inspirée du travail de Geuvers et Nederhof [56] pour les PTS.

2.9.1 Confluence réduite - Lemme de Geuvers

Une des propriétés essentielles du système est la *confluence*, ou propriété de Church-Rosser : Si $\Gamma \vdash A : C$ et $\Gamma \vdash B : C$ avec $A =_{\beta\eta\iota} B$, alors il existe un terme D tel que $A \rhd^* D$ et $B \rhd^* D$. On verra par ailleurs qu'on a alors aussi $\Gamma \vdash D : C$. Malheureusement, si la preuve de cette propriété peut être faite directement pour le λ-calcul pur, il n'en va pas de même pour des calculs explicitement typés, comme celui que nous considérons ici. Ainsi, on peut facilement trouver deux termes t_1 et t_2, convertibles, mais qui n'admettent pas de réduit commun. C'est le contre-exemple bien connu dû à Nederpelt : soit t le terme $[x : A](([y : B]y)\ x)$. On a $t \rhd_\beta [x : A]x$ et $t \rhd_\eta [y : B]y$. Pour peu que A et B soient des variables distinctes, ces deux termes sont distincts, et tous deux en forme normale. Il n'admettent donc pas de réduit commun. Bien sûr, on verra plus loin que si A et B ne sont pas convertibles alors t n'est pas typable.

Dans le cas d'un système comme le notre, il est donc essentiel de faire intervenir les propriétés du système de type dans la preuve de confluence. Par exemple, ici, t n'est pas un terme bien-formé. En fait, nous prouverons la confluence à la fin de cette étude méta-théorique, après la preuve de normalisation forte. Toutefois, pour énoncer un certain nombre de propriétés plus élémentaires du Calcul des Constructions Inductives, il nous faut au moins une forme faible de confluence. C'est Herman Geuvers qui a su donner la bonne formulation de cette propriété. Dans ce paragraphe, nous la généralisons aux systèmes avec types inductifs.

Lemme 2.5 (Confluence de $\beta\iota$) *Soient trois termes t_1, t_2, t_3 tels que $t_1 \rhd^*_{\beta\iota} t_2$ et $t_1 \rhd^*_{\beta\iota} t_3$. Alors il existe un terme t_4 tel que $t_2 \rhd^*_{\beta\iota} t_4$ et $t_3 \rhd^*_{\beta\iota} t_4$.*

On utilise la méthode usuelle, dite de Tait–Martin-Löf, fondée sur la notion de *réduction parallèle*. En fait nous ne mentionnons cette propriété qu'à titre indicatif, car elle n'est pas utilisée par la suite. On pourra reprendre les preuves très similaires ci-dessous, ou celle donnée dans [149].

Lemme 2.6 (Non-pertinence des domaines) *Pour tous termes A, B, C et M, pour toutes variables x et y, on a :*

$$C[x \setminus [y : A]M] =_{\beta\eta} C[x \setminus [y : B]M]$$

et donc aussi

$$C[x \setminus [y : A]M] =_{\beta\eta\iota} C[x \setminus [y : B]M].$$

Preuve Il suffit de choisir une variable z qui ne soit pas libre dans M et A. On définit alors $t \equiv C[x \setminus [z : B](([y : A]M)\ z)]$ et on a bien $t \rhd^*_\eta C[x \setminus [y : A]M]$ et $t \rhd^*_\beta C[x \setminus [y : B]M]$. ■

Les termes démarqués

Comme la non-confluence est le seul fait de la présence d'annotations de type dans les λ-abstractions, on peut obtenir un calcul confluent en effaçant celles-ci. Cette propriété sera cruciale pour le lemme de Geuvers. Nous définissons ici ces "termes démarqués" et les règles de réduction correspondantes.

Définition 2.19 (termes démarqués) *On se donne une variable notée* _ *dont on admet qu'elle n'est jamais utilisée par ailleurs*[1]. *A tout terme t, on associe le terme $\|t\|$, défini comme suit :*

$$\begin{array}{lcl}
\|x\| & \equiv & x \\
\|s\| & \equiv & s \\
\|(t_1\ t_2)\| & \equiv & (\|t_1\|\ \|t_2\|) \\
\|[x:T]t\| & \equiv & [x:_]\|t\| \\
\|(x:T_1)T_2\| & \equiv & (x:\|T_1\|)\|T_2\| \\
\|\mathsf{Constr}(i,I)\| & \equiv & \mathsf{Constr}(i,\|I\|) \\
\|\mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}\| & \equiv & \mathsf{Ind}(X:\|A\|)\{\overrightarrow{\|C_i(X)\|}\} \\
\|\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{f_i}\}\| & \equiv & \mathsf{Elim}(\|I\|,\|Q\|,\|\vec{u}\|,\|t\|)\{\overrightarrow{\|f_i\|}\}
\end{array}$$

On notera $\|\mathcal{L}\|$ pour $\{\|t\|, t \in \mathcal{L}\}$.

Une conséquence immédiate du lemme précédent est :

Lemme 2.7 *Pour tout terme t, on a $\|t\| =_{\beta\eta} t$.*

On peut remarquer facilement que $\|\mathcal{L}\|$ est clos pour la β et la η-réduction. Il ne l'est pas pour la ι-réduction, car $\Delta[\|C\|, \|f\|, \|g\|]$ peut contenir à nouveau des λ-abstractions annotées. Pour remédier à ce détail, on définit la relation $\rhd_{\iota_0}$ par :

Définition 2.20

$$t \rhd_{\iota_0} \|t'\| \Leftrightarrow (t \rhd_\iota t' \wedge \|t\| \neq \|t'\|)$$

Il est évident que $\|\mathcal{L}\|$ est clos par $\rhd_{\beta\eta\iota_0}$. Par ailleurs :

Lemme 2.8 *Pour tous termes t, t', si $t \rhd_\beta t'$, alors $\|t\| \rhd_\beta \|t'\|$ ou $\|t\| = \|t'\|$. De même, si $t \rhd_\iota t'$ alors $\|t\| \rhd_{\iota_0} \|t'\|$ ou $\|t\| = \|t'\|$. Par ailleurs, pour tous termes t et t', si $\|t\| \rhd^*_{\beta\iota} \|t'\|$, alors il existe un terme u, tel que $t \rhd^*_{\beta\iota} u$ et $\|u\| = \|t'\|$.*

On faut remarquer que cette dernière propriété est fausse pour la η-réduction ; prenons par exemple le terme

$$t \equiv [x:\alpha]([y:\alpha][z:(\beta\ x)]z\ x)$$

qui est même bien-typé dans le contexte $[] :: (\alpha, \mathrm{Set}) :: (\beta, \alpha \to \mathrm{Set})$. On a bien $\|t\| \rhd_\eta [y:\_][z:\_]z$, mais t ne contient pas de η-radical[2].

Lemme 2.9 *La $\beta\iota_0\eta$-réduction est confluente sur $\|\mathcal{L}\|$.*

[1] La variable _ n'a pas de classe. Ceci ne posera pas de difficultés, les termes démarqués n'étant utilisés que pour prouver des résultats de confluence purement syntaxiques, indépendants du typage.

[2] On verra en fait, au chapitre 5, que cette propriété est vraie pour les termes bien-typés et $\beta\iota$-normaux.

On prouve ce lemme grace à la méthode des réductions parallèles, due à Tait et diffusée par Martin-Löf [96]. Bien qu'elle ne présente pas de grandes surprises, on détaillera la preuve. La notion de *réduction parallèle* est une relation binaire entre termes qui est comprise entre les relations $\triangleright_{\beta\iota 0\eta}$ et $\triangleright^*_{\beta\iota 0\eta}$; elle correspond à la réduction simultanée d'un nombre arbitraire de radicaux présents dans le terme de départ. Ici, on note cette relation $\twoheadrightarrow$:

Définition 2.21 (Réduction parallèle) *On définit $\twoheadrightarrow$ comme la plus petite relation binaire sur $\|\mathcal{L}\|$ vérifiant les clauses suivantes, dans lesquelles $u \twoheadrightarrow u'$, $v \twoheadrightarrow v'$, $I \twoheadrightarrow I'$, etc :*

$$\begin{array}{rcll}
u & \twoheadrightarrow & u & \\
(u\ v) & \twoheadrightarrow & (u'\ v') & \\
[x:_]u & \twoheadrightarrow & [x:_]u' & \\
\mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\} & \twoheadrightarrow & \mathsf{Ind}(X:A')\{\overrightarrow{C_i(X)'}\} & \\
\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{f_i}\} & \twoheadrightarrow & \mathsf{Elim}(I',Q',\vec{u}',t')\{\overrightarrow{f_i'}\} & \\
([x:_]u\ v) & \twoheadrightarrow & u'[x\setminus v'] & \\
[x:_](u\ x) & \twoheadrightarrow & u' & \text{si } x\notin FV(u) \\
\mathsf{Elim}(I,Q,\vec{u},(\mathsf{Constr}j,I\ \vec{m})) & \twoheadrightarrow & (\Delta[C_j(X)',f_j',\mathsf{Fun_Elim}(I',Q,\vec{f_i}')]\ \vec{m}') & \\
 & & & \text{avec } I\equiv\mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}
\end{array}$$

Une propriété importante est que $\twoheadrightarrow$ commute avec la substitution :

Lemme 2.10 *Soit une variable x et u, v, u' et v' de éléments de $\|\mathcal{L}\|$ tels que*

$$u \twoheadrightarrow u' \text{ et } v \twoheadrightarrow v'$$

alors

$$u[x\setminus v] \twoheadrightarrow u'[x\setminus v'].$$

PREUVE La preuve est par récurrence sur la structure de u, ou plus précisément sur la preuve de $u \twoheadrightarrow u'$

■

On aura besoin des deux propriétés simples suivantes :

Lemme 2.11 *Soit un terme $C \equiv (\vec{x_i} : \vec{U_i})(X\ \vec{t_j})$. Si C se réécrit en C' par $\triangleright$ (respectivement $\twoheadrightarrow, \triangleright_{\beta\eta\iota 0}, \triangleright_{\beta\iota 0}, \triangleright_\beta$, etc), alors $C' \equiv (\vec{x_i} : \vec{U_i'})(X\ \vec{t_j'})$ où tous les U_i et t_j se réécrivent en U_i' et t_j' par $\triangleright$ (respectivement $\twoheadrightarrow$ etc).*

PREUVE Immédiate par récurrence sur la structure de C. ■

Lemme 2.12 *Soient deux termes $C \equiv (\vec{x_i} : \vec{U_i})(X\ \vec{t_j})$ et $C' \equiv (\vec{x_i} : \vec{U_i'})(X\ \vec{t_j'})$ tels que tous deux se réécrivent en un terme C'' par $\triangleright$ (respectivement $\twoheadrightarrow, \triangleright_{\beta\iota 0,\eta}$, etc). Alors $C'' \equiv (\vec{x_i} : \vec{U_i''})(X\ \vec{t_j''})$ où pour tout i, U_i et U_i' se réécrivent tous deux en U_i'' et pour tout j, t_j et t_j' se réécrivent tous deux en t_j'' par $\triangleright$ (respectivement $\twoheadrightarrow$ etc).*

La encore la preuve est quasi-immédiate. En fait dans la suite, on se sert aussi des propriétés similaires pour $C \equiv \mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}$ mais les énoncer précisément ne présente que peu d'intérêt.

Le point important est que l'on voit facilement que $\triangleright \subset \twoheadrightarrow \subset \triangleright^*$; c'est-à-dire que pour tous termes t et t' :

$$t \triangleright t' \Rightarrow t \twoheadrightarrow t' \Rightarrow t \triangleright^* t'.$$

Il est bien connu que cette remarque suffit pour ramener la propriété de confluence de $\triangleright$ à la confluence forte de $\twoheadrightarrow$ (voir par exemple [70]). En d'autre termes, il nous reste à démontrer :

Lemme 2.13 (Confluence forte de $\twoheadrightarrow$) *Pour tous termes démarqués t, u et v,*

$$(t \twoheadrightarrow u \wedge t \twoheadrightarrow v) \Rightarrow \exists w. u \twoheadrightarrow w \wedge v \twoheadrightarrow w.$$

PREUVE La preuve est par récurrence sur la structure de t ; on ne considère par les cas symétriques de ceux déja traités.

1. Si $t = [x : \_](t_1\ x)$ avec $x \notin FV(t_1)$, on peut avoir :
   (a) $u = u_1$ et $v = v_1$ tels que
   $$t_1 \twoheadrightarrow u_1 \text{ et } t_1 \twoheadrightarrow v_1$$
   alors l'hypothèse de récurrence pour t_1 garantit l'existence de w_1 tel que $u_1 \twoheadrightarrow w_1$ et $v_1 \twoheadrightarrow w_1$. Il suffit alors de prendre $w = w_1$.
   (b) $u = u_1$ et $v = [x : \_](v_1\ x)$ avec
   $$t_1 \twoheadrightarrow u_1 \text{ et } t_1 \twoheadrightarrow v_1.$$
   Il existe alors un w_1 comme dans le cas précédent. De plus, comme $x \notin t_1$, on a aussi $x \notin FV(v_1)$ et donc $v \twoheadrightarrow w_1$. On prend donc $w = w_1$.
   (c) $u = [x : \_](u_1\ x)$ et $v = [x : \_](v_1\ x)$ avec
   $$t_1 \twoheadrightarrow u_1 \text{ et } t_1 \twoheadrightarrow v_1.$$
   Il existe alors un w_1 comme dans les cas précédents, et on peut prendre au choix $w = w_1$ ou $w = [x : \_](w_1\ x)$.
2. Si $t = [x : \_]t_1$ et t_1 n'est pas de la forme $(t_2\ x)$ avec $x \notin FV(t_2)$, on a forcément
   $$u = [x : \_]u_1 \text{ avec } t_1 \twoheadrightarrow u_1 v = [x : \_]v_1 \text{ avec } t_1 \twoheadrightarrow v_1$$
   l'hypothèse de récurrence sur t_1 assure donc l'existence de w_1 avec
   $$u_1 \twoheadrightarrow w_1 \text{ et } v_1 \twoheadrightarrow w_1.$$
   Il suffit alors de prendre $w = [x : \_]w_1$.
3. Si $t = ([x : \_]t_1\ t_2)$, on peut avoir :
   (a) $u = u_1[x \setminus u_2]$ et $v = v_1[x \setminus v_2]$ avec
   $$t_1 \twoheadrightarrow u_1 \text{ et } t_1 \twoheadrightarrow v_1$$
   $$t_2 \twoheadrightarrow u_2 \text{ et } t_2 \twoheadrightarrow v_2.$$
   L'hypothèse de récurrence pour t_1 et t_2 garantit respectivement l'existence de w_1 et w_2 tels que
   $$u_1 \twoheadrightarrow w_1 \text{ et } v_1 \twoheadrightarrow w_1$$
   $$u_2 \twoheadrightarrow w_2 \text{ et } v_2 \twoheadrightarrow w_2.$$
   Il suffit alors de prendre $w = w_1[x \setminus w_2]$ qui convient d'après le lemme 2.10.

(b) $u = ([x : \_]u_1\ u_2)$ et $v = v_1[x \setminus v_2]$ avec

$$t_1 \twoheadrightarrow u_1 \text{ et } t_1 \twoheadrightarrow v_1$$
$$t_2 \twoheadrightarrow u_2 \text{ et } t_2 \twoheadrightarrow v_2.$$

On prend alors le même w que ci-dessus.

(c) $u = ([x : \_]u_1\ u_2)$ et $v = ([x : \_]v_1\ v_2)$ avec

$$t_1 \twoheadrightarrow u_1 \text{ et } t_1 \twoheadrightarrow v_1$$
$$t_2 \twoheadrightarrow u_2 \text{ et } t_2 \twoheadrightarrow v_2.$$

On se donne alors w_1 et w_2 comme dans les deux cas ci-dessus et l'on prend $w = ([x : \_]w_1\ w_2)$.

4. $t = \mathsf{Elim}(I, Q, \vec{n}, (\mathsf{Constr}(j, I_0)\ \vec{m_k}))\{\overrightarrow{f_i}\}$, on peut avoir :

(a) $u = (\Delta[C_j(X)', f'_j, \mathsf{Fun\_Elim}(I', Q', \vec{f'_i})]\ \vec{m'_k})$
et
$$v = (\Delta[C_j(X)'', f''_j, \mathsf{Fun\_Elim}(I'', Q'', \vec{f''_i})]\ \vec{m''_k})$$
avec

$$\begin{array}{rcl} I' & \equiv & \mathsf{Ind}(X : A')\{\overrightarrow{C_i(X)'}\} \\ I'' & \equiv & \mathsf{Ind}(X : A'')\{\overrightarrow{C_i(X)''}\} \\ A & \twoheadrightarrow & A' \\ A & \twoheadrightarrow & A'' \\ C_i(X) & \twoheadrightarrow & C_i(X)' \\ C_i(X) & \twoheadrightarrow & C_i(X)'' \\ \text{etc.} & & \end{array}$$

On sait que I' et I'' (respectivement Q' et Q'', les $C_i(X)'$ et $C_i(X)''$) se réduisent tous deux en une étape de $\twoheadrightarrow$ vers un terme I''' (respectivement Q''' etc). Le lemme 2.12 permet alors aisément de vérifier qu'il est correct de prendre

$$w = (\Delta[C_j(X)''', f'''_j, \mathsf{Fun\_Elim}(I''', Q''', \vec{f'''_i})]\ \vec{m'''_k}).$$

(b) $u = (\Delta[C_j(X)', f'_j, \mathsf{Fun\_Elim}(I', Q', \vec{f'_i})]\ \vec{m'_k})$ et $v = \mathsf{Elim}(I'', Q'', \vec{n''}, (\mathsf{Constr}(j, I''_0)\ \vec{m''_k}))\{\overrightarrow{f''_i}\}$.
On définit alors $A''', C_i(X)''', I''', Q'''$ et m'''_k comme ci-dessus, et on prend

$$w = (\Delta[C_j(X)''', f'''_j, \mathsf{Fun\_Elim}(I''', Q''', \vec{f'''_i})]\ \vec{m'''_k}).$$

(c) $u = \mathsf{Elim}(I', Q', \vec{n'}, (\mathsf{Constr}(j, I'_0)\ \vec{m'_k}))\{\overrightarrow{f'_i}\}$ et $v = \mathsf{Elim}(I'', Q'', \vec{n''}, (\mathsf{Constr}(j, I''_0)\ \vec{m''_k}))\{\overrightarrow{f''_i}\}$.
Ce cas ne pose pas de problème.

■

Ceci clôt également la preuve du lemme 2.9

Corollaire 2.1 *Soient quatre termes A, A', B, B'. Si $(x : A)B =_{\beta\eta\iota} (x : A')B'$, alors $A =_{\beta\eta\iota} A'$ et $B =_{\beta\eta\iota} B'$.*

PREUVE D'après le lemme précédent, $(x : \|A\|)\|B\|$ et $(x : \|A'\|)\|B'\|$ admettent un réduit commun. Ce réduit ne peut être que de la forme $(x : U)V$, avec $\|A\| \rhd^* U$, $\|A'\| \rhd^* U$, $\|B\| \rhd^* V$ et $\|B'\| \rhd^* V$. Alors :

$$A =_{\beta\eta\iota} \|A\| =_{\beta\eta\iota} U =_{\beta\eta\iota} \|A'\| =_{\beta\eta\iota} A' \text{ et } B =_{\beta\eta\iota} \|B\| =_{\beta\eta\iota} V =_{\beta\eta\iota} \|B'\| =_{\beta\eta\iota} B'.$$

■

Corollaire 2.2 *Si u et v sont, soit deux sortes distinctes, soit deux variables distinctes, soit une variable et une sorte, alors $u \neq_{\beta\iota\eta} v$.*

Retard de la η-réduction

Lorsque l'on étudie le λ-calcul avec η-réduction, l'une des propriétés essentielles est le lemme de retard de la η-réduction :

$$t \rhd^*_{\beta\eta} t' \Rightarrow \exists t''.t \rhd^*_\beta t'' \rhd^*_\eta t'.$$

Dans le système que nous étudions ici, on aimerait avoir cette même propriété, en remplaçant β par $\beta\iota$. Hélas, cette fois encore, la propriété n'est vraie que pour les termes bien typés. Considérons par exemple,

$$\begin{array}{rcl} \mathsf{Elim}(\mathrm{Nat}, Q, [], [x : T](\mathsf{Constr}(1, \mathrm{Nat})\ x))\{f_0, f_S\} & \rhd_\eta & \mathsf{Elim}(\mathrm{Nat}, Q, [], \mathsf{Constr}(1, \mathrm{Nat}))\{f_0, f_S\} \\ & \rhd_\iota & f_0 \end{array}$$

où le premier η-redexe (mal-typé) $(\mathsf{Constr}(1, \mathrm{Nat})\ x)$ ne peut être retardé.

Il nous faut donc ici aussi nous livrer à quelques contorsions déplaisantes. La solution est heureusement plus facile que dans le cas de la confluence. Nous allons définir une notion de η'-réduction, qui dans le cas des termes bien typés sera équivalente à la ι-réduction, et pour laquelle nous aurons la propriété de retard de η, même pour les termes mal typés.

Définition 2.22 (ι'-réduction) *On définit la relation de ι'-réduction sur les termes par :*

$$\mathsf{Elim}(I, Q, \vec{u}, [\vec{x}_i : \vec{A}_i](\mathsf{Constr}(k, I)\ \vec{m}\ \vec{t_i}))\{\overrightarrow{f_j}\} \rightarrow_{\iota'} (\Delta[C_k(I), f_k, \mathsf{Fun\_Elim}(I, Q, \vec{f}, k)]\ \vec{m})$$

si $I = \mathsf{Ind}(X : A)\{\overrightarrow{C_j(X)}\}$ et pour tout i, $\|t_i\| \rhd^*_\eta x_i$, et $x_i \notin FV(\|\vec{m}\|) \cup FV(\|I\|)$.

A partir de là, on définit comme d'habitude les relations $\rhd_{\iota'}$, $\rhd^*_{\iota'}$, $\rhd_{\beta\iota'}$, etc.

Lemme 2.14 *Si $t \rhd^*_{\beta\eta\iota'} t'$ alors $FV(t') \subset FV(t)$.*

Lemme 2.15 *Quelques soient les termes t et t',*

$$t \rhd_\iota t' \Rightarrow t \rhd_{\iota'} t'$$

et

$$t =_{\beta\eta\iota} t' \Leftrightarrow t =_{\beta\eta\iota'} t'$$

Lemme 2.16 *Soient trois termes t, t' et t''. Si $t \rhd_\eta t' \rhd_{\beta\iota'} t''$, alors soit $t \rhd_{\beta\iota'} t''$, soit il existe t''' tel que $t \rhd_{\beta\iota'} t''' \rhd^*_\eta t''$.*

PREUVE Dans le cadre du λ-calcul pur, Barendregt présente une preuve relativement élégante dans [8], mais elle utilise une extension du calcul. A ce point, il nous semble indécent d'infliger au lecteur l'introduction d'encore un autre calcul. Aussi nous nous contenterons d'une simple récurrence sur la structure du terme t. C'est un peu monotone, certes, mais à notre sens un moindre mal.
Les cas $t = x$, $t = (x : T_1)T_2$ et $t = \mathsf{Constr}(i, I)$ sont immédiats.
Si $t = \mathsf{Elim}(I, Q, \vec{u}, t_1)\{\overrightarrow{f_i}\}$, le seul cas qui ne soit pas trivialement traité par l'hypothèse de récurrence est $t_1 = [x : T](t_2\ x)$ et $t \rhd_\eta t' = \mathsf{Elim}(I, Q, \vec{u}, t_2)\{\overrightarrow{f_i}\}$ avec

$$t_2 = [\vec{x}_j : \vec{A}_j](\mathsf{Constr}(i, I)\ \vec{m}\ \vec{t}_j)$$

et

$$t' \rhd_{\iota'} t'' = (\Delta[C_i(I), f_i, \mathsf{Fun\_Elim}(I, Q, \vec{f_i})]\ \vec{m}).$$

Mais alors il est facile de voir que l'on a directement $t \rhd_{\iota'} t''$.
Le cas $t = (t_1\ t_2)$ est comme d'habitude un peu plus complexe :

- Si $t' = (t'_1\ t_2)$ avec $t_1 \rhd_\eta t'_1$ c'est immédiat. Il suffit d'utiliser l'hypothèse de récurrence.
- Si $t' = (t_1\ t'_2)$ avec $t_2 \rhd_\eta t'_2$ c'est immédiat aussi, sauf si $t_2 = [x : T]t_3$ et $t'' = t_3[x \setminus t'_2]$. Mais alors il suffit de prendre $t''' = t_3[x \setminus t_2]$ et d'observer qu'on a alors $t''' \rhd^*_\eta t''$; la longueur de la chaîne de réductions étant égale au nombre d'occurrences de x dans t_3.

Enfin on considère le cas $t = [x : T]t_1$ qui est le plus subtil. Le seul sous-cas qui ne soit pas traité immédiatement par application de l'hypothèse de récurrence est $t_1 = (t_2\ x)$ avec $t' = t_2$ et donc $t_2 \rhd_{\beta\iota'} t''$. On peut alors remarquer que comme d'après le lemme 2.14 $FV(t'') \subset FV(t_2)$ on a :

$$[x : T](t_2\ x) \rhd_{\beta\iota'} [x : T](t''\ x) \rhd_\eta t''$$

et il suffit donc de choisir $t''' = [x : T](t''\ x)$. ■

Lemme 2.17 (Retard de la η-réduction) *Quelques soient les termes t et t', si*

$$t \rhd^* t'$$

alors il existe un terme t'' tel que

$$t \rhd^*_{\beta\iota'} t'' \rhd^*_\eta t'.$$

PREUVE On raisonne par récurrence sur le nombre n de β et ι'-réductions dans la chaîne qui va de t à t'. Le cas $n = 0$ est trivial. Si $n \geq 1$, alors pour un certain entier naturel p, on a :

$$t \rhd^p_\eta t_1 \rhd_{\beta\iota'} t_2 \rhd^* t'$$

la chaîne allant de t_2 à t' comportant exactement $n - 1$ β et ι-réductions.

En appliquant p fois le lemme précédent, on établit l'existence d'un terme t_3, tel que $t \rhd_{\beta\iota'} t_3 \rhd^*_\eta t_2$. On a donc une chaîne allant de t_3 à t' en $n - 1$ β et ι-réductions. L'hypothèse de récurrence nous garantit alors qu'il existe un terme t_4, tel que $t_3 \rhd^*_{\beta\iota'} t_4 \rhd^*_\eta t'$. Comme $t \rhd_{\beta\iota'} t_3$, il s'ensuit que $t \rhd^*_{\beta\iota'} t_4 \rhd^*_\eta t'$. ■

Il nous faut malheureusement également définir la contre-partie de la ι'-réduction pour les termes démarqués. On définit :

$$(t \rhd'_\iota t' \wedge \|t\| \neq \|t'\|) \Leftrightarrow t \rhd_{\iota'_0} \|t'\|$$

avec la propriété attendue :

Lemme 2.18 *Soient deux termes t et t', tels que $t \rhd_{\iota'} t'$. Alors soit $\|t\| = \|t'\|$, soit $\|t\| \rhd_{\iota'_0} \|t'\|$.*
Par ailleurs, si $\|t\| \rhd_{\iota'_0} \|t'\|$, alors $t \rhd_{\iota'} t'$.

Le lemme de Geuvers

Ce lemme, que nous pouvons énoncer maintenant, correspond à une forme faible de confluence, qui sera suffisante pour prouver les deux premières propriétés essentielles du système de type : l'unicité du type et la correction de la réduction vis-à-vis du typage. Il dit en substance que tout terme convertible avec un terme en forme normale de tête se $\beta\iota'$-réduit vers une forme η-expansée de cette forme normale de tête. La présence de types inductifs ne change ni sa preuve, ni son énoncé. Il y a juste plus de cas possibles pour la forme normale de tête.

Lemme 2.19 (Geuvers)

- *Si* $Q =_{\beta\eta\iota} (x : P_1)P_2$, *alors :*

$$Q \rhd^*_{\beta\iota'} [\vec{y_i} : \vec{A_j}](((x : Q_1)Q_2)\ \vec{t_i})$$

  avec $Q_1 =_{\beta\eta\iota} P_1$, $Q_2 =_{\beta\eta\iota} P_2$ *et pour tout* i, $t_i \rhd^*_\eta y_i$.
- *Si* $Q =_{\beta\eta\iota} (x\ \vec{P_i})$ *alors*

$$Q \rhd^*_{\beta\iota'} [\vec{y_j} : \vec{A_j}](x\ \vec{Q_i}\ \vec{t_j})$$

  avec pour tout i, $Q_i =_{\beta\eta\iota} P_i$ *et pour tout* j, $t_j \rhd^*_\eta y_j$.
- *Si* $Q =_{\beta\eta\iota} (\mathsf{Constr}(i, I)\ \vec{P_i})$ *alors*

$$Q \rhd^*_{\beta\iota'} [\vec{y_j} : \vec{A_j}](\mathsf{Constr}(i, I')\ \vec{Q_i}\ \vec{t_j})$$

  avec pour tout i, $Q_i =_{\beta\eta\iota} P_i$ *et pour tout* j, $t_j \rhd^*_\eta y_j$ *et* $I =_{\beta\eta\iota} I'$.
- *Si* $Q =_{\beta\eta\iota} (\mathsf{Ind}(X : A)\{\overrightarrow{C_k}\}\ \vec{P_i})$ *alors*

$$Q \rhd^*_{\beta\iota'} [\vec{y_j} : \vec{A_j}](\mathsf{Ind}(X : A')\{\overrightarrow{C'_k}\}\ \vec{Q_i}\ \vec{t_j})$$

  avec $A =_{\beta\eta\iota} A'$, *pour tout* k, $C_k =_{\beta\eta\iota} C'_k$, *pour tout* i, $Q_i =_{\beta\eta\iota} P_i$ *et pour tout* j, $t_j \rhd^*_\eta y_j$.
- *Si* $Q =_{\beta\eta\iota} (\mathsf{Elim}(I, Q, \vec{u_l}, t)\{\overrightarrow{F_k}\}\ \vec{P_i})$, *si* t *n'est convertible avec aucun terme de la forme* $(\mathsf{Constr}(l, J)\ \vec{U})$, *alors*

$$Q \rhd^*_{\beta\iota'} [\vec{y_j} : \vec{A_j}](\mathsf{Elim}(I', Q', \vec{u_l}', t')\{\overrightarrow{F'_k}\}\ \vec{Q_i}\ \vec{t_j})$$

  avec $I =_{\beta\eta\iota} I'$, $Q =_{\beta\eta\iota} Q'$, $t =_{\beta\eta\iota} t'$, *pour tout* i, $Q_i =_{\beta\eta\iota} P_i$, *pour tout* k, $F_k =_{\beta\eta\iota} F'_k$, *pour tout* l, $u_l =_{\beta\eta\iota} u'_l$ *et pour tout* j, $t_j \rhd^*_\eta y_j$.

PREUVE Les cinq cas sont similaires. A chaque fois on raisonne par récurrence sur la longueur du chemin qui va de Q à P. On détaillera simplement le cas $P = (x\ \vec{P_i})$. Par hypothèse de récurrence, il existe Q', tel que

$$Q' \rhd^*_{\beta\iota} [\vec{y_j} : \vec{A_i}](x\ \vec{Q'_i}\ \vec{t_j})$$

et $Q \rhd^1 Q'$ ou $Q' \rhd^1 Q$.

- Le cas $Q \rhd_{\beta\iota} Q'$ est immédiat.
- Si $Q' \rhd_{\beta\iota} Q$, on sait, grâce à la confluence de $\beta\iota'$, qu'il existe Q'' réduit commun de Q et Q' par $\beta\iota'$. Il est facile de voir que Q'' est bien de la forme requise.

– Si $Q \rhd_\eta Q'$, le lemme de retard de la η-réduction nous assure qu'il existe Q'' tel que

$$Q \rhd^*_{\beta\iota'} Q'' \rhd^*_\eta [\vec{y_j} : \vec{A_i}](x\ \vec{Q'_i}\ \vec{t_j}).$$

Là aussi, on voit facilement que Q'' est bien de la forme requise.

– le cas le plus délicat est $Q' \rhd_\eta Q$. La confluence de $\beta\iota'_0\eta$ sur $\|\mathcal{L}\|$ nous assure que $\|Q\| \rhd^* (x\ \vec{q_i})$ avec pour tout i, $\|Q_i\| \rhd^*_{\beta\eta\iota'_0} q_i$. Le lemme de retard de η nous permet d'en déduire :

$$\|Q\| \rhd^*_{\beta\iota'_0} [\vec{y_j} : \_](x\ \vec{q'_i}\ \vec{u_j}) \rhd^*_\eta (x\ \vec{q_i})$$

avec pour tout j, $u_j \rhd^*_\eta y_j$ et pour tout i, $q'_i \rhd^*_\eta q_i$.
On en déduit l'existence d'un terme $Q_0 \equiv [\vec{y_j} : \vec{A_j}](c\ \vec{Q_i}\ \vec{v_j})$ avec

$$Q \rhd^*_{\beta\iota'} Q_0 \text{ et } \|Q_0\| = [\vec{y_j} : \_](x\ \vec{q_i}\ \vec{u_j}).$$

On a donc pour tout i, $\|Q_i\| = q_i$ et donc $Q_i =_{\beta\eta\iota} q_i =_{\beta\eta\iota} t_i =_{\beta\eta\iota} P_i$.

■

Nous n'utiliserons plus les termes démarqués jusqu'à la dernière partie du chapitre 4.

2.9.2 De la forme des termes bien typés

Les résultats qui suivent sont tous classiques et bien connus pour les PTS fonctionnels avec β-conversion. La présence de la η-conversion nous oblige à remanier quelque peu l'ordre de leur présentation. Dans [53], Geuvers indique succinctement comment adapter les démonstrations de [56]. Il a ensuite détaillé une méthode un peu différente dans sa thèse [55]. Ce que nous proposons ici ressemble beaucoup à la première version, mais nous semble un peu plus simple. Il faut remarquer que la présence des types inductifs ne change pas l'ordre des lemmes, et peu les démonstrations. Par ailleurs, tout ce qui suit s'adapte très facilement à tout PTS extensionnel (avec η-conversion).

Lemme 2.20 (Variables libres) *Si le jugement $\Gamma \vdash t_1 : t_2$ est dérivable, alors $FV(t_1) \cup FV(t_2) \in \mathcal{V}(\Gamma)$.*

Preuve La preuve est immédiate par récurrence structurelle sur la dérivation. ■

Lemme 2.21 (Sous-termes) *Tout sous-terme de tout terme bien-formé est bien formé.*

Lemme 2.22 (Substitution) *Si les jugements suivants sont dérivables*

$$\Gamma_1 :: (x : A)\Gamma_2 \vdash u : B \quad (1)$$
$$\Gamma_1 \vdash v : A \quad (2)$$

alors on peut dériver :

$$\Gamma_1\Gamma_2 \vdash u[x \setminus v] : B[x \setminus v]$$

Preuve On raisonne par récurrence sur la structure de la dérivation de 2. ■

Lemme 2.23 (Affaiblissement) *Si les jugements suivants sont dérivables*

$$\Gamma \vdash u : A \tag{3}$$
$$\Gamma' \vdash v : B \tag{4}$$

avec $\Gamma \subset \Gamma'$*, alors on peut dériver :*

$$\Gamma' \vdash u : A$$

PREUVE Par récurrence sur la structure de la dérivation de 4. ■

Lemme 2.24 *Si le jugement* $\Gamma_1 :: (x, T)\Gamma_2 \vdash A : B$ *est dérivable, alors* $\Gamma_1 \vdash T : S$ *est dérivable, où* S *est la sorte associée à* x*. De plus on a aussi* $\Gamma_1 :: (x, T)\Gamma_2 \vdash T : S$*.*

PREUVE Par récurrence sur la structure de la dérivation. ■

Lemme 2.25 *Si :*

$$\Gamma \vdash (x : T_1)T_2 : S \tag{5}$$

alors

$$\Gamma :: (x, T_1) \vdash T_2 : S.$$

PREUVE Par récurrence sur la dérivation de 5. Celle-ci se termine par une règle PROD, W ou CONV. Les deux premiers cas sont immédiats ; dans le cas de la règle CONV, l'hypothèse de récurrence nous assure $\Gamma :: (x, T_1) \vdash T_2 : S'$ où S' est une sorte et $S =_{\beta\eta\iota} S'$. On a vu qu'alors $S = S'$. ■

Lemme 2.26 *Si :*

$$(1)\quad \Gamma \vdash \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} : T$$

alors $T =_{\beta\eta\iota} A$*,* $\Gamma \vdash A : \mathrm{Type}$ *et pour tout* i *on a* $\Gamma :: (X, A) \vdash C_i(X) : \mathrm{Set}$*.*

PREUVE Par récurrence sur la dérivation de (1). Celle-ci se termine par une règle IND, W ou CONV. Les trois cas sont immédiats. ■

Lemme 2.27 *Si le jugement* $\Gamma \vdash t : T$ *est dérivable, alors soit* $T = \mathrm{Extern}$*, soit* $\Gamma \vdash T : S$ *pour une certaine sorte* S*.*

PREUVE C'est encore une récurrence sur la structure de la dérivation. Les cas des règles AX, PROD, LAM, W, CONV, IND et ELIM sont immédiats. Dans le cas APP, si $t = (t_2\ t_1)$ et $T = T_2[x \setminus t_1]$, on sait que $\Gamma \vdash (x : T_1)T_2 : S$, ce qui d'après le lemme 2.25 implique $\Gamma :: (x, T_1) \vdash T_2 : S$, et donc $\Gamma \vdash T_2[x\backslash t_1] : S$ par le lemme de substitution. Dans le cas de INTRO, si $t = \mathsf{Constr}(j, \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\})$ alors $T = C_j(\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\})$ et le lemme 2.26 nous assure que $\Gamma \vdash \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} : A$ et $\Gamma :: (X, A) \vdash C_j(X) : \mathrm{Set}$; par substitution on a alors bien $\Gamma \vdash T : \mathrm{Set}$. ■

Lemme 2.28 (Inversion) *Si le jugement* $\Gamma \vdash t : T$ *est dérivable, alors :*

(i) $t = a$ $\Rightarrow$ $a \in \Gamma$
$T =_{\beta\eta\iota} \Gamma(a)$
$\Gamma \vdash T : \text{Set}$

(ii) $t = \alpha$ $\Rightarrow$ $\alpha \in \Gamma$
$T =_{\beta\eta\iota} \Gamma(\alpha)$
$\Gamma \vdash T : \text{Type}$

(iii) $t = \text{Set}$ $\Rightarrow$ $T =_{\beta\eta\iota} \text{Type}$

(iv) $t = \text{Type}$ $\Rightarrow$ $T =_{\beta\eta\iota} \text{Extern}$

(v) $t \neq \text{Extern}$

(vi) $t = (x : T_1)T_2$ $\Rightarrow$ $\Gamma \vdash T_1 : s_1$
$\Gamma :: (x : T_1) \vdash T_2 : s_2$
$T =_{\beta\eta\iota} s_2$
avec $s_1 \in \{\text{Set}, \text{Type}\}$ *et* s_2 *une sorte quelconque*

(vii) $t = [x : T_1]t_2$ $\Rightarrow$ $\Gamma \vdash T_1 : s_1$
$\Gamma :: (x : T_1) \vdash t_2 : B$
$\Gamma :: (x : T_1) \vdash B : s_2$
$\Gamma \vdash T : s_2$
$T =_{\beta\eta\iota} (x : T_1)B$
ou s_1 *est la sorte de* x *et* $s_2 \in \{\text{Set}, \text{Type}\}$

$(viii)$ $t = (t_1\ t_2)$ $\Rightarrow$ $\Gamma \vdash t_1 : (x : T_2)T_1$
$\Gamma \vdash t_2 : T_2$
$T =_{\beta\eta\iota} T_1[x \setminus t_2]$

(ix) $t = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$ $\Rightarrow$ $A \equiv (\vec{x} : \vec{A})\text{Set}$
$\forall i.\mathsf{Constr}(C_i(X))$
$\forall i.\Gamma :: (X : A) \vdash C_i(X) : \text{Set}$

(x) $t = \mathsf{Constr}(i, I)$ $\Rightarrow$ $I = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$
avec les mêmes conditions sur I *que ci-dessus*

(xi) $t = \mathsf{Elim}(I, Q, \vec{a}, t)\{\overrightarrow{f_i}\}$ $\Rightarrow$ $I = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$
avec les mêmes conditions sur I *que ci-dessus et :*
$\Gamma \vdash Q : (\vec{x} : \vec{A})(I\ \vec{x}) \to s$ *avec* $s \in \{\text{Set}, \text{Type}\}$
$\Gamma \vdash T : s$
$\Gamma \vdash t : (I\ \vec{a})$
$\forall i.\Gamma \vdash f_i : \Delta[C_i(X), Q, \mathsf{Constr}(i, I)]$
$T =_{\beta\eta\iota} (Q'\ \vec{a}\ t)$
de plus $s = \text{Type} \Rightarrow \forall i.\mathit{Small}(C_i(X))$

PREUVE Il s'agit bien sûr d'une nouvelle récurrence sur la structure de la dérivation. Remarquons toutefois que les cas (i) et (ii) sont par ailleurs conséquences de lemmes déjà énoncés. Les cas des règles AX, PROD, LAM, W, VAR, APP, IND sont immédiats. La règle CONV ne pose pas de problèmes non plus ; pour (xi) il faut juste remarquer que $\text{Type} \neq_{\beta\iota\eta} \text{Set}$. Les cas des règles INTRO et ELIM se traitent également facilement grâce à l'hypothèse de récurrence appliquée aux prémisses. ■

Lemme 2.29 (Unicité du type) *Si* $\Gamma \vdash t : A$ *et* $\Gamma \vdash t : A'$, *alors* $A =_{\beta\eta\iota} A'$.

PREUVE Par récurrence structurelle sur t. Pour chaque cas, on utilise la clause correspondante du lemme d'inversion. ■

Corollaire 2.3 *Il n'existe pas de terme bien formé, dont un des sous-termes est de la forme*

$$\mathsf{Elim}(I, Q, \vec{u}, [x : T]t)\{\overrightarrow{f}\}.$$

PREUVE S'il en était ainsi, ce sous-terme serait lui aussi bien formé. Alors, d'après le lemme d'inversion $[x : T]t$ serait bien formé de type $(\mathsf{Ind}(X : A)\{\overrightarrow{C}\}\ \vec{u})$ dans un certain contexte. Mais le lemme d'inversion nous assure par ailleurs qu'alors $[x : T]t$ est bien typé de type $(x : T)B$ dans ce même contexte, et alors $(x : T)B =_{\beta\eta\iota} (\mathsf{Ind}(X : A)\{\overrightarrow{C}\}\ \vec{u})$, ce qui est en contradiction avec le lemme de Geuvers. ■

Une conséquence immédiate est qu'on a plus besoin de la notion de ι'-réduction :

Corollaire 2.4 *Soit t un terme bien formé. Si $t \rhd_{\iota'} t'$, alors $t \rhd_{\iota} t'$.*

2.9.3 Réductions sur les termes bien-formés

C'est bien sûr l'une des premières caractéristiques demandées à un système de type, que le type d'un terme reste inchangé si l'on réduit ce terme. C'est la propreté de correction des réductions vis-à-vis du typage, souvent désignée par son nom anglais de *subject réduction*. C'est sa preuve qui suit.

Lemme 2.30 (Correction de la $\beta\iota$-réduction) *Si le jugement $\Gamma \vdash t : A$ est dérivable, et si $t \rhd^1_{\beta\iota} t'$ et $\Gamma \rhd^1_{\beta\iota} \Gamma'$, alors les jugements suivants sont dérivables :*

$$\Gamma \vdash t' : A$$

$$\Gamma' \vdash t : A$$

PREUVE Par récurrence sur la taille de la dérivation de $\Gamma \vdash t : A$. La plupart des cas sont immédiats et nous ne traitons que les plus délicats :

- La règle VAR : La réduction se fait forcément dans Γ ; si ce n'est pas dans le type associé à x dans Γ, il n'y a pas de problèmes. Sinon on a une dérivation de $\Gamma \vdash A : B$ avec $\Gamma = \Gamma_1 :: (x, t)\Gamma_2$ et $\Gamma' = \Gamma_1 :: (x, t')\Gamma_2$; elle contient donc une sous-dérivation de $\Gamma_1 \vdash t : S$ se qui permet d'assurer $\Gamma' \vdash t : S$. Comme $\Gamma' \vdash x : t'$ et $t =_{\beta\eta\iota} t'$, on a bien $\Gamma' \vdash x : t$.
- La règle IND : Il suffit de vérifier que le $\beta\iota$-réduit d'un type de constructeur (respectivement d'une arité) est également un type de constructeur (respectivement une arité). Le reste suit par simple application de l'hypothèse de récurrence.
- La règle APP : Elle ne pose pas de problèmes particuliers pour la ι-réduction, mais c'est bien sûr le cas crucial pour la β-réduction. On prend $t = (t_1\ t_2)$. Le cas $t_1 \rhd_\beta t'_1$ est immédiat. Si $t_2 \rhd_\beta t'_2$, c'est facile aussi : l'hypothèse de récurrence nous permet d'obtenir $\Gamma \vdash (t_1\ t'_2) : T_1[x \setminus t'_2]$. Par ailleurs $\Gamma \vdash (t_1\ t_2) : T_1[x \setminus t_2]$ implique $\Gamma \vdash T_1[x \setminus t_2] : s$. Par ailleurs $\Gamma \vdash T_1[x \setminus t'_2] : s'$, et comme $\mathsf{Cl}(t_2) = \mathsf{Cl}(x) = \mathsf{Cl}(t'_2)$ on a bien $s = s'$. On en déduit $\Gamma \vdash (t_1\ t'_2) : T_1[x \setminus t_2]$ par CONV.
  Il reste à traiter le cas $t_1 = [x : T_3]t_3 = t'$ et $t \rhd_\beta t_3[x \setminus t_2]$. Le lemme d'inversion nous assure que $\Gamma :: (x, T_3) \vdash t_3 : T_4$ avec $(x : T_3)T_4 =_{\beta\eta\iota} (x : T_2)T_1$. D'après le corollaire 2.1, ceci implique $T_3 =_{\beta\eta\iota} T_2$ et $T_4 =_{\beta\eta\iota} T_1$, et aussi $\mathsf{Cl}(T_3) = \mathsf{Cl}(T_2)$ et $\mathsf{Cl}(T_4) = \mathsf{Cl}(T_1)$. On en déduit $\Gamma \vdash t_2 : T_3$ et par substitution $\Gamma \vdash t_3[x \setminus t_2] : T_4[x \setminus t_2]$ et donc $\Gamma \vdash t_3[x \setminus t_2] : T_1[x \setminus t_2]$.

– La règle Elim : À l'inverse du cas précédent, c'est bien sûr le cas de la ι-réduction qu'il nous faut traiter en détail ici. On prend $t = \mathsf{Elim}(I, Q, \vec{u}, v)\{\overrightarrow{f_i}\}$ avec $I = \mathsf{Ind}(X : (\vec{x} : \vec{A})\mathrm{Set})\{\overrightarrow{C_i(X)}\}$.
Les cas correspondants à des réductions dans des sous-termes de t se traitent facilement. Nous nous intéresserons donc au cas $v = (\mathsf{Constr}(j, I)\ \vec{v})$, avec

$$t' = (\Delta[C_i(I), f_j, \mathsf{Fun\_Elim}(I, Q, \vec{f_i})]\ \vec{v}).$$

Une simple utilisation du lemme d'inversion permet de vérifier les hypothèses nécessaires à la dérivation de :

$$\Gamma \vdash \mathsf{Fun\_Elim}(I, Q, \vec{f_i}) : (\vec{x} : \vec{A})(c : (I\ \vec{x}))(Q\ \vec{x}\ c).$$

Une simple récurrence sur la structure de $C_j(X)$ permet à partir de là de vérifier que si

$$C_j(X) \equiv (\vec{y_k} : \vec{U_k})(X\ \vec{w})$$

alors

$$\Gamma \vdash \Delta[C_i(I), f_j, \mathsf{Fun\_Elim}(I, Q, \vec{f_i})] : (\vec{y_k} : \vec{U_k})(Q\ \vec{w}\ (\mathsf{Constr}(j, I)\ \vec{y_k})).$$

Comme par ailleurs le lemme d'inversion assure que $\Gamma \vdash \vec{v} : (\vec{y_k} : \vec{U_k})$, on a bien :

$$\Gamma \vdash (\Delta[C_i(I), f_j, \mathsf{Fun\_Elim}(I, Q, \vec{f_i})]\ \vec{v}) : T.$$

■

A partir du lemme précédent et du corollaire 2.4, on déduit aisément :

Corollaire 2.5 *Soit t un terme bien formé. Si $t \rhd^*_{\beta\iota'} t'$, alors $t \rhd^*_{\beta\iota} t'$, et donc t' est bien formé.*

On en déduit en particulier la version définitive du lemme de retard de la η-réduction :

Corollaire 2.6 (Retard de la η-réduction) *Soit t un terme bien formé. Si $t \rhd^* t'$, alors il existe un terme (bien-formé) t'' tel que $t \rhd^*_{\beta\iota} t'' \rhd^*_{\eta} t'$.*

Les deux derniers résultats nous permettent aussi de vérifier que la classe d'un terme est préservée par conversion.

Lemme 2.31 *Soient deux jugements dérivables $\Gamma \vdash A : B$ et $\Gamma \vdash A' : B'$. Si $A =_{\beta\eta\iota} A'$, alors $\mathsf{Cl}(A) = \mathsf{Cl}(A')$.*

Preuve On sait que $\|A\|$ et $\|A'\|$ admettent un réduit commun A_0. On a alors

$$\|A\| \rhd^*_{\beta\iota'_0} A_1 \rhd^*_{\eta} A_0 \text{ et } \|A'\| \rhd^*_{\beta\iota'_0} A'_1 \rhd^*_{\eta} A_0$$

et donc aussi :

$$A \rhd^*_{\beta\iota} A_2 \text{ et } A' \rhd^*_{\beta\iota} A'_2$$

avec $\|A_2\| = A_1$ et $\|A'_2\| = A'_1$. Comme le type est préservé par la $\beta\iota$-réduction, on a $\mathsf{Cl}(A) = \mathsf{Cl}(A_2)$ et $\mathsf{Cl}(A') = \mathsf{Cl}(A'_2)$. Il est très facile de voir que la classe d'un terme est préservée par la η-réduction, et que pour tout terme u, $\mathsf{Cl}(u) = \mathsf{Cl}(\|u\|)$. Aussi :

$$\mathsf{Cl}(A) = \mathsf{Cl}(A_2) = \mathsf{Cl}(A_1) = \mathsf{Cl}(A_0) = \mathsf{Cl}(A'_1) = \mathsf{Cl}(A'_2) = \mathsf{Cl}(A').$$

■

En particulier, et contrairement à Geuvers, nous obtenons le résultat suivant sans utiliser la normalisation :

Corollaire 2.7 *Si les jugements* $\Gamma \vdash A : s$ *et* $\Gamma \vdash A' : s'$ *sont dérivables, et si* $A =_{\beta\eta\iota} A'$, *alors* $s = s'$.

Une autre conséquence est que nous pouvons affaiblir les prémisses de la règle de conversion en préservant la relation de typage.

Corollaire 2.8 *La règle suivante est admissible* [3] :

$$(\textsc{Conv'})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_2 : s \qquad T_1 =_{\beta\eta\iota} T_2}{\Gamma \vdash t : T_2}$$

PREUVE Grace aux résultats précédents, la récurrence sur la structure de la dérivation est à peu près évidente. ■

Pour prouver la correction de la η-réduction, nous avons besoin du lemme suivant :

Lemme 2.32 *Si* $\Gamma_1 :: (y, T_0)\Gamma_2 \vdash t : T$, *avec* $y \notin FV(\Gamma_2) \cup FV(t)$, *alors il existe* T' *tel que* $\Gamma_1\Gamma_2 \vdash t : T'$ *(ce qui implique par ailleurs* $T =_{\beta\eta\iota} T'$*).*

PREUVE Par récurrence sur la structure de la dérivation ; on ne traite que les cas non triviaux.

- La règle LAM, avec $t = [x : T_2]t_1$ et $T = (x : T_2)T_1$. L'hypothèse de récurrence nous assure qu'il existe $T_3 =_{\beta\eta\iota} T_1$ tel que $\Gamma_1\Gamma_2 :: (x, T_2) \vdash t_1 : T_3$. Alors $\Gamma_1\Gamma_2 :: (x, T_2) \vdash T_3 : S$ et $\Gamma_1\Gamma_2 \vdash (x : T_2)T_3 : S$. On en déduit $\Gamma_1\Gamma_2 \vdash t : (x : T_2)T_3$.
- La règle APP, avec $t = (t_1\ t_2)$ et $T = T_1[x \setminus t_2]$. On sait qu'il existe $T_3 =_{\beta\eta\iota} (x : T_2)T_1$ et $T_4 =_{\beta\eta\iota} T_2$ tels que $\Gamma_1\Gamma_2 \vdash t_2 : T_4$ et $\Gamma_1\Gamma_2 \vdash t_1 : T_3$. Alors $T_3 \rhd^*_{\beta\iota} (x : T_2')T_1'$ d'après le lemme de Geuvers et la correction de la $\beta\iota$-réduction [4], en remarquant qu'un terme $((x : U)V\ W)$ n'est jamais bien typé. Aussi, $\Gamma_1\Gamma_2 \vdash t_1 : (x : T_2')T_1'$ avec $T_1' =_{\beta\eta\iota} T_1$ et $T_2' =_{\beta\eta\iota} T_2$. On en déduit $\Gamma_1\Gamma_2 \vdash t_2 : T_4$ et donc $\Gamma_1\Gamma_2 \vdash t : T_3[x \setminus T_4]$.

■

Ce lemme nous permettra de traiter le cas clé dans la preuve de correction de la η-réduction, puisqu'il nous permet d'escamoter une variable qui, après son introduction, n'est plus utilisée dans la suite de la dérivation. Cette propriété est souvent donnée dans sa formulation ci-dessous [54], qui nous semble pourtant un peu moins pratique.

Remarque (Renforcement) *Si* $\Gamma_1 :: (y, T_0)\Gamma_2 \vdash t : T$, *avec* $y \notin FV(\Gamma_2) \cup FV(t) \cup FV(T)$, *alors* $\Gamma_1\Gamma_2 \vdash t : T$.

PREUVE D'après le lemme précédent, il existe $T' =_{\beta\eta\iota} T$ avec $\Gamma_1\Gamma_2 \vdash t : T'$. De plus $\Gamma_1 :: (y, T_0)\Gamma_2 \vdash T : S$, et la encore le lemme précédent nous permet de conclure $\Gamma_1\Gamma_2 \vdash T : S$. Alors une simple application de la règle CONV donne bien $\Gamma_1\Gamma_2 \vdash t : T$. ■

Lemme 2.33 (Correction de la η-réduction) *Si* $\Gamma \vdash t : T$, $t \rhd_\eta t'$ *et* $\Gamma \rhd_\eta \Gamma'$, *alors on a* $\Gamma \vdash t' : T$ *et* $\Gamma' \vdash t : T$.

[3] Cette formulation est celle généralement utilisée dans les présentations de systèmes de types. Le fait d'avoir choisi une règle CONV plus restrictive nous sera utile pour la preuve de normalisation. En revanche ce corollaire montre que les deux systèmes sont équivalents, et qu'il est donc possible d'utiliser la formulation ci-dessus dans une implémentation.

[4] Remarquons que c'est la première fois que nous utilisons le lemme de Geuvers.

PREUVE Par récurrence structurelle sur t. Le cas délicat est bien sûr $t = [x : T_1]t_2$: On sait alors que $\Gamma \vdash T_1 : S_1$, $\Gamma :: (x, T_1) \vdash T_2 : S_2$, $\Gamma :: (x, T_1) \vdash t_2 : T_2$ et $T =_{\beta\eta\iota} (x : T_1)T_2$. Les cas $T_1 \rhd_\eta T_1'$ et $t_2 \rhd_\eta t_2'$ ne posent pas de problèmes ; le seul cas non-trivial est $t_2 = (t_3\ x)$ avec $t' = t_3$. Mais on sait alors que $\Gamma :: (x, T_1) \vdash t_3 : (y : T_1')T_2'$ avec $T_1 =_{\beta\eta\iota} T_1$ et $T_2 =_{\beta\eta\iota} T_2'[y \setminus x]$. On en déduit d'après le lemme précédent que $\Gamma \vdash t_3 : C$ avec $C =_{\beta\eta\iota} (x : T_1)T_2$. Aussi $C \rhd^*_{\beta\iota} (y : T_1'')T_2'' =_{\beta\eta\iota} T$ d'après le lemme de Geuvers, et comme $\Gamma \vdash T : S$ on a bien $\Gamma \vdash t_3 : T$ par application de la règle conv. ■

On résume les résultats précédents par :

Théorème 2 *Soit un terme t bien formé. Si $t \rhd^* t'$, alors t' est bien formé.*

2.9.4 Algèbres des classes de termes

Nous allons utiliser les résultats de ce chapitre pour décrire plus précisément les classes de termes (preuves, prédicats, etc). Cette description nous sera particulièrement utile lors de la preuve de normalisation.

Notations

Dans toute la suite de cette thèse, et sauf indication contraire, nous utiliserons des symboles dédiés pour chaque classe de termes :

t, t_1, t_2, t', f, f', etc pour les termes preuves

$T, T_1, T', F, F', F_1, I, A$, etc pour les prédicats

K, K_1, K', etc pour les ordres.

Par ailleurs les symboles suivants seront utilisés pour désigner indifféremment des termes de toutes les classes : A, B, C, D, E, u, v, w,etc.

Nous pouvons maintenant raffiner la classification des termes bien-formés par une description de ces dernirs en termes de grammaires définies par récurrence croisée :

Les termes bien typés de chaque classe de termes appartiennent respectivement aux algèbres suivantes :

Termes preuves
$t ::= \quad a \mid (t\ t) \mid (t\ T) \mid [a : T]t \mid [\alpha : K]t \mid \mathsf{Constr}(i, T) \mid \mathsf{Elim}(T, T, \vec{u}, t)\{\overrightarrow{t}\}$

Prédicats
$T ::= \quad \alpha \mid (a : T)T \mid (\alpha : K)T \mid (T\ t) \mid (T\ T)$
$\mid [a : T]T \mid [\alpha : K]T \mid \mathsf{Ind}(\alpha : K)\{\overrightarrow{T}\} \mid \mathsf{Elim}(T, K, \vec{u}, t)\{\overrightarrow{T}\}$

Ordres
$K ::= \quad \mathrm{Set} \mid (x : T)K \mid (\alpha : K)K \mid (K\ t) \mid (K\ T) \mid [a : T]K \mid [\alpha : K]K$

Schémas d'Ordres
$E ::= \quad \mathrm{Type} \mid (a : T)E \mid (\alpha : K)E$

Lemme 2.34 *Tout terme bien formé voit sa classe inchangée par réduction.*

PREUVE C'est une conséquence immédiate du théorème ci-dessus et des lemmes d'inversion, d'unicité du typage, et de la correction de la réduction. ■

2.9.5 Forme ordre-normales des ordres

Définition 2.23 (Forme ordre-normale) *Soit K un ordre bien-formé. On dit que K est en forme ordre-normale si l'une des conditions suivantes est vérifiée :*

- $K = \text{Set}$
- $K = (\alpha : K_1)K_2$ *où* K_1 *et* K_2 *sont des ordres en forme ordre-normale*
- $K = (x : T)K$ *où* T *est un prédicat et* K_1 *un ordre en forme ordre-normale*
- $K = [x : t]K_1$ *où* K_1 *est un ordre en forme ordre-normale.*
- $K = [\alpha : K_1]K_2$ *où* K_1 *et* K_2 *sont deux ordres en forme ordre-normale.*

Lemme 2.35 *Pour tout ordre bien formé K, il existe un ordre K' en forme ordre-normale tel que $K \rhd_\beta^* K'$. Par ailleurs le lemme de correction de la β-réduction assure que K' est bien formé de même type que K.*

PREUVE Il suffit de remarquer que la réduction d'un radical $([x : U]K\ u)$ ne crée pas de radical de la même forme. Il est donc possible de les réduire tous "en parallèle" et obtenir ainsi un terme en forme ordre-normale. ■

Corollaire 2.9 *Si* $\Gamma \vdash K : \text{Type}$*, alors sa forme ordre-normale K' est une arité : $Ar(K', \text{Set})$.*

PREUVE C'est une conséquence immédiate du lemme précédent, de la correction du typage, et du lemme d'inversion. ■

Lemme 2.36 *Soit une dérivation* (1) *d'un jugement* $\Gamma \vdash K : \text{Type}$ *et* $A \equiv (\vec{x_i} : \vec{U_i})\text{Set}$ *sa forme ordre normale. Il existe alors une séquence de termes $\vec{V_i}$, avec pour tout i, $V_i \rhd_\beta^* U_i$, et une dérivation strictement plus petite que* (1) *de* $\Gamma \vdash (\vec{x_i} : \vec{V_i}) : \vec{s_i}$*, où pour tout i, $s_i = \text{Set}$ si x_i est une variable de preuve, et $s_i = \text{Type}$ si x_i est une variable de prédicat.*

2.9.6 A propos des formes normales

Il est bien connu, et facile à montrer, que les termes β-normaux du λ-calcul sont tous de la forme $\lambda\vec{x}.(z\ \vec{N_i})$ où les N_i sont tous eux-mêmes des termes normaux. Il s'agit de la forme "arbre de Böhm" des termes normaux. Le lemme suivant et la simple adaptation de cette remarque au Calcul des Constructions Inductives. Nous n'utiliserons ce lemme qu'à la fin du chapitre 4, mais nous préférons l'énoncer maintenant, puisqu'il est conséquence de ce qui précède.

Lemme 2.37 *Tout terme $\beta\iota$-normal bien typé est de l'une des formes suivantes :*

1. $\vec{x} : \vec{N}]N$
2. $(x : N)N'$
3. $s \in \{\text{Set}, \text{Type}, \text{Extern}\}$
4. $(x\ \vec{N})$
5. $(\mathsf{Ind}(X : N)\{\overrightarrow{N}\}\ \vec{N'})$
6. $(\mathsf{Elim}(N, N', \vec{N}, M)\vec{N'})$ *avec M et N deux termes normaux, respectivement des formes 4 et 5.*
7. $(\mathsf{Constr}(i, N)\ \vec{N})$ *avec N terme normal de la forme 5.*

Bien sûr, dans ce qui précède, N, N', M et M' dénotent des termes bien typés eux-mêmes en forme normale, et $\vec{N}$, $\vec{N'}$ des séquences de tels termes.

Chapitre 3

Introduction à la Réductibilité

Ce chapitre est une introduction aux preuves de normalisation forte de λ-calcul typés. Le but principal est d'expliquer la notion de *réductibilité* qui est au centre de ces preuves. On essaye ensuite d'indiquer les idées essentielles et les points clés de la preuve de normalisation forte de CCI. À l'exception de sa partie 3.7, ce chapitre est sensé pouvoir être lu indépendamment du reste de la thèse. Le but de ce chapitre étant d'abord pédagogique, les parties techniques de certaines démonstrations sont données en petits caractères pour indiquer que leur lecture n'est pas indispensable pour saisir l'idée générale.

3.1 Rappels

Ces quelques lignes sont inutiles pour qui aura lu les chapitres précédents.

On considère un ensemble dénombrable de *variables*, que l'on désignera généralement par les lettres x, y, etc. Les λ-termes sont décrits par la grammaire habituelle :

$$\boxed{t \ := \ x \mid (t\ t) \mid \lambda x.t}$$

On définit l'opération de substitution, notée $t[x \setminus t']$ de la manière usuelle. Un *radical* est un terme de la forme $(\lambda x.t\ t')$. On définit la relation binaire $\rightarrow_\beta$ par :

$$(\lambda x.t\ t') \rightarrow_\beta t[x \setminus t']$$

et comme dans le chapitre précédent, on écrit $t \rhd_\beta t'$ pour dire que l'on obtient t' par reécriture d'un sous-terme de t par $\rightarrow_\beta$. Un terme t est *normal* si aucun de ses sous-termes n'est un radical, ou de manière équivalente, s'il n'existe pas de terme t' tel que $t \rhd_\beta t'$. Enfin un terme est dit *fortement normalisable*, s'il n'existe pas de chaîne infinie de réductions partant de t. On appelle $\mathcal{SN}$ l'ensemble des termes fortement normalisables.

> *Dans ce chapitre, on ne s'intéressera pas à la η-réduction. Aussi tous les systèmes considérés dans ce chapitre sont confluents et admettent la propriété de Church-Rosser indépendamment de la propriété de normalisation. On omettra systématiquement les preuves correspondantes.*

3.2 Normalisation et λ-calcul pur

Tout étudiant à qui l'on présente le λ-calcul apprend lors du premier cours, que certains termes n'admettent pas de formes normales. On donne en général l'exemple fameux :

$$(\lambda x.(x\ x)\ \lambda x.(x\ x)) \rhd_\beta (\lambda x.(x\ x)\ \lambda x.(x\ x)).$$

En simplifiant à peine, on peut dire que le but d'un système de type est d'isoler une classe aussi intéressante que possible de λ-termes qui vérifieront certaines propriétés, et en particulier celle de normalisation. La question posée ici est de prouver que tout terme bien typé est effectivement normalisable.

Avant de se lancer dans le typage, un bon exercice mental est d'oublier pour un temps le contre-exemple ci-dessus, et d'essayer de prouver naïvement la normalisation du λ-calcul pur. L'outil essentiel dont nous disposons pour prouver des propriétés sur les λ-termes est la récurrence structurelle. Essayons :

- Le terme x est fortement normalisable.
- Si le terme t est fortement normalisable, alors le terme $\lambda x.t$ l'est évidemment aussi.
- C'est le cas de l'application qui pose problème : même si les termes t_1 et t_2 sont fortement normalisables, le terme $(t_1\ t_2)$ ne l'est pas forcément. En effet, si $t_1 = \lambda x.t_3$, alors $(t_1\ t_2) \rhd_\beta t_3[x \setminus t_2]$ et nous ne pouvons rien dire sur ce dernier terme. Si l'on reprend l'exemple ci-dessus de terme non-normalisable, on voit bien que $\lambda x.(x\ x) \in \boldsymbol{\mathcal{SN}}$.

On peut essayer de renforcer l'hypothèse de récurrence, en quantifiant par exemple sur toutes les substitutions possibles sur t en prouvant par récurrence sur t :

si l'on substitue des termes fortement normalisables à toutes les variables libres de t, on obtient un terme fortement normalisable.

Le lecteur curieux qui s'y essayera verra que cette fois encore, on ne peut terminer la preuve par récurrence à cause du cas de l'application. Dans un premier temps, c'est donc d'abord elle qui retiendra notre attention.

3.3 Types simples

On redonne rapidement la définition du λ-calcul simplement typé (voir aussi [83] par exemple). On se donne un ensemble de *types atomiques* que l'on n'a pas besoin de préciser. L'ensemble des types est alors défini inductivement :

- tout type atomique est un type
- si A et B sont des types, alors $A \to B$ est un type.

La présentation du λ-calcul simplement typé est plus concise si on utilise des variables explicitement typées : on considère simplement qu'à toute variable est associée un type, et pour tout type il existe une infinité de variables qui lui sont associées. On notera x^A, x^B, y^A, etc. À partir de là on définit la relation de typage par :

$$x^A : A \qquad \frac{t : A}{\lambda x^B.t : A \to B} \qquad \frac{t_1 : A \to B \qquad t_2 : A}{(t_1\ t_2) : B}$$

Considérons maintenant le problème de la normalisation forte des λ-termes simplement typés. Pour chaque terme formé par application, nous disposons d'une information essentielle : si $(t_1\ t_2)$ est bien formé, disons de type A, alors t_1 est bien formé d'un type $B \to A$ et t_2 est bien formé de type

B. L'idée est de renforcer à nouveau l'hypothèse de récurrence, mais en se servant de l'information de type : pour chaque type U, on définit un prédicat $|U|$ sur les termes, de manière à ce que si t_1 vérifie $|B \to A|$ et t_2 vérifie $|B|$, alors $(t_1\ t_2)$ vérifie bien $|A|$. On se donne donc :

- $|A|(t) \Leftrightarrow t \in \mathcal{SN}$, si A est atomique.
- $|A \to B|(t) \Leftrightarrow \forall u.|A|(u) \Rightarrow |B|(t\ u)$.

C'est cette nouvelle condition de récurrence que l'on appelle *réductibilité*. On montre alors facilement la condition essentielle que pour tout type la notion de réductibilité est plus forte que la normalisation forte :

Lemme 3.1 *Pour tout type U et tout terme t, si $|U|(t)$, alors t est fortement normalisable.*

PREUVE Par récurrence structurelle sur U. On montre simultanément que $|U| \subset \mathcal{SN}$, et que pour toute variable x et toute séquence $\vec{t}$ de termes fortement normalisables on a $(x\ \vec{t}) \in |U|$. Le cas U atomique est immédiat. Si $U = A \to B$ et $t \in |A \to B|$, on a $x \in |A|$, et donc $(t\ x) \in |B|$. Comme $|B| \subset \mathcal{SN}$, on voit que t est fortement normalisable. Par ailleurs soit $(x\ \vec{t})$ et $u \in |A|$; alors $u \in \mathcal{SN}$ et aussi $(x\ \vec{t}\ u) \in |B|$; on a donc $(x\ \vec{t}) \in |A \to B|$. ■

Dans la suite on identifiera le prédicat $|U|$ et l'ensemble des termes qui le vérifient. On écrira donc indifféremment $|U|(t)$ et $t \in |U|$.

On va donc chercher à montrer que tout terme t de type A vérifie effectivement $|A|(t)$. Cette proposition n'est pas encore tout à fait assez forte, car il nous faut tenir compte des substitutions possibles sur les variables libres de t. Mais comme chacune de ces variables libres possède un type associé, on peut utiliser la notion de réductibilité pour *restreindre le domaine de quantification des substitutions* :

Définition 3.1 (Interprétations adaptées) *On appellera interprétation adaptée, toute substitution $\mathcal{I}$ qui à toute variable libre x^A de type A associera un terme reductible de type A :*

$$\mathcal{I}(x^A) \in |A|.$$

À partir de là, nous sommes capables de quantifier sur toutes les instanciations légales des variables libres d'un terme. Aussi sommes nous maintenant en état de prouver :

Théorème 3 *Si le jugement $t : U$ est dérivable, alors pour toute interprétation adaptée $\mathcal{I}$, on a $t[\mathcal{I}] \in |U|$.*

La preuve se fait par récurrence sur la structure de t, c'est-à-dire, à isomorphisme près, par récurrence sur la structure de la dérivation du jugement. Deux cas sont maintenant immédiats :

• Si $t = x^A$, on sait que $U = A$, et comme $\mathcal{I}$ est une interprétation adaptée, on a $t[\mathcal{I}] = \mathcal{I}(x^A) \in |A| = |U|$.

• Si $t = (t_1\ t_2)$, nous profitons de la définition du prédicat de réductibilité : on sait que $t_1 : A \to B$ et $t_2 : A$ avec $U = B$. Par hypothèse de récurrence on a $t_1[\mathcal{I}] \in |A \to B|$ et $t_2[\mathcal{I}] \in |A|$, alors la définition de $|A \to B|$ nous assure que

$$t[\mathcal{I}] = (t_1[\mathcal{I}]\ t_2[\mathcal{I}]) \in |B| = |U|.$$

Nous avons donc contourné la difficulté liée à l'application. En revanche le problème est repoussé au cas de l'abstraction. En effet l'introduction d'un λ augmente la fonctionnalité du type, et donc l'hypothèse à prouver est d'autant plus forte. C'est là qu'intervient le "miracle" de la réductibilité. On peut en effet généraliser la propriété $(x\ \vec{t}) \in |U|$ pour remarquer :

Lemme 3.2 *Si tous les réduits d'un terme $(t_1\ t_2)$ sont réductibles d'un type U, alors $(t_1\ t_2)$ l'est aussi.*

Ainsi que cet autre lemme qui nous sera également utile :

Lemme 3.3 *Soit un type U. L'ensemble $|U|$ est clos par β-réduction :*

$$t \in |U| \wedge t \rhd t' \Rightarrow t' \in |U|.$$

PREUVE On prouve les deux lemmes conjointement par récurrence sur la structure de U. Le cas U atomique est évident. Si $U = A \to B$, alors on commence par remarquer que $(t_1\ t_2)$ est fortement normalisable puisque tous ses réduits le sont. Ensuite pour tout terme $u \in |A|$, on peut montrer par récurrence sur le nombre maximal de réductions dans u que $(t_1\ t_2\ u) \in |B|$. En effet, on peut appliquer l'hypothèse de récurrence et se contenter de prouver que tous les réduits de $(t_1\ t_2\ u)$ vérifient $|B|$. Or $(t_1\ t_2\ u)$ peut se réduire en :

- $(t'\ u)$ avec $(t_1\ t_2) \rhd t'$. Mais alors $t' \in |A \to B|$ et donc $(t'\ u) \in |B|$.
- $(t_1\ t_2\ u')$ avec $u \rhd u'$. Mais alors on a toujours $u' \in |A|$ (hypothèse de récurrence sur A) et le nombre de réductions possibles dans u a décru.

De même prenons $t \in |A \to B|$ avec $t \rhd t'$. Pour vérifier que $t' \in |A \to B|$, prenons $u \in |A|$. On a $(t\ u) \in |B|$, et donc par hypothèse de récurrence, $(t'\ u) \in |B|$, et donc $t' \in |A \to B|$. ■

À partir de là nous pouvons revenir à la preuve du théorème et traiter maintenant le cas de l'abstraction :

• Si $t = \lambda x^A.t_1$, on sait que $U = A \to B$, avec $t_1 : B$ et pour toute interprétation adaptée $\mathcal{I}$, $t_1[\mathcal{I}] \in |B|$. Montrer que $t[\mathcal{I}] \in |A \to B|$ c'est prouver que pour tout $u \in |A|$, $(t[\mathcal{I}]\ u) \in |B|$. Pour cela, d'après le lemme ci-dessus, il suffit de vérifier que tout réduit de $(t[\mathcal{I}]\ u)$ est bien élément de $|B|$. On raisonne par récurrence sur le nombre maximal de réductions faisables dans $t_1[\mathcal{I}]$ et u. Les réduits de $(t[\mathcal{I}]\ u)$ sont de l'une des formes suivantes :

$$t_1[\mathcal{I}; x \leftarrow u]$$
$$(t'_1\ u) \text{ avec } t_1[\mathcal{I}] \rhd t'_1$$
$$(t_1[\mathcal{I}]\ u') \text{ avec } u \rhd u'$$

Dans le premier cas, comme $\mathcal{I}; x \leftarrow u$ est bien une interprétation adaptée, $t_1[\mathcal{I}; x \leftarrow u]$ appartient bien à $|B|$. Dans les deux autres cas, il suffit de remarquer que le nombre maximal de réductions possibles dans les deux termes a décru et appliquer l'hypothèse de récurrence, en utilisant, dans le troisième cas, le lemme 3.3 pour assurer que $u' \in |A|$.

Ceci clôt la preuve du théorème.

On s'assure ensuite que le théorème suffit bien à prouver la normalisation forte en vérifiant que pour tout que type U, $x^U \in |U|$. Il s'ensuit que l'identité est une interprétation adaptée, et donc que si $t : A$, alors $t \in |A|$ et donc $t \in \mathcal{SN}$.

Cette preuve est due à Tait [140]. La formulation ci-dessus est très proche de [83]. Ce qui est essentiel à retenir, c'est le rôle des différentes étapes de la preuve, que nous retrouverons dans toutes les preuves de normalisation de tous les systèmes de types. À savoir, en simplifiant un peu :

- Chaque type U est interprété par un ensemble de termes, noté $|U|$.
- Preuve que $|U| \subset \mathcal{SN}$.
- Preuve d'un certain nombre de propriétés de clôture pour chaque ensemble $|U|$.
- Preuve que si $t : U$, alors $t[\mathcal{I}] \in |U|$, et donc en particulier $t \in |U|$ et $t \in \mathcal{SN}$.

On peut voir ce travail comme la construction d'un modèle syntaxique pour le système, chaque type étant interprété par un ensemble de termes. On peut également le voir comme les trois étapes d'un raisonnement par récurrence : définition de l'hypothèse de récurrence (construction de $|U|$), vérification que cette hypothèse est bien plus forte que le résultat final ($|U| \subset \boldsymbol{\mathcal{SN}}$), et enfin la preuve par récurrence proprement dite, que si $t : U$ est dérivable, l'hypothèse de récurrence $t[\mathcal{I}] \in |U|$ est bien vérifiée.

Lors de l'étude de systèmes de types plus complexes, le problème essentiel sera la définition de l'interprétation des nouveaux types (inductifs, polymorphes, définis par récurrence, etc). Il faut toutefois remarquer que la définition de $|A \to B|$, qui est le cœur de la preuve, reste inchangée pour tous les systèmes.

3.4 Le système T et son extension par des ordinaux

Le système T a été introduit par Gödel en 1958 [61], pour prouver la cohérence de l'arithmétique fonctionnelle du premier ordre. Il s'agit essentiellement du λ-calcul simplement typé étendu par le type inductif des entiers naturels. Tout comme nous l'avons vu, ceci correspond à l'introduction de deux constructeurs 0 et S, et d'un opérateur de récurrence structurelle, noté R. Les règles de typage sont les mêmes que pour le λ-calcul simplement typé, étendues par les règles suivantes :

$$\frac{\begin{array}{cc} 0 : \mathrm{Nat} & S : \mathrm{Nat} \to \mathrm{Nat} \\ t_0 : T & t_S : \mathrm{Nat} \to T \to T \qquad u : \mathrm{Nat} \end{array}}{(R\ u\ t_0\ t_S) : T}$$

De plus, la β-réduction est étendue par :

$$\begin{array}{rcl} (R\ 0\ t_0\ t_S) & \rhd_\iota & t_0 \\ (R\ (S\ t)\ t_0\ t_S) & \rhd_\iota & (t_S\ t\ (R\ t\ t_0\ t_S)) \end{array}$$

La preuve de normalisation forte est alors une extension très simple de celle, donnée ci-dessus, pour le λ-calcul simplement typé. La définition de la réductibilité reste inchangée. En particulier, on prend :

$$|\mathrm{Nat}| \equiv \boldsymbol{\mathcal{SN}}.$$

La preuve ne diffère alors pas beaucoup de celle du λ-calcul simplement typé. Le travail essentiel est d'adapter le lemme 3.2 ; pour cela, il est commode d'introduire une notion de *termes neutres*.

Définition 3.2 (termes neutres) *Un terme est dit* neutre *s'il n'est pas de l'une des formes suivantes :*

$$\lambda x.t \qquad 0 \qquad (S\ t).$$

On appelle $\boldsymbol{\mathcal{NT}}$ l'ensemble des termes neutres.

On obtient alors une formulation des propriétés de clôture de $|T|$ plus synthétique que les lemmes 3.2 et 3.3.

Lemme 3.4 *Pour tout type T :*
- $|T| \subset \boldsymbol{\mathcal{SN}}$
- $\forall t \in |T|. t \rhd t' \Rightarrow t' \in |T|$

– $\forall t \in \mathcal{NT}.(\forall t'.t \rhd t' \Rightarrow t' \in |T|) \Rightarrow t \in |T|$

La preuve est à peu près identique à celle des lemmes 3.2 et 3.3.

On peut ensuite conserver la définition d'interprétation adaptée du paragraphe précédent ainsi que la formulation du théorème :

Théorème 4 *Pout tout jugement $t : T$ dérivable dans le système T, pour toute interprétation adaptée $\mathcal{I}$ on a $t[\mathcal{I}] \in |T|$.*

PREUVE Là encore elle est par récurrence sur la dérivation du jugement, et diffère peu de la précédente. Le cas intéressant est celui de l'opérateur d'élimination : On sait que $u[\mathcal{I}] \in \mathcal{SN}$, $t_0[\mathcal{I}] \in |T|$, $t_S[\mathcal{I}] \in \mathcal{SN} \to |T| \to |T|$ et il faut vérifier $(R\ u\ t_0\ t_S)[\mathcal{I}] \in |T|$. La preuve utilise deux récurrences emboîtées : une première sur la *taille de la forme normale de* u, une deuxième sur le nombre maximal de réductions possibles dans $u[\mathcal{I}], t_0[\mathcal{I}]$ et $t_S[\mathcal{I}]$. Comme $(R\ u\ t_0\ t_S)[\mathcal{I}]$ est neutre, il suffit de prouver que tous ses réduits sont dans $|T|$, or les-dits réduits sont de l'une des formes suivantes :

– $(R\ u'\ t_0[\mathcal{I}]\ t_S[\mathcal{I}])$ avec $u[\mathcal{I}] \rhd u'$
– $(R\ u[\mathcal{I}]\ t'_0\ t_S[\mathcal{I}])$ avec $t_0[\mathcal{I}] \rhd t'_0$
– $(R\ u[\mathcal{I}]\ t_0[\mathcal{I}]\ t'_S)$ avec $t_S[\mathcal{I}] \rhd t'_S$
– $t_0[\mathcal{I}]$ si $u[\mathcal{I}] = 0$
– $(t_S[\mathcal{I}]\ v\ (R\ v\ t_0[\mathcal{I}]\ t_S[\mathcal{I}]))$.

Les trois premiers cas sont aisément traités par la seconde hypothèse de récurrence. Le quatrième cas est facile : $t_0[\mathcal{I}] \in |T|$. Le dernier cas est le plus subtil : la forme normale de v est plus petite que celle de u, on a donc $(R\ v\ t_0[\mathcal{I}]\ t_S[\mathcal{I}])) \in |T|$ (première hypothèse de récurrence), comme on a également $v \in |\mathrm{Nat}|$, on peut conclure $(t_S[\mathcal{I}]\ v\ (R\ v\ t_0[\mathcal{I}]\ t_S[\mathcal{I}])) \in |T|$. ■

L'exemple du système T est essentiel, mais ne suffit pas tout à fait pour illustrer le principe des preuves par réductibilité adaptées aux types inductifs. En effet, une récurrence sur un entier correspond exactement à une récurrence structurelle sur la forme normale de sa représentation dans le système T [1]. C'est cette dernière propriété qui permet de choisir l'ensemble $\mathcal{SN}$ des termes fortement normalisables pour l'interprétation $|\mathrm{Nat}|$ du type des entiers naturels. Le cas général est plus complexe. Un bon exemple est à ce titre l'extension du système T par le type des ordinaux. Celui-ci est défini à partir du type des entiers naturels à l'aide de trois constructeurs :

$$\begin{array}{lcl} 0_o & : & \mathrm{Ord} \\ S_o & : & \mathrm{Ord} \to \mathrm{Ord} \\ \mathrm{lim} & : & (\mathrm{Nat} \to \mathrm{Ord}) \to \mathrm{Ord} \end{array}$$

Le schéma d'élimination obéit alors à la règle de typage suivante :

$$\frac{t_0 : T \qquad t_S : \mathrm{Ord} \to T \to T \qquad t_l : (\mathrm{Nat} \to \mathrm{Ord}) \to (\mathrm{Nat} \to t) \to t \qquad u : \mathrm{Ord}}{(R_{\mathrm{Ord}}\ u\ t_0\ t_S\ t_l) : T}$$

avec les nouvelles règles de réduction :

$$\begin{array}{lcl} (R_{\mathrm{Ord}}\ 0_o\ t_0\ t_S\ t_l) & \rhd_\iota & t_0 \\ (R_{\mathrm{Ord}}\ (S_o\ u)\ t_0\ t_S\ t_l) & \rhd_\iota & (t_S\ u\ (R_{\mathrm{Ord}}\ u\ t_0\ t_S\ t_l) \\ (R_{\mathrm{Ord}}\ (\mathrm{lim}\ f)\ t_0\ t_S\ t_l) & \rhd_\iota & (t_l\ f\ \lambda x^{\mathrm{Nat}}.(R_{\mathrm{Ord}}\ (f\ x)\ t_0\ t_S\ t_l) \end{array}$$

[1] Cette propriété est en fait vraie pour tout type de données.

C'est bien sûr la troisième de ces règles qui est nouvelle par rapport au cas des entiers naturels. Le point important est l'appel récursif sur $(f\ x)$, qui n'est pas un sous-terme de $(\lim f)$. Il faudra donc, au moins implicitement, exprimer le fait que, quel que soit l'entier x, $(f\ x)$ est inférieur à $(\lim\ f)$ pour un certain ordre bien-fondé. Ceci n'est pas surprenant, puisque $(\lim\ f)$ est justement censé représenter la borne supérieure de l'ensemble $\{f(n), n \in \mathbb{N}\}$. En termes de réductibilité, on exigera donc que $(\lim\ f)$ soit réductible de type Ord (élément de |Ord|), si et seulement si pour tout terme u réductible de type Nat (élément de |Nat|), le terme $(f\ u)$ est bien réductible de type Ord (c.à.d. est lui-même élément de |Ord|). Plus précisément, on définit l'ensemble |Ord| inductivement comme le plus petit ensemble vérifiant les conditions suivantes :

- Si un terme u est fortement normalisable et n'admet pas de réduit de la forme $(S_o\ v)$ ou $(\lim\ f)$, alors $u \in |\text{Ord}|$.
- Si un terme u est fortement normalisable et se réduit vers $(S_o\ v)$ avec $v \in |\text{Ord}|$, alors $u \in |\text{Ord}|$.
- Si un terme u est fortement normalisable et se réduit vers $(\lim\ f)$, avec pour tout $v \in |\text{Nat}|$, $(f\ v) \in |\text{Ord}|$, alors $u \in |\text{Ord}|$.

La justification précise des définitions inductives en théorie des ensembles fait appel au lemme du point fixe de Tarski. C'est-à-dire que l'ensemble |Ord| est en fait défini comme le plus petit point fixe d'un opérateur croissant des ensembles de termes vers les ensembles de termes. Toutefois cette définition de |Ord| est un petit peu lourde à utiliser ; elle implique :

- de refaire la démonstration des lemmes 3.3, 3.2 et 3.1 car Ord est un type atomique qui n'est pas interprété par $\boldsymbol{\mathcal{SN}}$
- de faire une preuve par récurrence sur *la structure de la preuve de* $t \in \text{Ord}$ pour vérifier que $(R_o\ t_0\ f_S\ f_l)$ est bien réductible.

Pour ces deux raisons, nous préférons introduire dès maintenant la notion de *candidats de réductibilité*, qui nous permettra de factoriser les preuves des lemmes 3.3, 3.2 et 3.1 et aussi d'utiliser une définition plus commode de |Ord|. Par ailleurs, cette notion sera, c'est bien connu, indispensable lorsque nous passerons à des systèmes de type imprédicatifs.

Dans ce qui précède, nous avons utilisé les propriétés suivantes :

- Pour tout type T, $|T| \subset \boldsymbol{\mathcal{SN}}$.
- Pour tout type T, l'ensemble $|T|$ est clos par réduction :

$$\forall t, t'.(t \in |T| \wedge t \rhd t') \Rightarrow t' \in |T|.$$

- Pour tout type T, si t est un terme neutre tel que pour tout terme t', $t \rhd t' \Rightarrow t' \in |T|$, alors $t \in |T|$.

Une remarque importante est que lorsque l'on prouve ces propriétés par récurrence sur la structure de T, on ne se sert *que* des hypothèses de récurrence. A partir de là on va pouvoir factoriser la plus grande partie de cette preuve, en distinguant les ensembles de termes vérifiant ces propriétés :

Définition 3.3 (Candidats de réductibilité) *Un ensemble de termes $\mathcal{C}$ est un candidat de réductibilité s'il vérifie les propriétés suivantes :*

CR 1	$\mathcal{C} \subset \boldsymbol{\mathcal{SN}}$
CR 2	$\forall t \in \mathcal{C}.\forall t'.(t \in \mathcal{C} \wedge t\ \mathbf{B}\ t') \Rightarrow t' \in \mathcal{C}$
CR 3	$\forall t \in \boldsymbol{\mathcal{NT}}.(\forall t'.t\ \mathbf{B}\ t' \Rightarrow t' \in \mathcal{C}) \Rightarrow t \in \mathcal{C}$

Notre remarque précédente est alors illustrée par les trois lemmes :

Lemme 3.5 *L'ensemble $\boldsymbol{\mathcal{SN}}$ est un candidat de réductibilité.*

Lemme 3.6 *Soient $\mathcal{C}$ et $\mathcal{D}$ deux candidats de réductibilité. Alors l'ensemble suivant est un candidat de réductibilité :*

$$\mathcal{C} \to \mathcal{D} \equiv \{t, \forall u \in \mathcal{C}.(t\ u) \in \mathcal{D}\}.$$

PREUVE Soit $t \in \mathcal{C} \to \mathcal{D}$. $\mathcal{C}$ contient tous les termes normaux neutres, en particulier toutes les variables, on a donc $(t\ x) \in \mathcal{D}$ et donc $(t\ x)$ et t sont fortement normalisables. Si $(t\ u) \in \mathcal{D}$ et t **B** t', alors $(t\ u)$ **B** $(t'\ u)$ et donc $(t'u) \in \mathcal{D}$. On en déduit $t' \in \mathcal{C} \to \mathcal{D}$. Enfin supposons maintenant que t soit neutre et que tous ses réduits soient dans $\mathcal{C} \to \mathcal{D}$. Soit alors $u \in \mathcal{C}$, on prouve par récurrence sur le nombre maximal de réductions faisables dans $\mathcal{C}$ que $(t\ u) \in \mathcal{D}$; en effet, $(t\ u)$ est neutre et ses réduits sont de la forme :

- $(t'\ u)$ qui est dans $\mathcal{D}$ puisque $t' \in \mathcal{C} \to \mathcal{D}$
- $(t\ u')$ qui est dans $\mathcal{D}$ par hypothèse de récurrence.

Il n'y a pas d'autre cas, puisque t est neutre. ■

Lemme 3.7 *L'intersection d'un ensemble non-vide de candidats de réductibilité est un candidat de réductibilité.*

La preuve de ce dernier lemme est à peu près immédiate. Ce lemme est important, car il dit que tout ensemble de candidats de réductibilité admet une *borne inférieure* pour l'ordre d'inclusion. En d'autres termes, l'ensemble des candidats de réductibilité muni de l'ordre d'inclusion est un inf-demi-treillis. Une conséquence essentielle est que tout opérateur croissant sur les candidats de réductibilité admet un *plus petit point fixe*. Nous allons tout de suite faire usage de cette possibilité pour construire |Ord|.

Définition 3.4 *On définit* |Ord| *comme le plus petit candidat de réductibilité vérifiant la condition suivante : un terme t appartient à* |Ord| *si et seulement si pour tout candidat de réductibilité $\mathcal{C}$, pour tous termes $t_0 \in \mathcal{C}$, $t_S \in |\mathrm{Ord}| \to \mathcal{C} \to \mathcal{C}$, $t_l \in (|Nat| \to |\mathrm{Ord}|) \to (|Nat| \to \mathcal{C}) \to \mathcal{C}$, on a*

$$(R_{\mathrm{Ord}}\ t\ t_0\ t_S\ t_l) \in \mathcal{C}.$$

Il faut bien sûr justifier l'existence de l'ensemble |Ord|. Grâce a la remarque précédente, nous allons pouvoir le définir comme le plus petit point fixe d'un opérateur croissant sur les candidats :

Soit S un ensemble de termes et $\mathcal{C}$ un candidat de réductibilité. On peut définir l'ensemble de termes suivant :

$$G(S,\mathcal{C}) \equiv \{t, \forall (t_0, t_S, t_l) \in \mathcal{C} \times S \to \mathcal{C} \to \mathcal{C} \times (\boldsymbol{\mathcal{SN}} \to S) \to (\boldsymbol{\mathcal{SN}} \to \mathcal{C}) \to \mathcal{C}.(R_o\ t\ t_0\ t_S\ t_l) \in \mathcal{C}\}$$

et aussi l'application F des ensembles de termes vers les ensembles de termes, définie par :

$$F(S) \equiv \bigcap_{\mathcal{C} \in \mathcal{CR}} G(S,\mathcal{C}).$$

On peut alors définir |Ord| comme le plus petit point fixe de F à condition de vérifier les deux conditions suivantes :

- F est croissante : si $S_1 \subset S_2$ alors $F(S_1) \subset F(S_2)$
- F est effectivement un opérateur sur les candidats : si $S \in \mathcal{CR}$ alors $F(S) \in \mathcal{CR}$.

Le premier point est facile ; il suffit de remarquer $\mathcal{C} \subset \mathcal{C}' \Rightarrow \mathcal{C}' \to \mathcal{D} \subset \mathcal{C} \to \mathcal{D}$ et le reste suit aisément. Pour le deuxième point, $\mathcal{CR}$ étant clos pour l'intersection, il suffit de vérifier que $G(S,\mathcal{C}) \in \mathcal{CR}$. Les preuves des conditions **CR 1** et **CR 2** sont immédiates. La preuve est similaire à celle du lemme 3.6, par récurrence sur le nombre maximal de réductions possibles sur les termes t_0, t_S et t_l.

À partir de là, il est immédiat que :

Lemme 3.8 *Soit T un type quelconque du système avec ordinaux. L'ensemble $|T|$ est un candidat de réductibilité.*

On peut expliquer ainsi la définition de $|\mathrm{Ord}|$: comme tous les $|T|$ sont des candidats de réductibilité, *il suffit de quantifier sur tous les candidats de réductibilité pour quantifier en particulier sur toutes les interprétations $|T|$ possibles.* C'est aussi exactement ce qui sera fait dans la section suivante pour interpréter la quantification polymorphe du système F.

Le reste de la preuve est très facile. Comme précédemment on prouve que si $t : T$, alors pour toute interprétation adaptée $\mathcal{I}$ on a $t[\mathcal{I}] \in |T|$. On détaille les cas des nouvelles règles :

- introduction de 0_o : si $\mathcal{C} \in \mathcal{CR}$, $t_0 \in \mathcal{C}$ et t_S et t_l sont fortement normalisables, on vérifie facilement, comme d'habitude maintenant par récurrence sur le nombre maximal de réductions possibles dans t_0, t_S et t_l que $(R_o\ 0_o\ t_0\ t_S\ t_l) \in \mathcal{C}$.
- les cas des règles d'introduction de S_o et lim sont tout à fait similaires [2].
- introduction de R_o : c'est immédiat ; par définition, si $u[\mathcal{I}] \in |\mathrm{Ord}|$, $t_o[\mathcal{I}] \in |T|$, $t_S[\mathcal{I}] \in |\mathrm{Ord} \to T \to T|$ et $t_l \in |(\mathrm{Nat} \to \mathrm{Ord}) \to (\mathrm{Nat} \to T) \to T|$ alors

$$(R_o\ u[\mathcal{I}]\ t_o[\mathcal{I}]\ t_S[\mathcal{I}]\ t_l[\mathcal{I}]) \in |T|.$$

3.5 Le système F

Le système F, également appelé λ-calcul du second ordre, a été introduit par Girard en 1970 pour interpréter les preuves de l'arithmétique fonctionnelle du second ordre et prouver ainsi sa cohérence [58, 59]. En arithmétique fonctionnelle du second ordre, on peut définir une proposition ou un prédicat en quantifiant sur toutes les propositions ou prédicats ; c'est ce que l'on appelle l'imprédicativité. Sa contre-partie en λ-calcul est précisément la présence de types polymorphes. En fait, le système F est le λ-calcul polymorphe le plus simple qui soit. Les types de ce système sont décrits par l'algèbre :

$$T \ := \ \alpha \mid T \to T \mid \forall \alpha.T$$

où α désigne un élément d'un ensemble dénombrable et donné de *variables de types.*

Nous considérerons la version du système où les applications de types ne sont pas données explicitement dans les termes. Du fait de la présence de variables liées dans les types, une présentation utilisant des contextes est plus commode ; ceci rend par ailleurs inutiles les annotations de types sur les variables. Les termes sont décrits par :

$$t \ := \ x \mid (t\ t) \mid \lambda x.t$$

et les règles de typage sont :

$$(\textsc{Var})\ \frac{(x,T) \in \Gamma}{\Gamma \vdash x : T}$$

$$(\textsc{Lam})\ \frac{\Gamma :: (x, T_2) \vdash t : T_1}{\Gamma \vdash \lambda x.t : T_2 \to T_1} \qquad (\textsc{App})\ \frac{\Gamma \vdash t_1 : T_2 \to T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash (t_1\ t_2) : T_1}$$

$$(\forall\text{-I})\ \frac{\Gamma \vdash t : T \qquad \forall (x, T_0) \in \Gamma . \alpha \notin FV(T_0)}{\Gamma \vdash t : \forall \alpha.T} \qquad (\forall\text{-E})\ \frac{\Gamma \vdash t : \forall \alpha.T}{\Gamma \vdash t[\alpha \setminus T_1] : T[\alpha \setminus T_1]}$$

[2] Notons en particulier que l'on n'a pas besoin d'induction transfinie pour vérifier que $\mathrm{lim} \in |(\mathrm{Nat} \to \mathrm{Ord}) \to \mathrm{Ord}|$

La définition des contextes et de FV ainsi que la signification de $(x,T) \in \Gamma$ sont évidentes ; on peut reprendre les définitions formelles du chapitre 2. De même les définitions des termes neutres et fortement normalisables ainsi que des candidats de réductibilité restent quasiment inchangées.

La différence essentielle avec les systèmes précédents est bien sûr la présence de variables de types et de la quantification du second ordre. Le travail consistera essentiellement à réussir à interpréter ces derniers. Il est clair que l'interprétation d'une variable de type dépend de la manière dont elle sera ensuite instanciée. Pour celà, on commence par se donner l'interprétation de chaque variable pour définir les interprétations des types. De cette manière, on va être capable de quantifier sur toutes les instanciations possibles. Comme dans la définition de Ord dans le système précédent, une quantification universelle sur tous les candidats de réductibilité permettera de capturer une quantification universelle sur tous les types.

Définition 3.5 (Interprétation) *On appelle interprétation une application associant un candidat de réductibilité à toute variable de type et un λ-terme à toute variable de terme.*

Une interprétation sera généralement désignée par la lettre $\mathcal{I}$. Cette notion étend la notion de substitution utilisée jusqu'alors dans la dernière partie de la preuve. On pourrait ici distinguer interprétations des variables de type et interprétations des variables de terme, mais dans les systèmes avec types dépendants il nous faudra de toute façons les mélanger.

On peut alors définir, pour tout type T, le candidat de réductibilité $|T|_{\mathcal{I}}$ pour toute interprétation $\mathcal{I}$. On désignera par $\mathcal{I};\alpha \leftarrow \mathcal{C}$ l'interprétation valant $\mathcal{C}$ en α et égale à $\mathcal{I}$ partout ailleurs.

$$\begin{aligned} |\alpha|_{\mathcal{I}} &\equiv \mathcal{I}(\alpha) \\ |A \to B|_{\mathcal{I}} &\equiv |A|_{\mathcal{I}} \to |B|_{\mathcal{I}} \\ |\forall \alpha.A|_{\mathcal{I}} &\equiv \bigcap_{\mathcal{C} \in \mathcal{CR}} |A|_{\mathcal{I};\alpha \leftarrow \mathcal{C}} \end{aligned}$$

D'après les lemmes précédents, il est clair que $|T|_{\mathcal{I}}$ est effectivement un candidat de réductibilité. Par ailleurs une propriété essentielle est que l'interprétation est compatible avec la substitution :

Lemme 3.9 *Pour tous types T et T_1, toute variable de type α et tout interprétation $\mathcal{I}$, on a :*

$$|T[\alpha \setminus T_1]|_{\mathcal{I}} = |T|_{\mathcal{I};\alpha \leftarrow |T_1|_{\mathcal{I}}}.$$

Preuve On raisonne bien sûr par récurrence structurelle sur T ; les trois cas sont quasi-immédiats. ■

Il nous suffit alors d'adapter la définition 3.1 à la notion d'interprétation pour terminer notre preuve :

Définition 3.6 (Interprétations adaptées) *Une interprétation $\mathcal{I}$ est dite adaptée à un contexte Γ , si pour tout couple (x,T) apparaissant dans Γ, on a bien $\mathcal{I}(x) \in |T|_{\mathcal{I}}$.*

Théorème 5 *Si un jugement $\Gamma \vdash t : T$ est dérivable, alors pour toute interprétation $\mathcal{I}$ adaptée à Γ, on a $t[\mathcal{I}] \in |T|_{\mathcal{I}}$.*

La preuve est bien sûr par récurrence sur la structure de la dérivation du jugement de typage ; elle ressemble énormément à la preuve correspondante pour le λ-calcul simplement typé. Les cas supplémentaires ne sont pas difficiles.

– Le cas de la règle Var est immédiat : par définition, $\mathcal{I}(x) \in |T|_{\mathcal{I}}$.

- Le cas de App est identique au cas des types simples : $t_1[\mathcal{I}] \in |T_2 \to T_2|_\mathcal{I}$ et $t_2[\mathcal{I}] \in |T_2|_\mathcal{I}$. Aussi : $(t_1\ t_2)[\mathcal{I}] = (t_1[\mathcal{I}]\ t_2[\mathcal{I}]) \in |t_1|_\mathcal{I}$.
- Le cas de Lam n'est pas vraiment nouveau non plus. On sait que pour toute interprétation $\mathcal{I}$ adaptée à $\Gamma :: (x, T_2)$, $t[\mathcal{I}] \in |T_1|_\mathcal{I}$. Soit maintenant $t_2 \in |T_2|_\mathcal{I}$; il nous faut vérifier

  $$(\lambda x^{T_2}.t[\mathcal{I}; x^{T_2} \leftarrow x^{T_2}]\ t_2) \in |T_1|_\mathcal{I}$$

  pour prouver $(\lambda x^{T_2}.t)[\mathcal{I}] \in |T_1 \to T_2|_\mathcal{I}$. Comme précédemment, on raisonne par récurrence sur le nombre maximal de réductions possibles dans $t[\mathcal{I}]$ et t_2. Le terme $(\lambda x^{T_2}.t[\mathcal{I}]\ t_2)$ peut se réduire en :
  - $(\lambda x^{T_2}.t'\ t'_2)$, cas traité par l'hypothèse de récurrence
  - $t[\mathcal{I}; x^{T_2} \leftarrow t_2]$, mais alors comme $t_2 \in |T_2|_\mathcal{I}$, $\mathcal{I}; x^{T_2} \leftarrow t_2$ est une interprétation adaptée à $\Gamma :: (x, T_2)$, et donc $t[\mathcal{I}; x^{T_2} \leftarrow t_2] \in |T_1|_\mathcal{I}$.
- Soit $\mathcal{I}$ une interprétation adaptée à Γ et $\mathcal{C}$ un candidat de réductibilité quelconque. Il faut vérifier que $\mathcal{I}; \alpha \leftarrow \mathcal{C}$ est bien une interprétation adaptée à Γ. C'est évident, puisque pour tout (x, T_0) apparaissant dans Γ, on a $\alpha \notin FV(T_0)$, et donc $|T_0|_\mathcal{I} = |T_0|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$.
- L'ensemble $|T_1|_\mathcal{I}$ est un candidat de réductibilité ; donc

  $$|T[\alpha \setminus T_1]|_\mathcal{I} = |T|_{\mathcal{I};\alpha\leftarrow|T_1|_\mathcal{I}} \subset |\forall\alpha.T|_\mathcal{I}$$

  ce qui implique $t[\mathcal{I}] \in |T[\alpha \setminus T_1]|_\mathcal{I}$.

Enfin pour vérifier que tous les termes typables sont fortement normalisables, il suffit de remarquer que l'interprétation $\mathcal{I}$ telle que $\mathcal{I}(\alpha) = \boldsymbol{\mathcal{SN}}$ pour tout α et $\mathcal{I}(x) = x$ pour tout x est adaptée à tout contexte.

Il faut remarquer que la concision et la simplicité de présentation du système F se retrouve au niveau de la preuve de normalisation ; cette dernière est plus courte et plus élégante que, par exemple, la preuve pour le système avec ordinaux. Une remarque bien connue est que comme le système F est imprédicatif, sa preuve de normalisation doit l'être aussi. Plus précisément, c'est le fait de quantifier sur toutes les interprétations $\mathcal{I}$ qui est effectivement un raisonnement d'ordre supérieur : une interprétation définit en effet un prédicat sur les λ-termes pour chaque variable de type. On quantifie donc bien sur une classe de prédicats.

Il faut aussi noter que prouver la normalisation du système F étendu par les entiers primitifs du système T, ainsi que des ordinaux comme présenté dans la section précédente ne pose maintenant aucun problème. Il suffit de suivre la preuve pour le système F, en reprenant les définitions de |Nat| et |Ord| données précédemment. Cette remarque est importante puisque nous cherchons ultimement à prouver la normalisation forte pour un système incorporant polymorphisme et types inductifs.

3.6 Les types dépendants

La notion de types dépendants correspond, comme indiqué dans le chapitre 1, à la possibilité de faire dépendre un type d'un terme, ce qui permet de représenter les prédicats de manière interne au calcul. Il connu que, pour les λ-calculs dits du "Cube de Barendregt", on peut assez facilement ramener la normalisation forte du système avec types dépendants à celui du système sans types dépendants correspondant (F_ω pour le Calcul des Constructions, types simples pour LF...). Nous ne ferons toutefois pas usage de cette facilité, car nous n'en aurons plus la possibilité lorsque nous ajouterons l'élimination forte.

Le système que nous considérerons ici est le système F étendu par des types dépendants. Dans la classification du Cube de Barendregt, il est connu sous le nom de λP_2. Il se distingue du Calcul des Constructions uniquement en ce qu'il interdit la formation de fonctions des types vers les types (ou plus généralement des prédicats vers les prédicats) : pour tout type A on peut définir $list_A$, le type des listes d'éléments de type A, mais on ne peut pas construire le type "générique" *list*, qui à A associe $list_A$.

Contrairement à ce que nous avons fait dans le chapitre précédent pour le Calcul des Constructions inductives, nous donnons une présentation stratifiée des preuves (t), des prédicats (T) et des ordres (K) :

$$\begin{array}{lcl} K & ::= & \mathrm{Set} \mid (x:T)K \\ T & ::= & \alpha \mid [x:T]T \mid (T\ t) \mid (\alpha:K)T \mid (x:T_1)T_2 \\ t & ::= & x \mid [x:T]t \mid (t\ t) \mid [\alpha:K]t \mid (t\ T) \end{array}$$

Pour ces trois classes de termes, on peut sans difficultés définir les notions de substitution, β-réduction normalisation et β-équivalence de la manière usuelle. Les contextes sont définis comme suit :

$$\Gamma \ ::= \ [] \mid \Gamma :: (x,T) \mid \Gamma :: (\alpha,K).$$

La complication de la présentation se comprend bien si l'on songe qu'une variable de type α peut effectivement représenter un type (c'est-à-dire $\alpha : \mathrm{Set}$), ou par exemple un prédicat sur un type donné ($\alpha : \mathrm{Nat} \to \mathrm{Set}$). Il est donc nécessaire de faire apparaître les liaisons des variables de type (ou plus exactement des variables de prédicat) dans le contexte. Les ordres servent à déterminer le domaine d'un prédicat : si $T : \mathrm{Set}$, alors T est effectivement un type, si $T : T_0 \to \mathrm{Set}$, T sera un prédicat sur les objet de type T_0, etc. Bien sûr, si un jugement $\Gamma \vdash t : T$ est dérivable, alors T sera un type et on pourra aussi dériver $\Gamma \vdash T : \mathrm{Set}$.

Les jugements peuvent être de trois formes :

- $\Gamma \vdash t : T$ pour dire que le terme t est bien formé et habite le type T.
- $\Gamma \vdash T : K$ pour dire que T est un prédicat bien-formé d'ordre K.
- $\Gamma \vdash K$ pour dire que K est un ordre bien-formé.

On écrit $\Gamma \vdash \mathcal{J}$ pour désigner indifféremment les trois formes de jugement. Les règles de dérivation

sont alors :

$$(\text{Ax})\ [] \vdash \text{Set} \qquad (\text{K-Prod})\ \frac{\Gamma :: (x,T) \vdash K}{\Gamma \vdash (x:T)K}$$

$$(\text{T-Lam})\ \frac{\Gamma :: (x,T_1) \vdash T : K}{\Gamma \vdash [x:T_1]T : (x:T_1)K} \qquad (\text{T-App})\ \frac{\Gamma \vdash T : (x:T_1)K \qquad \Gamma \vdash t : T_1}{\Gamma \vdash (T\ t) : K[x \setminus t]}$$

$$(\text{T-Prod1})\ \frac{\Gamma :: (x,T_1) \vdash T : \text{Set}}{\Gamma \vdash (x:T_1)T : \text{Set}} \qquad (\text{T-Prod2})\ \frac{\Gamma :: (\alpha,K) \vdash T : \text{Set}}{\Gamma \vdash (\alpha:K)T : \text{Set}}$$

$$(\text{T-Lam1})\ \frac{\Gamma :: (x,T_1) \vdash t : T}{\Gamma \vdash [x:T_1]t : (x:T_1)T} \qquad (\text{T-Lam2})\ \frac{\Gamma :: (\alpha,K) \vdash t : T}{\Gamma \vdash [\alpha:K] : (\alpha:K)T}$$

$$(\text{T-App1})\ \frac{\Gamma \vdash t_1 : (x:T_2)T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash (t_1\ t_2) : T_1[x \setminus t_2]} \qquad (\text{T-App2})\ \frac{\Gamma \vdash t : (\alpha:K)T \qquad \Gamma \vdash T_1 : K}{\Gamma \vdash (t\ T_1) : T[\alpha \setminus T_1]}$$

$$(\text{Var1})\ \frac{\Gamma \vdash T : \text{Set} \qquad x \notin \Gamma}{\Gamma :: (x,T) \vdash x : T} \qquad (\text{Var2})\ \frac{\Gamma \vdash K \qquad \alpha \notin \Gamma}{\Gamma :: (\alpha,K) \vdash \alpha : K}$$

$$(\text{Weak1})\ \frac{\Gamma \vdash \mathcal{J} \qquad \Gamma \vdash T : \text{Set}}{\Gamma :: (x,T) \vdash \mathcal{J}} \qquad (\text{Weak2})\ \frac{\Gamma \vdash \mathcal{J} \qquad \Gamma \vdash K}{\Gamma :: (\alpha,K) \vdash \mathcal{J}}$$

$$(\text{Conv})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_1 : \text{Set} \qquad T_1 =_\beta T_2 \qquad \Gamma \vdash T_2 : \text{Set}}{\Gamma \vdash t : T_2}$$

Dans ce chapitre, nous nous intéresserons uniquement à la normalisation forte des λ-termes purs sous-jacents aux preuves. On passe des unes aux autres par l'opération "d'oubli" suivante :

$$\begin{aligned} \|x\| &\equiv x \\ \|[x:T]t\| &\equiv \lambda x.\|t\| \\ \|[\alpha:K]t\| &\equiv \|t\| \\ \|(t_1\ t_2)\| &\equiv (\|t_1\|\ \|t_2\|) \\ \|(t\ T)\| &\equiv \|t\| \end{aligned}$$

On peut alors prouver facilement la propriété essentielle suivante :

Lemme 3.10 *Soit un terme t. Si $t\ \mathbf{B}^*_\beta\ t'$, alors $\|t\|\ \mathbf{B}^*_\beta\ \|t'\|$. Aussi, pour tous termes t et t', on a*

$$t =_\beta t' \Rightarrow \|t\| =_\beta \|t'\|.$$

L'adaptation de la méthode de réductibilité au présent système se comprend intuitivement bien. Comme précédemment, nous allons interpréter tout type (c.à.d. tout prédicat d'ordre Set) par un candidat de réductibilité, c.à.d. un ensemble de λ-termes ; comme les prédicats d'arité 1 (c.à.d. d'un ordre de la forme $T \to \text{Set}$) sont essentiellement des fonctions des preuves vers les types, ils seront

interprétés par des applications associant un candidat de réductibilité à tout terme pur, et ainsi de suite.

Formellement, on commence par définir *l'arité* d'un ordre comme le nombre de produits qui le forment :

$$\mathsf{Ar}(\mathrm{Set}) = 0$$
$$\mathsf{Ar}((x : T)K) = \mathsf{Ar}(K) + 1$$

on peut alors définir pour chaque entier n l'ensemble $\rho(n)$ des *valuations d'arité* n :
- $\rho(0)$ est l'ensemble des parties de Λ
- $\rho(n+1)$ est l'ensemble des applications de Λ vers $\rho(n)$.

Nous pouvons maintenant définir une notion de candidat de réductibilité pour chaque arité :

Définition 3.7 *Pour tout entier naturel n, on définit $\mathcal{CR}(n) \subset \rho(n)$, l'ensemble des* candidats de réductibilité de degré n *:*
- *$\mathcal{CR}(0)$ est l'ensemble des ensembles de termes $\mathcal{C}$ vérifiant les trois conditions* **CR 1**, **CR 2** *et* **CR 3** *déjà énoncées (définition 3.3)*
- *$\mathcal{CR}(n+1)$ est l'ensemble des applications f de Λ vers $\mathcal{CR}(n)$ vérifiant :*

$$\forall t, t'.t =_\beta t' \Rightarrow f(t) = f(t').$$

On a déjà vu que $\boldsymbol{\mathcal{SN}}$ était un candidat de réductibilité d'arité 0. On peut généraliser ceci en définissant pour tout n un élément canonique $Can(n)$ de $\mathcal{CR}(n)$:
- $Can(0) \equiv \boldsymbol{\mathcal{SN}}$
- $Can(n+1)$ est l'application qui à tout λ-terme associe $Can(n)$.

Cette remarque nous permet en particulier de quantifier sur l'ensemble $\mathcal{CR}(n)$, qui est donc non vide.

On veut maintenant définir les interprétations $\mathcal{I}$ et les valuations $|T|_{\mathcal{I}}$. Mais pour cela, il nous faut d'abord nous assurer que $\mathcal{I}$ interprétera chaque variable α par une valuation de la bonne arité. Pour cela, nous utilisons une notion de prédicats *bien-construits*.

Définition 3.8 (Arité-interprétation) *Une arité-interprétation est une application $\mathcal{A}$ qui associe un entier naturel $\mathcal{A}(\alpha)$ à toute variable de prédicat. On désigne par $\mathcal{A}; \alpha \leftarrow n$ l'arité-interprétation valant n en α et égale à $\mathcal{A}$ partout ailleurs.*

Définition 3.9 (Bonne-construction) *Soit T un prédicat et $\mathcal{A}$ une arité-interprétation. On dit que T est bien construit d'arité n sous $\mathcal{A}$ si :*
- *$T = \alpha$ et $\mathcal{A}(\alpha) = n$*
- *$T = (x : T_1)T_2$, $n = 0$ et T_1 et T_2 sont bien construits d'arité 0 sous $\mathcal{A}$.*
- *$T = (\alpha : K)T_0$, $n = 0$, et T_0 est bien construit d'arité 0 sous $\mathcal{A}; \alpha \leftarrow \mathsf{Ar}(K)$.*
- *$T = (T_0\ t)$ et T_0 est bien-construit d'arité $n+1$ sous $\mathcal{A}$.*
- *$T = [x : T_1]T_2$ et T_2 est bien-construit d'arité $n-1$ sous $\mathcal{A}$.*

Si un prédicat est bien-construit d'arité n, on peut alors l'interpréter par un candidat de réductibilité d'arité n, à condition de disposer de valuations de la bonne arité pour toute variable de prédicat.

Définition 3.10 (Interprétation) *Une interprétation $\mathcal{I}$ est un application qui :*

- *associe un* λ-terme pur $\mathcal{I}(x)$ *à toute variable de preuve* x.
- *associe un couple* $\mathcal{I}(\alpha) = (n, \mathcal{C})$ *à toute variable de prédicat* α, *où* n *est un entier naturel et* $\mathcal{C}$ *un candidat de réductibilité d'arité* n.

Toute interprétation définit évidemment une arité-interprétation associée. Étant donné un terme preuve t, *on désigne par* $t[\mathcal{I}]$ *le terme pur* $\|t\|$ *dont toutes les variables libres ont été substitués par leurs valuation* $\mathcal{I}(x)$. *Par exemple :*

$$[\alpha : K][x : T \to T](x\ y)[\mathcal{I}] = \lambda x.(x\ \mathcal{I}(y)).$$

Définition 3.11 *Soit* $\mathcal{I}$ *une interprétation et* T *une prédicat bien-construit d'arité* n *sous l'arité-interprétation associée à* $\mathcal{I}$. *On définit alors* $|T|_{\mathcal{I}}$, *candidat de réductibilité d'arité* n *par récurrence sur la structure de* T :

- $|\alpha|_{\mathcal{I}} \equiv \mathcal{C}$ *avec* $\mathcal{I}(\alpha) = (n, \mathcal{C})$
- $|(T\ t)|_{\mathcal{I}} \equiv |T|_{\mathcal{I}}(t[\mathcal{I}])$
- $|[x : T_1]T_2|_{\mathcal{I}}$ *est l'application qui à tout terme preuve* u *associe* $|T_2|_{\mathcal{I};x\leftarrow u}$
- $|(\alpha : K)T|_{\mathcal{I}} \equiv \bigcap_{\mathcal{C}\in\mathcal{CR}(\mathsf{Ar}(K))} |T|_{\mathcal{I};\alpha\leftarrow(\mathsf{Ar}(K),\mathcal{C})}$
- $|(x : T_1)T_2|_{\mathcal{I}}$ *est l'ensemble des* λ-*termes purs* u, *tel que pour tout* λ-*terme* $v \in |T_1|_{\mathcal{I}}$, *on a* $(u\ v) \in |T_2|_{\mathcal{I};x\leftarrow v}$

Il faut souligner que la définition de $|(\alpha : K)T|_{\mathcal{I}}$ est simplement la généralisation de la définition de $|\forall\alpha.T|_{\mathcal{I}}$ utilisée dans le cadre du système F, alors que la définition de $|(x : T_1)T_2|_{\mathcal{I}}$ correspond à la définition de $|A \to B|_{\mathcal{I}}$ utilisée jusqu'à maintenant, à la nuance près que l'interprétation de T_2 dépend maintenant de la valeur de x.

Pour que la définition ci-dessus soit rigoureuse, il nous faut vérifier que $|T|_{\mathcal{I}}$ est effectivement toujours un candidat de réductibilité de la bonne arité. La preuve n'est pas difficile ; il faut juste renforcer légèrement l'hypothèse de récurrence.

Preuve En même temps que l'on définit $|T|_{\mathcal{I}}$ par récurrence sur la structure de T, on prouve également les deux hypothèses de récurrence :

- Si T est bien construit d'ordre n sous $\mathcal{I}$, alors $|T|_{\mathcal{I}}$ est un candidat de réductibilité d'ordre n
- Soit $\mathcal{I}'$ une interprétation telle que $\mathcal{I}(\alpha) = \mathcal{I}'(\alpha)$ pour tout α et $\mathcal{I}(x) =_\beta \mathcal{I}'(x)$ pour tout x. Alors $|T|_{\mathcal{I}} = |T|_{\mathcal{I}'}$.

Le cas $T = \alpha$ est immédiat et $T = (T'\ t)$ et très facile ainsi que $T = [\alpha : K]T'$. On détaille un peu plus les autres cas :

- $T = [x : T_1]T_2$. Pour tout terme pur u, $|T_2|_{\mathcal{I};x\leftarrow u}$ est un candidat de réductibilité par hypothèse de récurrence. De plus si $u =_\beta u'$, alors $|T_2|_{\mathcal{I};x\leftarrow u} = |T_2|_{\mathcal{I};x\leftarrow u'} = |T_2|_{\mathcal{I}';x\leftarrow u}$.
- $T = (\alpha : K)T'$. On sait alors que $n = 0$, et $|T|_{\mathcal{I}}$ étant défini comme une intersection de candidats est bien un candidat. La preuve de l'autre hypothèse de récurrence est immédiate.
- $T = (x : T_1)T_2$. Ce cas est similaire au précédent. Il faut toutefois adapter le lemme 3.6 au cas des types dépendants.

■

On vérifie par ailleurs facilement que tout prédicat bien-formé est bien-construit. Il faut juste auparavant définir une adéquation entre contextes et arité-interprétations :

Définition 3.12 (arité-interprétation adaptée) *Une arité-interprétation* $\mathcal{A}$ *est adaptée à un contexte* Γ, *si et seulement si pour tout couple* (α, K) *apparaissant dans* Γ, *on a* $\mathcal{A}(\alpha) = \mathsf{Ar}(K)$.

Lemme 3.11 *Soit un jugement dérivable* $\Gamma \vdash T : K$ *et* $\mathcal{A}$ *une arité-interprétation adaptée à* Γ. *Alors* T *est bien construit d'arité* $\mathsf{Ar}(K)$ *sous* $\mathcal{A}$.

Preuve Par récurrence sur la structure de la dérivation du jugement. ■

À partir de là, on définit une adéquation entre contextes et interprétations. C'est la notion *d'interprétation adaptée* 3.1, revue et corrigée dans le cadre de systèmes avec types dépendants.

Définition 3.13 (interprétation adaptée) *Soit Γ un contexte bien-formé (c.à.d. tel qu'il existe un jugement $\Gamma \vdash J$ dérivable). Une interprétation $\mathcal{I}$ est adaptée à Γ, si :*

- *l'arité-interprétation correspondant par $\mathcal{I}$ est adaptée à Γ*
- *pour tout couple (x, T) apparaissant dans Γ, on a $\mathcal{I}(x) \in |T|_{\mathcal{I}}$.*

Pour s'assurer que cette définition est correcte, il faudrait vérifier les propriétés suivantes :

- si un contexte $\Gamma :: (x, T)\Gamma'$ est bien formé, alors $\Gamma \vdash T : \text{Set}$.
- si $\mathcal{A}$ est adaptée à Γ, alors $\mathcal{A}$ est adaptée à toute sous-partie de Γ.

La seconde propriété ne pose aucun problème. La première est facile à vérifier par récurrence sur la structure de la preuve de bonne-formation de Γ (elle est faite dans le cadre de CCI dans le chapitre précédent).

Nous avons maintenant construit tous les outils nécessaires pour vérifier la normalisation. Il nous reste un point très important à traiter : ce système se distingue des précédents par la présence de la règle de *conversion* :

$$(\textsc{Conv})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_1 : \text{Set} \qquad T_1 =_\beta T_2 \qquad \Gamma \vdash T_2 : \text{Set}}{\Gamma \vdash t : T_2}$$

Lorsque nous prouverons que $t : T \Rightarrow t[\mathcal{I}] \in |T|_{\mathcal{I}}$, il nous faudra traiter le cas de cette règle. On utilisera alors la propriété que l'interprétation des prédicats est *invariante par conversion.*

Lemme 3.12 (invariance par conversion) *Soit T un prédicat bien construit d'ordre n sous une-interprétation $\mathcal{I}$. Si $T\ \mathbf{B}_\beta\ T'$ alors T' est bien-construit d'ordre n sous $\mathcal{I}$, et $|T|_{\mathcal{I}} = |T'|_{\mathcal{I}}$.*

Cette propriété est la clé de l'adaptation de la méthode de réductibilité à des systèmes avec types dépendants. C'est elle qui nous posera le plus de difficultés dans des systèmes plus complexes, comme dans la partie suivante et surtout pour le Calcul des Constructions Inductives dans le chapitre suivant. Toutefois dans le cadre du système $\lambda P2$, la preuve est facile :

PREUVE Par récurrence sur la structure de T. On ne détaille que les cas intéressants où T est le radical réduit ou bien la réduction a lieu dans un terme preuve.

- Si $T = ([x : T_1]T_2\ t)$ et $T' = T_2[x \setminus T_1]$, c'est une conséquence immédiate de la propriété $|T_2[x \setminus t]|_{\mathcal{I}} = |T_2|_{\mathcal{I};x \leftarrow |t|_{\mathcal{I}}}$.
- Si $T = ([\alpha : K]T_1\ T_2)$ et $T' = T_1[\alpha \setminus T_2]$, c'est une conséquence immédiate de la propriété $|T_1[\alpha \setminus T_2]|_{\mathcal{I}} = |T_1|_{\mathcal{I};\alpha \leftarrow |T_2|_{\mathcal{I}}}$.
- Si $T = (T_1\ t)$ et $T' = (T_1\ t')$ avec $t\ \mathbf{B}\ t'$, l'hypothèse de récurrence nous assure que $|T_1|_{\mathcal{I}}$ est un candidat de réductibilité. Aussi comme $|t|_{\mathcal{I}} =_\beta |t'|_{\mathcal{I}}$, on a bien $|T_1|_{\mathcal{I}}(|t|_{\mathcal{I}}) = |T_1|_{\mathcal{I}}(|t'|_{\mathcal{I}})$.

■

Nous pouvons maintenant passer à la preuve de normalisation proprement dite :

Théorème 6 *Soit $\Gamma \vdash t : T$ un jugement dérivable et $\mathcal{I}$ une interprétation adaptée à Γ ; on a alors $t[\mathcal{I}] \in |T|_{\mathcal{I}}$.*

La preuve est une récurrence sur la structure de la dérivation du jugement, sur le modèle de ce qui a été fait précédemment. Il ne semble pas nécessaire de détailler.

Corollaire 3.1 *Si le jugement $\Gamma \vdash t : T$ est dérivable dans $\lambda P2$, alors $\|t\|$ est fortement normalisable.*

3.7 Types définis par récurrence

Il nous reste un point important à illustrer avant de conclure ce chapitre. C'est celui des types définis par récurrence sur des objets inductifs, c'est-à-dire la normalisation de systèmes comportant ce que nous avons appelé des opérateurs d'élimination forte. Il va être facile d'étendre le système $\lambda P2$ en ce sens, puisqu'il possède déjà des types dépendants. Nous considérons les deux types inductifs primitifs déjà étudiés précédemment : les entiers naturels et les ordinaux. L'ensemble des prédicats de $\lambda P2$ est donc étendu par les objets Nat et Ord, et on se donne les nouvelles règles d'inférence suivantes :

$$\boxed{[] \vdash \text{Nat} : \text{Set} \qquad [] \vdash \text{Ord} : \text{Set}}$$

De même l'algebre des *termes de preuves* est étendue par les constructeurs et les schémas d'élimination déja utilisés dans la partie 3.4. Les termes de preuves sont donc maintenant définis par :

$$\boxed{t \;::=\; x \mid [x:T]t \mid (t\ t) \mid [\alpha:K]t \mid (t\ T)}$$

Les règles d'introduction des constructeurs restent essentiellement inchangées. Par contre, et comme indiqué dans le chapitre 1, nous pouvons maintenant typer de manière plus précise les schémas d'élimination :

$$\boxed{\begin{array}{c}
(\text{0-N-Intro})\ [] \vdash 0 : \text{Nat} \qquad (\text{S-N-Intro})\ [] \vdash S : \text{Nat} \to \text{Nat} \\
(\text{0-O-Intro})\ [] \vdash 0_o : \text{Ord} \qquad (\text{S-O-Intro})\ [] \vdash S_o : \text{Ord} \to \text{Ord} \\
(\text{lim-Intro})\ [] \vdash \text{lim} : (\text{Nat} \to \text{Ord}) \to \text{Ord} \\
\\
(\text{Nat-Elim})\ \dfrac{\Gamma \vdash T : \text{Nat} \to \text{Set} \qquad \Gamma \vdash t : \text{Nat} \qquad \Gamma \vdash t_0 : (T\ 0) \qquad \Gamma \vdash t_S : (m : \text{Nat})(T\ m) \to (T\ (S\ m))}{\Gamma \vdash (R_N\ t\ t_0\ t_S) : (T\ nt)} \\
\\
(\text{Ord-Elim})\ \dfrac{\begin{array}{c}\Gamma \vdash T : \text{Ord} \to \text{Set} \qquad \Gamma \vdash t : \text{Ord} \qquad \Gamma \vdash t_0 : (T\ 0_o) \\ \Gamma \vdash t_S : (m : \text{Ord})(T\ m) \to (T\ (S_o\ m)) \\ \Gamma \vdash t_l : (f : \text{Nat} \to \text{Ord})((n : \text{Nat})(T\ (f\ n))) \to (T\ (\text{lim}\ f))\end{array}}{\Gamma \vdash (R_O\ t\ t_0\ t_S\ t_l) : (T\ t)}
\end{array}}$$

Les règles de réduction des opérateurs R_N et R_O sont les mêmes que celles utilisées dans la section 3.4. En particulier, elles étendent la règle de conversion. Comme nous utilisons des *termes purs*, non-typés, dans notre preuve de normalisation, il nous faut là aussi tenir compte des constructeurs de schémas d'élimination. Les termes purs sont donc maintenant définis par :

$$\boxed{t \;:=\; x \mid \lambda x.t \mid (t\ t) \mid 0 \mid S \mid 0_O \mid S_O \mid \text{lim} \mid R_N \mid R_O}$$

et l'opération d'oubli est bien sûr étendue par :

$$\begin{aligned}
\|0\| &\equiv 0 \\
\|S\| &\equiv S \\
\|R_N\| &\equiv R_N \\
&\text{etc}
\end{aligned}$$

avec les règles de réduction maintenant habituelles sur les termes purs.

À ce stade, nous avons défini une extension cohérente de $\lambda P2$. Il serait en revanche peu interessant d'étudier sa normalisation : il suffit littéralement de combiner la preuve du système T (définitions de |Nat| et |Ord|) avec celle de $\lambda P2$. Nous commençons donc par introduire les opérateurs d'élimination forte. Comme indiqué dans le chapitre 2, ces derniers sont construits sur le modèle des opérateurs d'élimination habituels, mais permettent de construire des types. Nous appellerons respectivement Rec et Rec_O les opérateurs d'élimination forte des naturels et des ordinaux. Les prédicats sont finalement décrits par :

$$\begin{array}{|ll|}\hline T := & \alpha \quad | \; [x:T]T \mid (T\ t) \mid (\alpha : K)T \mid (x:T_1)T_2 \\ & \quad | \; \mathrm{Nat} \mid \mathrm{Ord} \mid \mathsf{Rec}(t, T, \alpha.T) \mid \mathsf{Rec}_O(t, T, \alpha.T, \alpha.T) \\ \hline \end{array}$$

Remarquons que Rec et Rec_O lient des variables dans certains de leurs sous-termes. Par exemple dans le terme $\mathsf{Rec}(t, T_0, \alpha.T_S)$, la variable de prédicat α est liée dans le sous-terme T_S, avec les conséquences habituelles pour la substitution. Les règles de réduction sur les prédicats de la section précédente sont étendues par :

$$\begin{array}{l}
\mathsf{Rec}_N(0, T_0, \alpha.T_S)\ \mathbf{B}_\iota\ T_0 \\
\mathsf{Rec}_N((S\ t), T_0, \alpha.T_S)\ \mathbf{B}_\iota\ (T_S[\alpha \setminus \mathsf{Rec}_N(t, T_0, \alpha.T_S)]\ t) \\
\mathsf{Rec}_O(0_O, T_0, \alpha.T_S, \beta.T_l)\ \mathbf{B}_\iota\ T_0 \\
\mathsf{Rec}_O((S\ t), T_0, \alpha.T_S, \beta.T_l)\ \mathbf{B}_\iota\ (T_S[\alpha \setminus \mathsf{Rec}(t, T_0, \alpha.T_S, \beta.T_l)]\ t) \\
\mathsf{Rec}_O((\lim\ t), T_0, \alpha.T_S, \beta.T_l)\ \mathbf{B}_\iota\ (T_l[\beta \setminus [x:\mathrm{Nat}]\mathsf{Rec}((t\ x), T_0, \alpha.T_S, \beta.T_l)]\ t)
\end{array}$$

Remarque On voit que les comportements calculatoires respectifs de Rec et Rec_O sont essentiellement identiques à ceux de R et R_O. Les différences sont simplement dues à l'impossibilité, dans ce système, de construire des fonctions des prédicats vers les prédicats : $[\alpha : K]T$ est interdit. Dans le Calcul des Constructions étendu par Extern, cette restriction n'existe pas, et les opérateurs d'élimination faible et forte se comportent de manière identique.

Passons enfin aux règles d'inférence correspondant aux schémas d'élimination forte. On simplifie la présentation du système en autorisant uniquement la construction d'objets de type Set par élimination forte, contrairement aux Calcul des Construction Inductives, dans lequel il est possible de construire des prédicats d'ordre arbitraire.

$$\begin{array}{|ll|}\hline
(\textsc{Nat-S-Elim}) & \dfrac{\Gamma \vdash t : \mathrm{Nat} \qquad \Gamma \vdash T_0 : \mathrm{Set} \qquad \Gamma :: (\alpha, \mathrm{Set}) \vdash T_S : \mathrm{Nat} \to \mathrm{Set}}{\Gamma \vdash \mathsf{Rec}_N(t, T_0, \alpha.T_S) : \mathrm{Set}} \\[2ex]
(\textsc{Ord-S-Elim}) & \dfrac{\begin{array}{c}\Gamma \vdash t : \mathrm{Ord} \qquad \Gamma \vdash T_0 : \mathrm{Set} \qquad \Gamma :: (\alpha, \mathrm{Set}) \vdash T_S : \mathrm{Ord} \to \mathrm{Set} \\ \Gamma :: (\beta, \mathrm{Nat} \to \mathrm{Set}) \vdash T_l : (\mathrm{Nat} \to \mathrm{Ord}) \to \mathrm{Set}\end{array}}{\Gamma \vdash \mathsf{Rec}_O(t, T_0, \alpha.T_S, \beta.T_l) : \mathrm{Set}} \\[2ex]
\hline
\end{array}$$

On renvoie par exemple au chapitre 1 pour les exemples d'utilisation de cette règle.

3.7.1 La preuve de normalisation

On aura compris que celle-ci reprend la plupart des éléments de la preuve de normalisation de $\lambda P2$, et que la nouveauté essentielle est l'interprétation des types définis par élimination forte. À ce stade, il n'est sans doute pas inutile de redonner la définition précise des candidats de réductibilité, combinaison de 3.2 et de 3.7 :

Définition 3.14 *Un terme pur est neutre (c'est-à-dire élément de $\mathcal{NT}$) s'il n'est pas de la forme :*

$$\lambda x.t \quad 0 \quad (S\ t) \quad 0_O \quad (S_O\ t) \quad (\lim\ t).$$

Définition 3.15 *Pour tout entier naturel n, on définit $\mathcal{CR}(n) \subset \rho(n)$, l'ensemble des* candidats de réductibilité de degré n *:*

- *$\mathcal{CR}(0)$ est l'ensemble des ensembles de termes $\mathcal{C}$ vérifiant les trois conditions* **CR 1**, **CR 2** *et* **CR 3** *déjà énoncées (définition 3.3)*
- *$\mathcal{CR}(n+1)$ est l'ensemble des applications f de Λ vers $\mathcal{CR}(n)$ vérifiant :*

$$\forall t, t'.t =_{\beta\iota} t' \Rightarrow f(t) = f(t').$$

Les notions d'arité-interprétation 3.8 et d'interprétation 3.10 restent inchangées. Comme nous avons restreint l'élimination forte aux prédicats d'arité 0, il est facile d'étendre la notion de bonne-construction :

Définition 3.16

- *Le prédicat $\mathsf{Rec}_N(t, T_0, \alpha.T_S)$ est bien construit d'arité n sous une arité-interprétation $\mathcal{A}$ si et seulement si $n = 0$, T_0 est bien construit d'arité 0 sous $\mathcal{A}$ et T_S est bien construit d'arité 1 sous $\mathcal{A}; \alpha \leftarrow 0$.*
- *Le prédicat $\mathsf{Rec}_O(t, T_0, \alpha.T_S, \beta.T_l)$ est bien construit d'arité n sous une arité-interprétation $\mathcal{A}$ si et seulement si $n = 0$, T_0 est bien construit d'arité 0 sous $\mathcal{A}$, T_S est bien construit d'arité 1 sous $\mathcal{A}; \alpha \leftarrow 0$ et T_l est bien construit d'arité 1 sous $\mathcal{A}; \beta \leftarrow 1$.*

Les autres clauses définissant la bonne-construction restent inchangées par rapport à 3.9.

On vérifie comme précédemment que si $\Gamma \vdash T : K$ est dérivable, et si $\mathcal{A}$ est une arité interprétation adaptée à Γ, alors T est bien-construit d'arité $\mathsf{Ar}(K)$ sous $\mathcal{A}$.

Nous arrivons maintenant au point crucial : la définition de $|T|_{\mathcal{I}}$. Les définitions de $|\alpha|_{\mathcal{I}}, |(x : T_1)T_2|_{\mathcal{I}}, |(\alpha : K)T_1|_{\mathcal{I}}, |(T\ t)|_{\mathcal{I}}$ et $|[x : T_1]T_2|_{\mathcal{I}}$ sont bien sûr identiques à celles données en 3.11. Les ensembles $|\mathrm{Nat}|_{\mathcal{I}}$ et $|\mathrm{Ord}|_{\mathcal{I}}$ sont identiques à $|\mathrm{Nat}|$ et $|\mathrm{Ord}|$ définis dans la partie 3.4 (ils ne dépendent donc pas de $\mathcal{I}$). La difficulté réside bien sûr dans les définitions de $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$ et $|\mathsf{Rec}_O(t, T_0, \alpha.T_S, \beta.T_l)|_{\mathcal{I}}$. Dans un premier temps nous détaillerons seulement le cas des entiers (interprétation de Rec). La difficulté réside dans le traitement de la règle de conversion ; on peut dire que nous devons respecter les "spécifications" suivantes :

- Si $\Gamma \vdash T_0 : \mathrm{Set}$, $\Gamma :: (\alpha, \mathrm{Set}) \vdash T_S : \mathrm{Nat} \to \mathrm{Set}$, et si $\mathcal{I}$ est adaptée à Γ, alors

$$|\mathsf{Rec}_N(0, T_0, \alpha.T_S)|_{\mathcal{I}} = |T_0|_{\mathcal{I}}.$$

  Plus généralement, si $\Gamma \vdash t : \mathrm{Nat}$ et $|t|_{\mathcal{I}}\ \mathbf{B}^*\ 0$, alors

$$|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}} = |T_0|_{\mathcal{I}}.$$

– Si de plus $\Gamma \vdash t : \mathrm{Nat}$, alors

$$|\mathsf{Rec}_N((S\ t), T_0, \alpha.T_S)|_{\mathcal{I}} = |(T_S[\alpha \setminus \mathsf{Rec}_N(t, T_0, \alpha.T_S)]\ t)|_{\mathcal{I}}$$

Comme le deuxième cas est équivalent à

$$|\mathsf{Rec}_N((S\ t), T_0, \alpha.T_S)|_{\mathcal{I}} = |T_S|_{\mathcal{I};\alpha \leftarrow |\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}}(|t|_{\mathcal{I}})$$

on peut voir ces égalités comme une définition récursive de l'application qui à $|t|_{\mathcal{I}}$ et $\mathcal{I}$ associe $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$. Il nous reste à déterminer :

1. dans quel cas cette définition est bien fondée
2. que faire dans les cas où $|t|_{\mathcal{I}}$ ne se réduit ni vers 0 ni vers $(S\ u)$.

La réponse à la deuxième question est simple : si $|t|_{\mathcal{I}}$ ne se réduit pas vers une forme de constructeur, on peut considérer que $\mathsf{Rec}_N(t, T_0, \alpha.T_S)$ est un type atomique, et prendre pour $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$ un candidat de réductibilité arbitraire (typiquement $\boldsymbol{\mathcal{SN}}$).

Pour ce qui est de la première question, on voit que la récurrence considérée est bien fondée, si $|t|_{\mathcal{I}}$ est élément de $|\mathrm{Nat}|_{\mathcal{I}}$ (respectivement $|\mathrm{Ord}|_{\mathcal{I}}$ si on considère Rec_O). Dans le cas des entiers naturels tout d'abord. Supposons que $|t|_{\mathcal{I}}$ admette une forme normale u. Cette forme normale est bien sûr élément de $|\mathrm{Nat}|_{\mathcal{I}}$ qui est égal à $\boldsymbol{\mathcal{SN}}$. On définit alors formellement la fonction f qui à une interprétation $\mathcal{I}$ et un terme normal v associe un ensemble de termes :

$$\begin{array}{rcl} f(\mathcal{I}, 0) & \equiv & |T_0|_{\mathcal{I}} \\ f(\mathcal{I}, (S\ v)) & \equiv & |T_S|_{\mathcal{I};\alpha \leftarrow f(\mathcal{I}, v)}(v) \\ f(\mathcal{I}, v) & \equiv & \boldsymbol{\mathcal{SN}} \text{ si n'est pas de la forme 0 ou } (S\ v). \end{array}$$

Et on définit alors :

Définition 3.17 *Si $|t|_{\mathcal{I}} \in \boldsymbol{\mathcal{SN}}$, et si u est sa forme normale, alors $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$ est défini comme $f(\mathcal{I}, u)$.*

Il nous reste à décider de la valeur de $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$ dans le cas où $|t|_{\mathcal{I}}$ n'est pas normalisable. On sait qu'ultimement, si $\mathsf{Rec}_N(t, T_0, \alpha.T_S)$ est bien formé, et si $\mathcal{I}$ est une interprétation adaptée au contexte correspondant, ceci ne sera jamais le cas. Mais nous ne pouvons, bien sûr, pas encore invoquer cet argument.

Un problème similaire apparaît dans les preuves de normalisation de système de types de Martin-Löf avec univers. La solution adoptée par Per Martin-Löf lui-même [97] et par Catarina Coquand [139], est de ne pas définir $|T|_{\mathcal{I}}$ dans les cas correspondants. Il faut alors également définir le domaine sur lequel l'interprétation est définie. Ensuite on prouve que $|T|_{\mathcal{I}}$ est effectivement définie si T est bien typé et $\mathcal{I}$ vérifie certaines conditions bien-choisies.

La solution que nous proposons est légèrement différente : $|T|_{\mathcal{I}}$ est toujours défini. Dans le cas $|t|_{\mathcal{I}}$ on assigne à $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$ une valeur arbitraire :

Définition 3.17 (suite) *Si $|t|_{\mathcal{I}}$ n'est pas normalisable, alors :*

$$|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}} \equiv \boldsymbol{\mathcal{SN}}.$$

On vérifie alors facilement que $|T|_{\mathcal{I}}$ et toujours un candidat de réductibilité.

Nous simplifions ainsi la définition de l'interprétation, mais ne faisons essentiellement que repousser le problème. En effet, il n'est maintenant plus vrai que si $T =_\beta T'$ alors $|T|_\mathcal{I} = |T'|_\mathcal{I}$. Considérons par exemple :

$$\begin{aligned} T &\equiv [x : \mathrm{Nat}]\mathsf{Rec}((S\ x), T_0, \alpha.[y : nat]T_1) \\ T' &\equiv [x : nat]T_1 \end{aligned}$$

où T_1 est un type ne dépendant ni de x ni de y. Il est évident que $T\ \mathbf{B}\ T'$. Pourtant $|T'|_\mathcal{I}$ est l'application constante qui à tout terme pur associe $|T_1|_\mathcal{I}$, alors que si on applique un terme pur non-normalisable u à $|T|_\mathcal{I}$ on obtient $\boldsymbol{\mathcal{SN}}$ (à cause de la définition précédente et parce que $(S\ u)$ est non-normalisable). Or il est clair qu'il n'y a aucune raison particulière pour que $|T_1|_\mathcal{I} = \boldsymbol{\mathcal{SN}}$. Par contre, pour tout terme pur $v \in |\mathrm{Nat}|_\mathcal{I}$, on aura bien :

$$|T|_\mathcal{I}(v) = |T'|_\mathcal{I}(v) = |T_1|_\mathcal{I}.$$

Dans le cadre du système étudié ici, la solution la plus simple et la plus élégante m'a été indiquée par Milena Stefanova. Il suffit de modifier l'interprétation de la λ-abstraction des types dépendants et de prendre :

$$\begin{aligned} |[x : T_0]T|_\mathcal{I} &\equiv u \mapsto |T|_{\mathcal{I};x\leftarrow u} \text{ pour } u \in |T_0|_\mathcal{I} \\ &\equiv u \mapsto |T|_{\mathcal{I};x\leftarrow x} \text{ pour } u \notin |T_0|_\mathcal{I}. \end{aligned}$$

On ne change pas ainsi les autres étapes de la preuve, et on retrouve bien, pour tous prédicats bien formés T et T' la propriété

$$T =_\beta T' \Rightarrow |T|_\mathcal{I} = |T'|_\mathcal{I}.$$

Malheureusement, il semble que cette modification ne soit pas compatible avec la η-réduction pour des raisons évidentes. Comme notre but ici est d'illustrer les techniques utilisées dans le chapitre suivant, nous conservons notre définition première de l'interprétation des prédicats, et introduisons la propriété pour un prédicat d'être "invariant sur son domaine". Il s'agit simplement de formaliser le fait que lorsque l'on applique à un prédicat uniquement des termes qui sont éléments des interprétations de son domaine de quantification, on obtient un type dont l'interprétation est invariante par réduction. Cette propriété, qui rappelle beaucoup la réductibilité, sera prouvée lors de la récurrence finale.

Définition 3.18 *Soit un jugement dérivable $\Gamma \vdash K$ et une interprétation $\mathcal{I}$ adaptée à Γ. On définit $dom(K, \Gamma, \mathcal{I})$ comme un ensemble de $\mathsf{Ar}(K)$-uplets de termes purs par récurrence sur la structure de K :*

- *si $K = \mathrm{Set}$, alors $dom(K, \Gamma, \mathcal{I})$ est l'unique 0-uplet*
- *si $K = (x : T)K'$, alors $(u, u_1, \ldots, u_n)$ est élément de $dom(K, \Gamma, \mathcal{I})$ si et seulement si $u \in |T|_\mathcal{I}$, et $(u_1, \ldots, u_n) \in dom(K, \Gamma :: (x, T), \mathcal{I}; x \leftarrow u)$.* [3]

Définition 3.19 (invariant sur son domaine) *Soit un jugement dérivable $\Gamma \vdash T : K$ et une interprétation $\mathcal{I}$ adaptée à Γ. On dit que $(T, K, \Gamma, \mathcal{I})$ est invariant sur son domaine si pour tout $\mathcal{A} \in dom(K, \Gamma, \mathcal{I})$, si $T\ \mathbf{B}^*\ T'$, alors*

$$|T|_\mathcal{I}(\mathcal{A}) = |T'|_\mathcal{I}(\mathcal{A}).$$

[3] Pour être tout à fait rigoureux, il faudrait en fait prouver que si $\Gamma \vdash (x : T)K$ est dérivable, alors $\Gamma \vdash T : \mathrm{Set}$ et $\Gamma :: (x, T) \vdash K$ le sont aussi. Ce résultat est en fait démontré dans le chapitre 2 dans un cadre plus général, et sa preuve nous intéresse peu ici.

En particulier si $K = \mathrm{Set}$ *alors* $|T|_{\mathcal{I}} = |T'|_{\mathcal{I}}$.

Nous sommes maintenant parés pour la preuve finale :

Théorème 7 *Pour tout jugement dérivable* $\Gamma \vdash t : T$, $\|t\|$ *est fortement normalisable.*

PREUVE On reste bien sûr fidèle à la récurrence sur la structure de la dérivation. L'hypothèse de récurrence étant :

- si $\Gamma \vdash t : T$ est dérivable, pour toute interprétation $\mathcal{I}$ adaptée à Γ, on a $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$
- si $\Gamma \vdash T : K$ est dérivable, pour toute interprétation $\mathcal{I}$ adaptée à Γ, $(T, K, \Gamma, \mathcal{I})$ est invariant sur son domaine.

On ne détaille que les cas des règles correspondant au second cas, ainsi que la règle de conversion :

- La règle VAR2. C'est immédiat, puisque α est en forme normale.
- Les règles WEAK1 et WEAK2. Il suffit de remarquer que si x (respectivement α) n'est pas libre dans K, alors $\mathrm{dom}(K, \Gamma, \mathcal{I}) = \mathrm{dom}(K, \Gamma :: (x, T_1), \mathcal{I})$ (respectivement $\mathrm{dom}(K, \Gamma :: (\alpha, K_1), \mathcal{I})$).
- La règle T-APP. La conclusion de la règle est $\Gamma \vdash (T\ t) : K[x \setminus t]$. On sait par hypothèse de récurrence que $|t|_{\mathcal{I}} \in |T_1|_{\mathcal{I}}$ et que $(T, (x : T_1)K, \Gamma, \mathcal{I})$ est invariant sur son domaine. Alors pour tout $\mathcal{A} \in \mathrm{dom}(K[x \setminus t], \Gamma, \mathcal{I})$ on a $(|t|_{\mathcal{I}}, \mathcal{A}) \in \mathrm{dom}((x : T)K, \Gamma, \mathcal{I})$. Le résultat suit.
- La règle T-LAM. La conclusion est $\Gamma \vdash [x : T_1]T : (x : T_1)K$. Si $T\ \mathbf{B}^*\ T_0$, alors $T' = [x : T_1']T'$. Soit $(u_1, u_2, \ \ldots\ , u_n) \in \mathrm{dom}((x : T_1)K, \Gamma, \mathcal{I})$; alors $u_1 \in |T_1|_{\mathcal{I}}$ et donc $(T, K, \Gamma :: (x, T_1), \mathcal{I}; x \leftarrow u_1)$ est invariant sur son domaine par hypothèse de récurrence. Comme $(u_2, \ \ldots\ , u_n) \in \mathrm{dom}(K, \Gamma :: (x, T_1), \mathcal{I}; x \leftarrow u_1)$, on en déduit :

  $$|T|_{I;x \leftarrow u_1}(u_2, \ \ldots\ , u_n) = |T'|_{\mathcal{I};x \leftarrow u_1}(u_2, \ \ldots\ , u_n)$$

  c'est-à-dire aussi

  $$|[x : T_1]T|_{\mathcal{I}}(u_1, u_2, \ \ldots\ , u_n) = |[x : T_1']T'|_{\mathcal{I}}(u_1, u_2, \ \ldots\ , u_n).$$

- La règle PROD1. La conclusion est $\Gamma \vdash (x : T_1)T_2 : \mathrm{Set}$. Les réduits de $(x : T_1)T_2$ sont forcément de la forme $(x : T_1')T_2'$. Par hypothèse de récurrence on sait alors que $|T_1|_{\mathcal{I}} = |T_1'|_{\mathcal{I}}$, et si $u \in |T_1|_{\mathcal{I}}$, alors $|T_2|_{\mathcal{I};x \leftarrow u} = |T_2'|_{\mathcal{I};x \leftarrow u}$. on en déduit :

  $$|(x : T_1)T_2|_{\mathcal{I}} = (x : T_1')T_2'|_{\mathcal{I}}.$$

- La règle PROD2. Ce cas est similaire au précédent.
- La règle NAT-S-ELIM. La conclusion est

  $$\Gamma \vdash \mathsf{Rec}_N(t, T_0, \alpha.T_S) : \mathrm{Set}.$$

  On sait par hypothèse de récurrence que $|t|_{\mathcal{I}}$ est fortement normalisable ; on raisonne par récurrence sur sa forme normale.
  - Si $|t|_{\mathcal{I}}\ \mathbf{B}^*\ 0$, alors on sait que $|\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}} = |T_0|_{\mathcal{I}}$. Comme si $T_0\ \mathbf{B}^*\ T_0'$, et que $\mathsf{Rec}_N(t, T_0, \alpha.T_S)$ se réduit soit vers T_0', soit vers $\mathsf{Rec}_N(t', T_0', \alpha.T_S')$, le résultat suit.
  - Si $|t|_{\mathcal{I}}\ \mathbf{B}^*\ (S\ v)$, les réduits de $\mathsf{Rec}_N(t, T_0, \alpha.T_S)$ sont de la forme :
    - $\mathsf{Rec}_N(t', T_0', \alpha.T_S')$ auquel cas $|t'|_{\mathcal{I}}\ \mathbf{B}^*\ (S\ u)$. L'hypothèse de récurrence assure alors $|T_0|_{\mathcal{I}} = |T_0'|_{\mathcal{I}}$, et $|\mathsf{Rec}(u, T_0, \alpha.T_S)|_{\mathcal{I}} = |\mathsf{Rec}(u, T_0', \alpha.T_S')|_{\mathcal{I}}$. Le résultat suit.
    - un réduit de $(T_S'[\alpha \setminus T_2]\ t')$, avec $t\ \mathbf{B}^*\ (S\ t')$ et $\mathsf{Rec}(t', T_0, \alpha.T_S)\ \mathbf{B}^*\ T_2$. Comme substituer α par T_2 dans T_S', on sait que $((T_S'[\alpha \setminus T_2]\ t'), \mathrm{Set}, \Gamma, \mathcal{I})$ est invariant sur son domaine. Par ailleurs les hypothèses de récurrence assurent que $|(T_S'[\alpha \setminus T_2]\ t')|_{\mathcal{I}} = |\mathsf{Rec}_N(t, T_0, \alpha.T_S)|_{\mathcal{I}}$.

– Le cas de la règle CONV. La conclusion est $\Gamma \vdash t : T_2$. On sait que $|t|_{\mathcal{I}} \in |T_1|_{\mathcal{I}}$. Comme $T_1 =_\beta T_2$, T_1 et T_2 admettent un réduit commun T_3. Par ailleurs l'hypothèse de récurrence assure alors $|T_1|_{\mathcal{I}} = |T_3|_{\mathcal{I}}$ et $|T_2|_{\mathcal{I}} = |T_3|_{\mathcal{I}}$, et donc $|t|_{\mathcal{I}} \in |T_2|_{\mathcal{I}}$.

■

On voit que le travail nécessaire pour prouver l'invariance de $|T|_{\mathcal{I}}$ par réduction est non-trivial. Dans le cas du système étudié ci-dessus, il semble toutefois que la solution présentée ici est plus simple que celle consistant à définir $|T|_{\mathcal{I}}$ de manière partielle. C'est un peu moins vrai pour le Calcul des Constructions Inductives complet : la preuve détaillée dans le chapitre 4 est certainement compliquée à cause de ce problème, et il est possible, mais pas sûr, qu'une preuve plus proche de celle de Martin-Löf soit plus claire.

3.8 Petit bestiaire des Candidats de Réductibilité

Nous avons présenté en 3.4 la notion de *candidats de réductibilité*, puis vu comment cette notion était indispensable pour prouver la normalisation de tout système au moins aussi expressif que F. Depuis l'introduction de cette notion par Jean-Yves Girard, un certain nombre de variantes ont été présentées. Il nous a semblé intéressant, quitte à céder au démon de la taxinomie, de faire le point sur ces définitions alternatives et, à travers ce tour d'horizon, de caractériser un peu plus précisément cette notion.

Dans ce chapitre, on considèrera, à titre d'exemple, le système F. On garde les notations de la partie 3.5, à savoir $\mathcal{I}$ pour les interprétations, $|T|_{\mathcal{I}}$ pour l'interprétation du type T sous $\mathcal{I}$, etc.

3.8.1 Caractérisation des candidats

La classe des candidats de réductibilité est un ensemble d'ensembles de termes. Comme nous l'avons mentionné précédemment, l'interprétation de tout type T du système considéré (c'est-à-dire l'ensemble des termes réductibles de type T) sera un candidat de réductibilité. C'est nécessaire puisque nous voulons être capable de quantifier sur toutes les interprétations possibles de types en quantifiant sur l'ensemble des candidats. C'est l'origine de la dénomination de "candidats de réductibilité" : un candidat est susceptible de correspondre à l'ensemble des termes réductibles d'un certain type.

Pour que cette condition soit remplie, il faut et il suffit que $\mathcal{CR}$, l'ensemble des candidats soit clos pour les différentes clauses de la définition des interprétations des types. Dans le cas du système F il faut que :

1. Si $\mathcal{C}_1$ et $\mathcal{C}_2$ sont des candidats, alors $C_1 \to \mathcal{C}_2$ est un candidat.
2. L'intersection d'un ensemble non-vide de candidats de réductibilité est un candidat de réductibilité.

De plus, pour être capable de quantifier sur tous les candidats de réductibilité, $\mathcal{CR}$ ne doit pas être vide. Comme on veut être capable de construire une interprétation adaptée à tout contexte dans laquelle les variables de termes sont interprétées par elles-même ($\mathcal{I}(x) = x$), on peut écrire :

3. Il existe un candidat de réductibilité.
4. Tout candidat de réductibilité contient toute variable de terme.

Une autre condition moins intéressante est que tout élément d'un candidat de réductibilité est fortement normalisable. En effet, une variable de type α peut-être interprétée par un candidat de

réductibilité quelconque, et il nous faut garantir que les éléments de l'interprétation de α vérifient bien la condition que nous voulons prouver ultimement : la normalisation forte. On a donc :

5. $\forall \mathcal{C} \in \mathcal{CR} \,.\, \mathcal{C} \subset \boldsymbol{\mathcal{SN}}$.

L'autre condition clé est à nouveau celle qui nous permettera de traiter le cas de l'abstraction :

$$\frac{\Gamma :: (x, A) \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

Étant donnée une interprétation $\mathcal{I}$ adaptée à Γ, il est facile de ramener l'hypothèse de récurrence à l'énoncé :

$$\forall u \in |A|_{\mathcal{I}}.t[\mathcal{I}; x \leftarrow u] \in |B|_{\mathcal{I}}.$$

Il nous faut alors prouver que $\lambda x.t[\mathcal{I}] \in |A \to B|$, c'est-à-dire :

$$\forall u \in |A|_{\mathcal{I}}.(\lambda x.t[\mathcal{I}]\ u) \in |B|_{\mathcal{I}}.$$

En d'autres termes il nous faut être capable de déduire $(\lambda x.t[\mathcal{I}]\ u) \in |B|_{\mathcal{I}}$ de $t[\mathcal{I}; x \leftarrow u] \in |B|_{\mathcal{I}}$. Si l'on considère que les seules informations dont nous disposons sont :

- $|A|_{\mathcal{I}}$ et $|B|_{\mathcal{I}}$ sont des candidats de réductibilité,
- $t[\mathcal{I}; x \leftarrow u] \in |B|_{\mathcal{I}}$,
- $u \in |A|_{\mathcal{I}}$,

on peut en déduire la dernière condition que doivent vérifier les candidats :

6. $\forall A, B \in \mathcal{CR}.\forall u \in A.\forall t.t[x \setminus u] \in B \Rightarrow (\lambda x.t\ u) \in B$.

Un ensemble d'ensembles de termes qui vérifie les six conditions ci-dessus peut donc être choisi comme définition des candidats de réductibilité (est candidat à la candidature).

3.8.2 Une preuve de normalisation abstraite

Pour nous convaincre que les six critèreres ci-dessus sont suffisants, nous pouvons refaire la preuve de normalisation forte du système F. Dans cette partie, on supposera donc simplement que l'on dispose d'un ensemble d'ensembles de termes $\mathcal{CR}$ qui vérifie nos six conditions.

Pour toute interprétation condidat $\mathcal{I}$ et tout type T, on définit l'interprétation $|T|_{\mathcal{I}}$, un ensemble de termes, comme en 3.5 par récurrence sur la structure de T :

$$\begin{aligned} |\alpha|_{\mathcal{I}} &\equiv \mathcal{I}(\alpha) \\ |A \to B|_{\mathcal{I}} &\equiv |A|_{\mathcal{I}} \to |B|_{\mathcal{I}} \\ |\forall \alpha.A|_{\mathcal{I}} &\equiv \bigcap_{\mathcal{C} \in \mathcal{CR}} |A|_{\mathcal{I};\alpha \leftarrow \mathcal{C}}. \end{aligned}$$

Les conditions 1, 2 et 3 suffisent évidemment à assurer :

Lemme 3.13 *Pour tout type T et toute interprétation candidat $\mathcal{I}$, $|T|_{\mathcal{I}}$ est un candidat de réductibilité.*

La preuve du lemme de substitution reste inchangée :

Lemme 3.14 *Pour tous T, T', α et $\mathcal{I}$:*

$$|T[\alpha \setminus T']|_{\mathcal{I}} = |T|_{\mathcal{I};\alpha \leftarrow |T'|_{\mathcal{I}}}.$$

Il n'est pas non plus nécessaire de changer :

Définition 3.20 *Une interprétation $\mathcal{I}$ est dite adaptée à un contexte Γ si pour tout couple (x, T) de Γ on a $\mathcal{I}(x) \in |T|_{\mathcal{I}}$.*

Et comme d'habitude :

Théorème 8 *Pour tout jugement dérivable $\Gamma \vdash t : T$ et toute interprétation candidat $\mathcal{I}$ adaptée à Γ, on a $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$.*

PREUVE Par récurrence sur la dérivation du jugement. Tous les cas sont identiques à la preuve de 3.5 à l'exception du cas de la règle LAM qui est traité grâce à la condition 6 comme indiqué. ■

Corollaire 3.2 *Si $\Gamma \vdash t : T$ est dérivable, il existe une interprétation candidat $\mathcal{I}$ telle que $t \in |T|_{\mathcal{I}}$.*

PREUVE Comme il existe au moins un candidat de réductibilité $\mathcal{C}$ (condition 3), il suffit de prendre $\mathcal{I}(\alpha) = \mathcal{C}$ pour tout α et $\mathcal{I}(x) = x$ pour tout x. La condition 4 garantit que cette interprétation est adaptée à tout contexte. On a alors $|t|_{\mathcal{I}} = t$ et donc $t \in |T|_{\mathcal{I}}$. ■

Corollaire 3.3 *Si $\Gamma \vdash t : T$ est dérivable, alors t est fortement normalisable.*

PREUVE On prend $\mathcal{I}$ telle que définie précédemment. On a $t \in |T|_{\mathcal{I}}$, et comme $|T|_{\mathcal{I}}$ est un candidat de réductibilite, on a $t \in \mathcal{SN}$. ■

3.8.3 Autres réductions

Il est évident qu'il faut adapter les conditions énoncées lorsque l'on considère d'autres réductions que β comme par exemple l'élimination des entiers du système T ou plus généralement la ι-réduction des types inductifs. Une exception est la η-réduction (essentiellement à cause du lemme de retard de η) qui pour une fois ne pose pas de difficultés. En général il suffira d'étendre la condition 6 au cas où d'autres réductions se produisent en tête.

En λ-calcul typé, la λ-abstraction correspond à l'introduction du type flèche ; l'application à son élimination. La réduction correspond à l'élimination conjointe de ces deux opérateurs. Cette symétrie se retrouve dans toutes les extensions que nous considérons dans cette thèse : un constructeur permet l'introduction d'un type inductif et l'opérateur d'élimination permet précisément d'éliminer ce type ; la rencontre d'un constructeur et d'un schéma d'élimination donne lieu à un nouveau type de radical.

3.8.4 La définition de Girard

C'est la définition originelle de la Thèse d'État de Girard, simplifiée dans [83]. C'est aussi celle que nous avons présentée dans ce chapitre et que nous utilisons dans le chapitre suivant, après l'avoir adaptée aux définitions inductives génériques. Rappellons les trois conditions de Girard :

CR 1	$\mathcal{C} \subset \mathcal{SN}$
CR 2	$\forall t \in \mathcal{C}. \forall t'. (t \in \mathcal{C} \wedge t \mathbf{B} t') \Rightarrow t' \in \mathcal{C}$
CR 3	$\forall t \in \mathcal{NT}. (\forall t'. t \mathbf{B} t' \Rightarrow t' \in \mathcal{C}) \Rightarrow t \in \mathcal{C}$

La condition 5 est exactement **CR 1** et la condition 4 une conséquence immédiate de **CR 3**. Il est aussi immédiat de vérifier que $\mathcal{SN}$ est un candidat au sens de Girard ce qui permet de vérifier la condition 3. On a déja vu comment prouver les conditions 1 et 2 ; ce qu'il faut remarquer, c'est que **CR 2** est utilisé pour prouver la condition 1.

La preuve de la condition 6 est également implicitement présente dans la première preuve de normalisation de F. On remarque que là encore, **CR 2** est uniquement utilisée pour prouver 6, en conjonction, bien sûr avec **CR 3**.

Cette définition présente deux avantages : tout d'abord elle est remarquablement concise et élégante. Ensuite, l'utilisation de la notion de termes neutres permet de l'étendre de manière très uniforme à de nouvelles réductions, comme les types inductifs. Il suffit en effet d'étendre la notion de terme neutre aux formes de termes qui correspondent aux nouveaux radicaux, comme par exemple $R(t, t', t'')$.

On peut en revanche lui reprocher de n'être pas vraiment minimale et de cacher ainsi certains mécanismes de la preuve. En particulier, la condition **CR 2** n'est pas vraiment nécessaire, en ce qu'elle ne correspond vraiment à aucune des 6 conditions que doivent vérifier les candidats. C'est en fait le prix à payer pour avoir transformé la condition 6 en **CR 3** : on a du coup besoin de **CR 2** pour prouver 1 et surtout 6. De plus ces conditions sont prouvées par des récurrences emboitées (ou grace à l'introduction d'un ordre lexicographique) un peu lourdes.

Un autre "défaut" de cette définition est que les candidats de réductibilité ainsi définis ne sont pas clos par réunion. On ne peut donc pas les utiliser pour prouver la normalisation de systèmes avec types union.

3.8.5 Les ensembles saturés de Tait

Cette première alternative à la définition de Girard à été d'abord introduite par Tait [140]. Elle est très facile à comprendre si l'on considère les 6 conditions que nous avons énoncées :

Définition 3.21 (Ensembles saturés) *Un ensemble de λ-termes S est dit saturé si :*

SAT 1 *tout élément de S est fortement normalisable*
SAT 2 $(x\ \vec{N_i}) \in S$ *si tout N_i est fortement normalisable*
SAT 3 *si* $(M[x \setminus N]\ \vec{N}) \in S$ *et N est fortement normalisable, alors* $(\lambda x.M\ N\ \vec{N}) \in S$.

On reconnaît, en gros, trois des six conditions : **SAT 1** est exactement la condition 5, **SAT 2** et **SAT 3** sont respectivement des généralisations de nos conditions 4 et 6. On montre facilement que $\mathcal{SN}$ est saturé ce qui vérifie la condition 3. La preuve de la condition 2 est immédiate.

La preuve de la condition 1 est plus intéressante : elle explique pourquoi les clauses **SAT 2** et **SAT 3** sont plus fortes que les conditions 4 et 6.

Lemme 3.15 *La famille des ensembles saturés de Tait vérifie la condition 1.*

PREUVE Soient S_1 et S_2 deux ensembles saturés. On a :

- Si $t \in S_1 \to S_2$, comme $x \in S_1$ on a $(t\ x) \in S_2$ ce qui implique (**SAT 1**) que $(t\ x) \in \mathcal{SN}$ et donc t est fortement normalisable. $S_1 \to S_2$ vérifie **SAT 1**.
- Soit $(x\ \vec{N}) \in \mathcal{SN}$ et $N \in S_1$, d'après **SAT 2** pour S_2 et **SAT 1** pour S_1 on a $(x\ \vec{N}\ N) \in S_2$. Donc $S_1 \to S_2$ vérifie **SAT 2**.

– Soit $t[x \setminus u] \in S_1 \to S_2$ avec $u \in S$ pour un certain ensemble saturé S. Alors u est fortement normalisable. Soit $v \in S_1$, on a $(t[x \setminus u]\ v) \in S_2$. La clause **SAT 3** pour S_2 assure que $(\lambda x.t\ u\ v) \in S_2$. Donc $S_1 \to S_2$ vérifie **SAT 3** (le cas où l'on inverse S et $S_1 \to S_2$ dans l'énoncé est immédiat).

■

On remarque tout de suite que cette définition est plus proche de notre caractérisation des candidats que la définition de Girard ; aussi les preuves sont elles peut-être très légèrement plus simples. On peut aussi remarquer (nous ne sommes pas les premiers à le faire) que la notion d'ensemble saturé est moins restrictive que la définition de Girard : un candidat de réductibilité est saturé alors que la réciproque est fausse.

Lemme 3.16 *Soit $\mathcal{SAT}$ l'ensemble des ensembles saturés. On a : $\mathcal{CR} \subsetneq \mathcal{SAT}$.*

PREUVE Soit $\mathcal{C} \in \mathcal{CR}$, $\mathcal{C}$ vérifie évidemment **SAT 1** et **SAT 2** (respectivement identique et conséquence de **CR 1** et **CR 3**). La clause **SAT 3** se démontre de la manière usuelle, par récurrence sur le nombre maximal de réductions faisables sur les arguments ; c'est quasi-identique à la preuve que $\mathcal{CR}$ vérifie la condition 6.

Il nous reste à construire un ensemble saturé qui ne soit pas un candidat de réductibilité au sens de Girard. Considérons l'ensemble des termes fortement normalisables, tel que leur forme normale par réduction de tête faible soit de la forme $(x\ \vec{N})$ ou égale à $\lambda x.(\lambda z.zy)$. Plus précisément il s'agit de l'ensemble S_0 définit inductivement par :

– $(x\ \vec{N}) \in \boldsymbol{\mathcal{SN}} \Rightarrow (x\ \vec{N}) \in S_0$
– $\lambda x.(\lambda z.zy) \in S_0$
– $\forall t.\forall \vec{N}.\forall u \in \boldsymbol{\mathcal{SN}}.\forall x.(t[x \setminus u]\ \vec{N}) \in S_0 \Rightarrow (\lambda x.t\ u\ \vec{N}) \in S_0$.

Il est très facile de voir que S_0 est saturé. Par induction, on vérifie également que $\lambda x.(\lambda z.zy)$ est le seul élément de S_0 qui soit une λ-abstraction, et donc $\lambda x.y \notin S_0$. Or comme

$$\lambda x.(\lambda z.zy)\ \mathbf{B}_\beta\ \lambda x.y$$

S_0 ne vérifie pas **CR 2**. ■

Un avantage objectif des ensembles saturés est que leur famille est close par union arbitraire. On peut donc les utiliser pour prouver la normalisation de systèmes avec types union.

Lemme 3.17 *Soit $(S_i)_{i\in I}$ une famille arbitraire d'ensembles saturés. Alors $\bigcup_{i\in I} S_i$ est un ensemble saturé.*

PREUVE Les conditions **SAT 1** et **SAT 3** sont trivialement vérifiées. Si $(M[x\backslash N]\ \vec{N}) \in \bigcup_{i\in I} S_i$ et N est fortement normalisable, alors il existe $i \in I$ tel que $(M[x\backslash N]\ \vec{N}) \in S_i$ et donc $(\lambda x.M\ N\ \vec{N}) \in S_i$.
■

3.8.6 Une variante des ensembles saturés

La définition suivante est une légère variante de la précédente. Elle est due à David McAllester. On ajoute une clause à la définition des ensembles saturés :

SAT 1 tout élément de S est fortement normalisable
SAT 2 $(x\ \vec{N_i}) \in S$ si tout N_i est fortement normalisable
SAT 3 si $(M[x \setminus N]\ \vec{N}) \in S$ et N est fortement normalisable, alors $(\lambda x.M\ N\ \vec{N}) \in S$
SAT 4 si $M \in S$ et $M\ \mathbf{B}^{whd}\ M'$, alors $M' \in S$.

Où $\mathbf{B}^{whd}$ désigne la réduction faible de tête, définie inductivement par :
- $(\lambda x.t\ u)\ \mathbf{B}^{whd}\ t[x \setminus u]$
- $t\ \mathbf{B}^{whd}\ t' \Rightarrow (t\ u)\ \mathbf{B}^{whd}\ (t'\ u)$.

La réduction faible de tête est en fait une stratégie de réduction, en ce sens qu'elle est *déterministe* : tout terme peut se réécrire au plus d'une manière par $\mathbf{B}^{whd}$. Les termes qui n'admettent pas de réduit pour $\mathbf{B}^{whd}$ sont ceux de la forme $\lambda x.t$ ou $(x\ \vec{N})$; on les appelle les *valeurs*. Si la réduction de tête faible termine sur un terme t, elle termine forcément sur une valeur qu'on appelle la valeur de t. La particularité essentielle de $\mathcal{SAT}$ tel que redéfini ci-dessus est :

Lemme 3.18 *Soit $\mathcal{V}$ un ensemble de termes fortement normalisables et de la forme $\lambda x.t$. Alors il existe un et un seul ensemble saturé dont les valeurs soient exactement la réunion de $\mathcal{V}$ et de l'ensemble des termes $(x\ \vec{N})$ fortement normalisables.*

En d'autres termes, tout ensemble saturé est alors caractérisé exactement par l'ensemble de ses valeurs. La structure des ensembles saturés est ainsi plus homogène. On met également mieux en valeur le rôle particulier joué par la réduction de tête.

Nous laisserons le lecteur motivé vérifier que les 6 conditions sont bien vérifiées. On peut remarquer que cette définition est strictement plus restrictive que la précédente et strictement moins que celle de Girard (on peut reprendre pour cela le contre-exemple précédent). Par ailleurs, cette version de $\mathcal{SAT}$ est également close pour l'union.

3.8.7 La définition de Parigot

Dans toutes les définitions considérées jusqu'ici, on définit essentiellement les candidats à l'aide de variantes des conditions 4, 5 et surtout 6 (c'est-à-dire celles qui servent dans la dernière partie de la preuve), puis on vérifie que les conditions 1 et 2 sont aussi vérifiées (c'est-à-dire que l'ensemble des candidats est clos par les constructions de l'interprétation des types). Michel Parigot c'est rendu compte que l'on pouvait simplement inverser le problème et prendre les conditions 1 et 2 pour définir les candidats de réductibilité puis vérifier que les autres conditions étaient alors respectées. On peut s'étonner que personne n'ait eu cette idée plus tôt ; mais il y a une différence essentielle entre cette définition et les précédentes : il s'agit d'une définition *inductive* de l'ensemble $\mathcal{CR}$ et non plus juste d'un prédicat du premier ordre sur les ensembles de termes.

Cette définition a été inventée par Parigot pour une preuve directe de normalisation forte du système F étendu avec des opérateurs classiques (continuations) [117].

Définition 3.22 (Candidats de Réductibilité de Parigot) *On définit l'ensemble $\mathcal{CR}$ d'ensembles de termes comme le plus petit vérifiant :*
- $\boldsymbol{\mathcal{SN}} \in \mathcal{CR}$
- $\forall \mathcal{C}_1, \mathcal{C}_2 \in \mathcal{CR}\ .\ \mathcal{C}_1 \to \mathcal{C}_2 \in \mathcal{CR}$
- *pour toute famille $(\mathcal{C}_i)_{i\in I}$ d'éléments de $\mathcal{CR}$, $\bigcap_{i\in I} \mathcal{C}_i \in \mathcal{CR}$.*

Il est facile de voir que ces trois clauses correspondent à une définition inductive correcte. Bien sûr, les conditions 1, 2 et 3 sont vérifiées par cette nouvelle version de $\mathcal{CR}$. Pour vérifier que tous les éléments d'un candidat au sens de Parigot sont fortement normalisables, il suffit de vérifier que ces candidats sont non-vides (pour passer à la deuxième clause). Or, étant donné un candidat $\mathcal{C}$ de Parigot, on prouve aisément, par récurrence sur la preuve que $\mathcal{C}$ est un candidat, que

$$(x\ \vec{N}) \in \boldsymbol{\mathcal{SN}} \Rightarrow (x\ \vec{N}) \in \mathcal{C}.$$

Ce qui permet également de vérifier la condition 4.

La condition 6 est également prouvée par récurrence sur la preuve que $\mathcal{C} \in \mathcal{CR}$. Plus exactement on renforce l'hypothèse de récurrence et prouve que S vérifie la clause **SAT 3** de Tait :

- Elle est évidemment vérifiée par $\boldsymbol{\mathcal{SN}}$
- Si $u \in S'$ et $t[x \setminus u] \in S$ avec $S = S_1 \to S_2$, alors pour tout $v \in S_1$ on a

$$(t[x \setminus u]\ v) \in S_2$$

  et par hypothèse de récurrence sur S_2

$$(\lambda x.t\ u\ v) \in S_2$$

  ce qui prouve

$$(\lambda x.t\ u) \in S.$$

- Le cas de l'intersection est immédiat.

Il est clair que cette définition est la plus restrictive qui soit, modulo le choix du candidat canonique (ici $\boldsymbol{\mathcal{SN}}$), puisque les deux autres clauses de la définition inductive sont des conditions que doivent nécéssairement vérifier toute famille de candidats.

3.8.8 La définition de Parigot étendue par l'union

Tout comme la définition de Girard, celle de Parigot n'est pas close pour l'union. On peut toutefois facilement y remédier cette fois en l'ajoutant simplement à la définition :

- $\boldsymbol{\mathcal{SN}} \in \mathcal{CR}$
- $\forall \mathcal{C}_1, \mathcal{C}_2 \in \mathcal{CR}. \mathcal{C}_1 \to \mathcal{C}_2 \in \mathcal{CR}$
- pour toute famille $(\mathcal{C}_i)_{i \in I}$ d'éléments de $\mathcal{CR}$, $\bigcap_{i \in I} \mathcal{C}_i \in \mathcal{CR}$
- pour toute famille $(\mathcal{C}_i)_{i \in I}$ d'éléments de $\mathcal{CR}$, $\bigcup_{i \in I} \mathcal{C}_i \in \mathcal{CR}$.

Nous laissons le lecteur vérifier que cette variante vérifie bien les six conditions. Ce n'est pas étonnant puisque $\mathcal{SAT}$ est déja clos pour l'union, ce qui montre que la condition 6 "passe à l'union".

Chapitre 4

Le Théorème de Normalisation Forte

La preuve exposée dans ce chapitre est plus longue et complexe que celles données dans le chapitre 3, mais elle suit à peu près le même cheminement. Nous commençons par définir une notion de *termes purs*, non-typés, sous-jacents aux termes preuves. La partie essentielle du travail est alors de prouver la normalisation forte des termes purs qui correspondent à des termes preuves bien-formés. Les étapes essentielles de cette preuve sont les mêmes que celles de la preuve, donnée en 3.7, de normalisation de $\lambda P2$ étendu par des types définis par récurrence. À partir de là, en en déduit la normalisation forte de tous les termes bien-typés. Voici un plan plus détaillé du chapitre, section par section :

1. On définit un calcul de termes purs. Il s'agit en fait du λ-calcul pur étendu avec une notion non-typée de constructeurs et de filtrage récursif. On définit les notions de réduction et de normalisation de la manière usuelle. On prouve que ce calcul vérifie Church-Rosser, et on définit la "fonction d'extraction", qui associe un terme pur $\overline{t}$ à tout terme preuve t.
2. On définit une notion de candidat de réductibilité pour ce calcul de termes purs, et on vérifie les propriétés usuelles. De plus, on définit une notion de construction inductive pour les candidats, sur le modèle de ce qui se fait usuellement en théorie des ensembles.
3. On définit une notion de *prédicats bien construits*, sur le modèle de ce qui est fait en 3.6 et 3.7. Il s'agit simplement d'une forme affaiblie de typage, qui facilite la construction de l'interprétation dans la section suivante.
4. À chaque prédicat T bien-construit, on associe une interprétation $|T|_{\mathcal{I}}$. En particulier si T est d'ordre Set alors $|T|_{\mathcal{I}}$ sera un ensemble de termes purs. Le paramètre $\mathcal{I}$ correspond à l'interprétation des variables libres de T.
5. On vérifie que si $\mathcal{I}$ vérifie les conditions adéquates, alors $|T|_{\mathcal{I}}$ est invariant, une propriété plus faible que celle d'être un candidat de réductibilité. Ceci est essentiellement un résultat technique nécessaire pour l'étape suivante, qui est essentielle.
6. On montre ensuite que si les variables libres de T sont interprétées par des candidats de réductibilité, alors $|T|_{\mathcal{I}}$ est aussi un candidat de réductibilité.
7. On vérifie un certain nombre de propriétés de l'interprétation des types inductifs.
8. On vérifie que si $T\ \mathbf{B}_\beta\ T'$, alors $|T|_{\mathcal{I}} = |T'|_{\mathcal{I}}$.
9. On fait de même pour $T\ \mathbf{B}_\eta\ T'$.
10. On définit une notion de *prédicats invariants sur leur domaine*. Il s'agit de caractériser les prédicats T dont l'interprétation $|T|_{\mathcal{I}}$ est stable par réduction de T.

11. On vérifie que l'élimination forte est correcte ; c'est-à-dire que les prédicats construits par élimination forte sont invariants sur leurs domaines.
12. On construit une notion d'instanciation de contexte.
13. On dispose alors de tous les outils nécessaires pour prouver la normalisation forte des termes purs : si $\Gamma \vdash t : T$ est dérivable, alors le terme pur $\overline{t}$ est fortement normalisable.
14. On ramène la normalisation forte des termes bien-formés au théorème précédent à l'aide de deux codages successifs.

4.1 Termes purs

On pourrait discuter le choix que non faisons ici, d'introduire des termes purs non-typés. Un désavantages indéniables de cette méthode est le codage pénible et compliqué qui permet de déduire la normalisation forte des termes typés à partir de celle des termes purs sous-jacents. Un avantage appréciable est que, comme le calcul des termes purs est confluent, on évite dans un premier temps toutes les complications dues à la non-confluence sur les termes annotés, et que l'on peut ainsi repousser la preuve de Church-Rosser après la preuve de normalisation 5. Par ailleurs, on obtient ainsi une preuve plus syntaxique, clairement décomposée en plusieurs étapes. Remarquons aussi que l'on distingue ainsi clairement entre les rôles des différentes classes de termes (preuves, prédicats, ordres), plus peut-être que dans des preuves de normalisation plus "sémantiques". Mais ce dernier point est sans doute affaire de goût.

Comme indiqué, nous considérons le λ-calcul pur étendu par un opérateur de filtrage récursif.

Définition 4.1 (Termes purs) *Nous définissons l'algèbre* Λ *des* termes purs *par :*

$$\begin{array}{rcl} u & := & a \mid (u\ u) \mid \lambda a.u \mid \mathsf{Co}(n) \mid \mathsf{match}\ a.\{\overrightarrow{u}\} \\ \vec{u} & := & [] \mid u :: \vec{u} \end{array}$$

Par la suite on s'autorisera à utiliser x, y, z*,etc comme symboles de variables dans le cas de termes purs.*

Définition 4.2 (Réductions de termes purs) *On définit les réécritures suivantes sur les termes purs :*

$$\begin{array}{lcll} (\lambda x.u\ u') & \to_\beta & u[x \setminus u'] & \\ (\mathsf{match}\ a.\{\overrightarrow{u_j}\}\ (\mathsf{Co}(i)\ \vec{v})) & \to_\iota & (u_i[a \setminus \mathsf{match}\ a.\{\overrightarrow{u_j}\}]\ \vec{v}) & \\ \lambda x.(u\ x) & \to_\eta & u & si\ x \notin FV(u) \end{array}$$

On écrira $u\ \mathbf{B}\ u'$ *pour dire que le terme* u *se réécrit en* u' *par une transformation* $\to_\beta$*,* $\to_\eta$ *ou* $\to_\iota$ *d'un de ses sous-termes. On désignera respectivement par* $\mathbf{B}^+, \mathbf{B}^*, =_{\beta\iota}$ *les clôtures transitives, réflexive-transitive et réflexive-symétrique-transitive de* $\mathbf{B}$*.*

4.1.1 Confluence et formes normales

Définition 4.3 (Forme normale) *On dit qu'un terme pur u en forme normale si et seulement s'il n'existe pas de terme v tel que u $\mathbf{B}$ v. Plus simplement, on dit que u est* normal.

Définition 4.4 (Normalisable) *Un terme pur normal v tel que u $\mathbf{B}^*$ v est appellé* une forme normale *de u. On dit alors que u est* normalisable.

Théorème 9 (Confluence) *Soient u, u_1 et u_2 tel que u $\mathbf{B}^*$ u_1 et u $\mathbf{B}^*$ u_2. Alors il existe un terme u_3 tel que u_1 $\mathbf{B}^*$ u_3 et u_2 $\mathbf{B}^*$ u_3.*

Corollaire 4.1 (Church-Rosser) *Soient u_1 et u_2 tels que $u_1 =_{\beta\eta\iota} u_2$. Alors il existe u_3 tel que u_1 $\mathbf{B}^*$ u_3 et u_2 $\mathbf{B}^*$ u_3.*

Corollaire 4.2 *Tout terme pur admet au plus une forme normale.*

Définition 4.5 (Fortement normalisable) *On dit qu'un terme u est* fortement normalisable, *si il n'existe pas de suite $(u_i)_{i\in N}$, tel que $u = u_0$ et u_i $\mathbf{B}$ u_{i+1} pour tout i. On désignera alors par $\nu(u)$ la longueur de la plus longue séquence de réductions sur u.*

4.1.2 Des preuves aux termes purs

On a dit que l'interêt essentiel des termes purs était d'offrir une interprétation des termes preuves dans un calcul confluent. La traduction des preuves en termes purs est relativement standard, à l'exception, peut-être, des schémas d'élimination.

Définition 4.6 (Traduction des preuves) *Soit t un terme preuve. On définit le terme pur $\overline{t}$ par la récurrence structurelle suivante sur t :*

$$
\begin{array}{lcl}
\overline{a} & \equiv & a \\
\overline{(t\ t')} & \equiv & (\overline{t}\ \overline{t'}) \\
\overline{(t\ T)} & \equiv & \overline{t} \\
\overline{[a:T]t} & \equiv & \lambda a.\overline{t} \\
\overline{[\alpha:K]t} & \equiv & \overline{t} \\
\overline{\mathsf{Constr}(i,T)} & \equiv & \mathsf{Co}(i) \\
\overline{\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{f_i}\}} & \equiv & (\mathsf{match}\ a.\{\overrightarrow{\Delta[C_i(I),f_i,[\vec{x}:\vec{A}][c:(I\ \vec{x})](a\ c)]}\}\ \overline{t})
\end{array}
$$

Où $I \equiv \mathsf{Ind}(X:(\vec{z}:\vec{A})\mathrm{Set})\{\overrightarrow{C_i(X)}\}$.

Il faut tout de suite remarquer que la dernière clause de la définition est à première vue malformée, puisque les termes $\Delta[C_i(I), f_i, [\vec{x}:\vec{A}][c:(I\ \vec{x})](a\ c)]$ ne sont pas structurellement plus petits que $\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{f_i}\}$. Toutefois, en utilisant les autres clauses de la définition, et la définition de $\Delta[C(X), f, g]$ on peut définir $\overline{\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{f_i}\}}$ de manière équivalente, et en respectant la condition de récurrence structurelle. Prenons :

$$\overline{\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{f_i}\}} \equiv (\mathsf{match}\ a.\{\overrightarrow{[\![C_i(I), f_i, \lambda\vec{z}.\lambda c.(a\ c)]\!]}\}\ \overline{t})$$

où $\overrightarrow{\bar{z}}$ est la sous-séquence de $\vec{z}$ ne contenant que des variables de preuves :

$$\boxed{\overrightarrow{[]} \equiv [] \quad \overrightarrow{a :: l} \equiv a :: \overrightarrow{l} \quad \overrightarrow{\alpha :: l} \equiv \overrightarrow{l}}$$

et où $[\![C(X), f, g]\!]$ est défini par récurrence sur la structure de $C(X)$:

$$\boxed{\begin{array}{lcl} [\![(X\ \vec{t}), f, F]\!] & \equiv & f \\ [\![(a:T)D(X), f, F]\!] & \equiv & \lambda a.[\![D(X), (f\ a), F]\!] \\ [\![(\alpha:K)D(X), f, F]\!] & \equiv & [\![D(X), f, F]\!] \\ [\![((\vec{x}:\vec{t})(X\ \vec{t'})) \to D(X), f, F]\!] & \equiv & \lambda b.[\![D(X), (f\ b\ \lambda\overrightarrow{\bar{x}}.(F\ \overrightarrow{\bar{t'}}\ (p\ \overrightarrow{\bar{x}}))), F]\!] \end{array}}$$

Le lecteur vérifiera facilement l'équivalence entre ces deux définitions. Il est aussi facile de voir que $FV(\bar{t}) \subset FV(t)$.

Lemme 4.1 *Soient deux termes de preuve bien-formés t et t' et une variable de preuve a. Alors*

$$\overline{t[a \setminus t']} = \bar{t}[a \setminus \bar{t'}].$$

PREUVE Par récurrence sur la structure de t. La seule petite subtilité concerne le cas $t = \mathsf{Elim}(I, Q, \vec{u}, v)\{\overrightarrow{f_i}\}$; il faut alors remarquer que comme t est bien formé on a $I = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$ et que pour tout i,

$$\overrightarrow{\Delta[C_i(I), f_i, g][a \setminus t']} = \overrightarrow{\Delta[C_i(I)[a \setminus t'], f_i[a \setminus t'], g[a \setminus t']]}.$$

■

Lemme 4.2 (Correction de la traduction) *Pour tous termes de preuve bien-formés t et t', si $t\ \mathbf{B}^*\ t'$ alors $\bar{t}\ \mathbf{B}^*\ \bar{t'}$. Plus précisément si $t\ \mathbf{B}^i\ t'$, alors $\bar{t}\ \mathbf{B}^j\ \bar{t'}$ pour un certain $j \leq i$.*

PREUVE Par récurrence sur la structure de t. Il suffit bien sûr de considérer les cas où t est lui-même le radical. Le cas de la β-réduction est une conséquence du lemme précédent. Le cas de la η-réduction est une conséquence de la proposition $FV(\bar{t}) \subset FV(T)$. Il nous reste à étudier le cas de la ι-réduction :

$$t = \mathsf{Elim}(I, Q, \vec{u}, (\mathsf{Constr}(j, I)\ \vec{m}))\{\overrightarrow{f_i}\} \quad \text{et} \quad I = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$$

avec

$$t' = (\Delta[C_j(I), f_j, \mathsf{Fun\_Elim}(I, Q, \vec{f_i})]\ \vec{m})$$

on a alors

$$\begin{array}{rcl} \bar{t} & = & (\mathsf{match}\ a.\{\overrightarrow{\overrightarrow{\Delta[C_i(I), f_i, [\vec{x} : \vec{A}][c : (I\ \vec{x})](a\ c)]}}\}\ (\mathsf{Co}(j)\ \overrightarrow{\vec{m}})) \\ & = & (\mathsf{match}\ a.\{\overrightarrow{[\![C_i(I), \overline{f_i}, \overrightarrow{[\vec{x} : \vec{A}][c : (I\ \vec{x})](a\ c)}]\!]}\}\ (\mathsf{Co}(j)\ \overrightarrow{\vec{m}})) \\ & = & (\mathsf{match}\ a.\{\overrightarrow{[\![C_i(I), \overline{f_i}, \lambda\overrightarrow{\bar{x}}.\lambda c.(a\ c)]\!]}\}\ (\mathsf{Co}(j)\ \overrightarrow{\vec{m}})) \\ \mathbf{B}_\iota & & ([\![C_j(I), \overline{f_j}, \lambda\overrightarrow{\bar{x}}.\lambda c.(\mathsf{match}\ a'.\{\overrightarrow{[\![C_i(I), \overline{f_i}, \lambda\overrightarrow{\bar{x}}.\lambda c'.(a'\ c')]\!]}\}\ c)]\!]\ \overrightarrow{\vec{m}}) \\ & = & ([\![C_j(I), \overline{f_j}, \lambda\overrightarrow{\bar{x}}.\lambda c.\overrightarrow{\mathsf{Elim}(I, Q, \_, c)\{\overrightarrow{f_i}\}}]\!]\ \overrightarrow{\vec{m}}) \\ & = & ([\![C_j(I), \overline{f_j}, \overrightarrow{\mathsf{Fun\_Elim}(I, Q, \vec{f_i})}]\!]\ \overrightarrow{\vec{m}}) \\ & = & \overrightarrow{(\Delta[C_j(I), f_j, \mathsf{Fun\_Elim}(I, Q, \vec{f_i})]\ \vec{m})} \\ & = & \overline{t'} \end{array}$$

■

4.2 Candidats de réductibilité

Nous pouvons maintenant passer à la définition des candidats de réductibilité. Pour les explications informelles, nous renvoyons au chapitre 3, auquel il ne devrait pas y avoir grand chose à rajouter ici. On commence par définir pour chaque ordre K l'ensemble $\rho(K)$ qui contiendra les interprétations des termes de type K.

4.2.1 Interprétations des ordres

Définition 4.7 *Pour tout ordre K tel que $Ar(K, \mathrm{Set})$, on définit l'ensemble $\rho(K)$:*
- *$\rho(\mathrm{Set})$ est l'ensemble des parties de Λ, c'est-à-dire l'ensemble des ensembles de termes purs.*
- *$\rho((\alpha : K_1)K_2)$ est l'ensemble des applications de $\rho(K_1)$ vers $\rho(K_2)$*
- *$\rho((x : T_1)K_2)$ est l'ensemble des applications de Λ vers $\rho(K_1)$.*

On appellera aussi $dom(K)$ le domaine de $\rho(K)$:
- *$dom(\mathrm{Set})$ est un singleton quelconque ; par exemple $\{\emptyset\}$*
- *$dom((\alpha : K_1)K_2)$ est le produit cartésien $\rho(K_1) \times dom(K_2)$*
- *$dom((x : T_1)K_2)$ est le produit cartésien $\Lambda \times dom(K_2)$.*

Dans la suite de la preuve, on s'autorisera à parler d'objets fonctionnels à curryfication près. Par exemple on identifiera souvent $\rho(K)$ et l'ensemble des applications de $\mathrm{dom}(K)$ vers $\rho(\mathrm{Set})$.

4.2.2 Les Candidats

Les candidats de réductibilité d'ordre K sont alors des éléments de $\rho(K)$ vérifiant les bonnes conditions de clôture. On aura donc $\mathcal{CR}(K) \subset \rho(K)$. Comme on l'a vu dans le chapitre 3, en présence de types dépendants, les candidats de réductibilité ont deux caractéristiques :
- Les éléments de $\mathcal{CR}(\mathrm{Set})$ sont les ensembles de termes purs possédant les caractéristiques habituelles définies par Girard.
- Les candidats de réductibilité d'ordres de la forme $(a : T)K$, qui prennent donc un terme pur en argument, ne dépendent que de la classe d'équivalence de ce terme modulo la relation $=_{\beta\eta\iota}$.

De plus, on doit maintenant considérer les ordres de la forme $(\alpha : K)K'$. Les candidats correspondants sont simplement les applications associant un candidat d'ordre K' à tout candidat d'ordre K.

On commence par formaliser la seconde condition à travers les deux définitions suivantes :

Définition 4.8 *Pour chaque ordre K, on définit dans $\rho(K)$ la relation d'équivalence partielle notée $\simeq_K$ par récurrence sur la structure de K :*
- $\forall \mathcal{C}, \mathcal{C}' \in \rho(\mathrm{Set}).\mathcal{C} \simeq_{\mathrm{Set}} \mathcal{C}' \Leftrightarrow \mathcal{C} = \mathcal{C}'$
- $\forall \mathcal{C}, \mathcal{C}' \in \rho((a : T)K).\mathcal{C} \simeq_{(a:T)K} \mathcal{C}' \Leftrightarrow (\forall u, u' \in \Lambda.u =_{\beta\eta\iota} u' \Rightarrow \mathcal{C}(u) \simeq_K \mathcal{C}'(u'))$
- $\forall \mathcal{C}, \mathcal{C}' \in \rho((\alpha : K_1)K_2).C \simeq_{(\alpha:K_1)K_2} \mathcal{C}' \Leftrightarrow (\forall \mathcal{D}, \mathcal{D}' \in \rho(K_1).\mathcal{D} \simeq_{K_1} \mathcal{D}' \Rightarrow \mathcal{C}(\mathcal{D}) \simeq_{K_2} \mathcal{C}'(\mathcal{D}'))$.

Par la suite on s'autorisera à écrire simplement $\mathcal{C} \simeq \mathcal{C}'$ en omettant l'ordre.

Définition 4.9 (Invariant) *Soit $\mathcal{C} \in \rho(K)$. On dit que $\mathcal{C}$ est invariant si et seulement si $\mathcal{C} \simeq \mathcal{C}$.*

Pour la suite, il est pratique de définir la relation correspondant à $\simeq_K$ pour $\mathrm{dom}(K)$. On surcharge le symbole $\simeq$ à cet effet.

Définition 4.10 *On définit dans $dom(K)$ la relation d'équivalence partielle $\simeq_K$ par récurrence sur la structure de K :*

- $\forall \mathcal{A}, \mathcal{A}' \in dom(\mathrm{Set}).\mathcal{A} \simeq_{\mathrm{Set}} \mathcal{A}'$ [1]
- $\forall \mathcal{A}, \mathcal{A}' \in dom(K).\forall u, u' \in \Lambda.(u, \mathcal{A}) \simeq_{(a:T)K} (u', \mathcal{A}') \Leftrightarrow (u =_{\beta\eta\iota} u' \wedge \mathcal{A} \simeq_K \mathcal{A}')$
- $\forall \mathcal{A}, \mathcal{A}' \in \rho(K_1).\forall \mathcal{B}, \mathcal{B}' \in dom(K_2).(\mathcal{A}, \mathcal{B}) \simeq_{(\alpha:K_1)K_2} (\mathcal{A}', \mathcal{B}') \Leftrightarrow (\mathcal{A} \simeq_{K_1} \mathcal{A}' \wedge \mathcal{B} \simeq_{K_2} \mathcal{B}')$.

Lemme 4.3 *Pour tous $\mathcal{C}$ et $\mathcal{C}'$ éléments de $\rho(K)$ on a*

$$\mathcal{C} \simeq \mathcal{C}' \Leftrightarrow (\forall \mathcal{A}, \mathcal{A}' \in dom(K).\mathcal{A} \simeq \mathcal{A}' \Rightarrow \mathcal{C}(A) = \mathcal{C}'(\mathcal{A}')).$$

On définit ensuite les candidats de réductibilité d'ordre Set. Les notions de termes neutres et de candidats de réductibilité sont de simples adaptations de ce qui est fait dans le chapitre 3 :

Définition 4.11 (Termes neutres) *Un terme pur est dit* neutre, *si il n'est ni une abstraction $\lambda x.u$, ni de la forme* $\mathsf{match}\ a.\{\overrightarrow{u}\}$, *ni de la forme* $(\mathsf{Co}(i)\ \vec{m})$[2].

Définition 4.12 (Candidats d'ordre Set) *On appelle* candidat de réductibilité d'ordre Set *tout ensemble $\mathcal{C}$ de termes purs tel que :*

CR 1 *tout élément de $\mathcal{C}$ est fortement normalisable.*

CR 2 $\forall u \in \mathcal{C}.\forall v.u\ \mathbf{B}\ v \Rightarrow v \in \mathcal{C}$

CR 3 *pour tout terme neutre u, si quelque soit v tel que $u\ \mathbf{B}\ v$ on a v élément de $\mathcal{C}$, alors u est aussi élément de $\mathcal{C}$.*

On appellera $\mathcal{CR}(\mathrm{Set})$, l'ensemble des candidats de réductibilité d'ordre Set.

La définition des candidats de réductibilité d'ordres supérieurs est essentiellement une combinaison des définitions précédentes :

Définition 4.13 (Candidats de réductibilité) *Pour chaque arité d'ordre* Set, *K on définit le sous-ensemble $\mathcal{CR}(K)$ de $\rho(K)$:*

- $\mathcal{CR}(\mathrm{Set})$ *est défini ci-dessus.*
- $\mathcal{CR}((a:T)K)$ *est l'ensemble des éléments invariants $\mathcal{C}$ de $\rho((a:T)K)$ tels que $\mathcal{C}(\Lambda) \subset \mathcal{CR}(K)$.*
- $\mathcal{CR}((\alpha:K_1)K_2)$ *est l'ensemble des éléments invariants $\mathcal{C}$ de $\rho((\alpha:K_1)K_2)$ tels que*

$$\mathcal{C}(\mathcal{CR}(K_1)) \subset \mathcal{CR}(K_2).$$

On remarque bien sûr que tout candidat de réductibilité est invariant.

La caractérisation suivante de $\mathcal{CR}(K)$ est presque triviale, mais il est utile de l'énoncer :

Définition 4.14 *Pour tout ordre K on définit le sous-ensemble $dom\_\ C(K)$ par récurrence sur la structure de K :*

- $dom\_\ C(\mathrm{Set}) \equiv dom(\mathrm{Set})$
- $dom\_\ C((a:T)K) \equiv \Lambda \times dom\_\ C(K)$
- $dom\_\ C((\alpha:K_1)K_2) \equiv \mathcal{CR}(K_1) \times dom\_\ C(K_2)$

Lemme 4.4 *Soit $\mathcal{C} \in \rho(K)$. Si $\mathcal{C}$ est invariant, alors*

$$\mathcal{C} \in \mathcal{CR}(K) \Leftrightarrow \forall \mathcal{A} \in dom\_\ C(K).\mathcal{C}(\mathcal{A}) \in \mathcal{CR}(\mathrm{Set}).$$

[1] Rappelons que dom(Set) est réduit à un élément.

[2] Comme dans le chapitre 3, il s'agit essentiellement de décrire les termes susceptibles d'être des radicaux (neutres), ou ceux qui, lorsqu'ils sont substitués dans un terme, peuvent créer de radicaux (non-neutres). Le fait que $\mathsf{match}\ a.\{\overrightarrow{u}\}$ soit considéré comme non-neutre est simplement dû au fait qu'ici, il ne peut être un radical à lui tout seul, mais le devient lorsqu'on lui applique un argument avec un constructeur en tête.

4.2.3 Propriétés de clôture de $\mathcal{CR}$(Set)

Il s'agit des propriétés essentielles des candidats de réductibilité, qui assureront ultimement que les interprétations des prédicats bien formés seront effectivement des candidats de réductibilité.

Remarque Tout terme neutre en forme normale appartient à tout élément de $\mathcal{CR}$(Set). Ceci s'applique en particulier à toute variable. Il n'y a donc pas de candidat de réductibilité d'ordre Set qui soit vide.

Lemme 4.5 (Intersection de Candidats) *Soit $(\mathcal{C}_i)_{i\in I}$ une famille de candidats de réductibilité de* Set*, indicée par un ensemble quelconque I. Alors $\bigcap_{i\in I}\mathcal{C}_i$ est un candidat de réductibilité de* Set*.*

La preuve est facile et bien connue, voir le chapitre 3 par exemple.

Définition 4.15 (Demi-treillis complets - rappel) *Soit un ensemble $\mathcal{A}$ muni d'un ordre $<$. On dit que $(\mathcal{A}, <)$ est un inf-demi-treillis complet, si toute partie de $\mathcal{A}$ admet une borne inférieure.*

Corollaire 4.3 *$(\mathcal{CR}(\text{Set}), \subset)$ est un inf-demi-treillis complet.*

Lemme 4.6 *Soit $\mathcal{C}$ un ensemble non vide de termes purs fortement normalisables. Soit f une application de Λ vers $\mathcal{CR}$(Set), telle que pour tous u, u', éléments de $\mathcal{C}$, si $u =_{\beta\eta\iota} u'$, alors $f(u) = f(u')$. Alors l'ensemble $f^{\mathcal{C}}$ défini comme suit est un candidat de réductibilité de* Set *:*

$$f^{\mathcal{C}} \equiv \{u \in \Lambda, \forall v \in \mathcal{C}.(u\ v) \in f(v)\}$$

On remarque bien sûr que ce lemme s'applique en particulier au cas $\mathcal{C} \in \mathcal{CR}$(Set). Dans le cas où la valeur de f est une constante $\mathcal{C}'$, on écrira aussi $\mathcal{C} \to \mathcal{C}'$ pour $f^{\mathcal{C}}$.

Preuve Elle ne pose pas de problèmes :

- Si $u \in f^{\mathcal{C}}$ alors en particulier $(u\ x) \in f(x)$; donc $(u\ x)$ et aussi u sont fortement normalisables.
- Si $u \in f^{\mathcal{C}}$, alors $(u\ v) \in f(v)$ pour tout v dans $\mathcal{C}$. Comme $f(v)$ est un candidat de réductibilité, si $u\ \mathbf{B}\ u'$ on a aussi $(u'\ v) \in f(v)$, et donc $u' \in f^{\mathcal{C}}$.
  On peut alors remarquer que $f^{\mathcal{C}}$ peut aussi être défini comme :

  $$\{u \in \Lambda, \forall v \in \mathcal{C}.v\ \mathbf{B}^*\ v'.(u\ v') \in f(v)\}$$

  c'est-à-dire que $f^{\mathcal{C}} = f^{\mathcal{C}'}$ où $\mathcal{C}'$ est la cloture de $\mathcal{C}$ pour $\mathbf{B}$.
- Soit alors u neutre tel que pour tout u', $u\ \mathbf{B}\ u' \Rightarrow u' \in f^{\mathcal{C}}$. Soit $v \in \mathcal{C}$; on montre que si $(u\ v)\ \mathbf{B}\ w$, alors $w \in f(v)$ (cette proposition suffisant à établir que $u \in f^{\mathcal{C}}$ car $(u\ v)$ est neutre). La preuve se fait par récurrence sur la longueur de la plus longue chaîne de réductions commençant par v. En effet, $(u\ v)$ peut se réduire en :
  – $(u'\ v)$ qui appartient à $f(v)$ par hypothèse de départ.
  – $(u\ v')$ qui appartient à $f(v')$ par hypothèse de récurrence. Or $f(v) = f(v')$.
  Comme u est neutre, il n'y a pas d'autres réductions possibles.

■

Clôture par réduction de tête inverse

Lemme 4.7 (Clôture par β-réduction de tête inverse) *Soient deux termes t et t_1 et un candidat de réductibilité $\mathcal{C} \in \mathcal{CR}(\text{Set})$. Si t_1 est fortement normalisable, et si $t[x \setminus t_1] \in \mathcal{C}$, alors $(\lambda x.t\ t_1) \in \mathcal{C}$. En d'autres termes, si $t[x \setminus t_1] \in \mathcal{C}$, alors $\lambda x.t \in \{t_1\} \to \mathcal{C}$.*

PREUVE La démonstration est similaire à celle du lemme 4.6, par récurrence sur $\nu(t) + \nu(t_1)$ (rappelons que $\nu(u)$ est la longueur de la plus longue suite de réductions sur le terme fortement normalisable u). ■

Corollaire 4.4 *Soient t, un terme pur, $\mathcal{C}$ un candidat de réductibilité d'ordre* Set*, $\vec{x}_i$ et $\vec{u}_i$ respectivement une séquence de variables et une séquence de termes de même longueur. Si pour tout i, t_i est fortement normalisable, et si $t[\vec{x}_i \setminus \vec{u}_i] \in \mathcal{C}$, alors $(\lambda \vec{x}_i.t\ \vec{u}_i) \in \mathcal{C}$.*

PREUVE Par récurrence sur la longueur de $\vec{x}_i$. Le cas de la liste vide est trivial. Par ailleurs si

$$t[x :: \vec{x} \setminus u :: \vec{u}] \in \mathcal{C}$$

il nous faut vérifier que

$$(\lambda x :: \vec{x}.t\ u :: \vec{u}) \in \mathcal{C}$$

or

$$(\lambda x :: \vec{x}.t\ u :: \vec{u}) = (\lambda x.\lambda \vec{x}.t\ u\ \vec{u})$$

et comme u est fortement normalisable, on se ramène à prouver

$$(\lambda \vec{x}.t[x \setminus u]\ \vec{u}) \in \mathcal{C}$$

ce qui est vrai par hypothèse de récurrence. ■

Enfin nous pouvons établir le lemme correspondant pour la ι-réduction.

Lemme 4.8 (Clôture par ι-réduction de tête inverse) *Pour tout candidat de réductibilité $\mathcal{C}$ d'ordre* Set*, pour toutes séquences de termes purs fortement normalisables $\vec{f}_i$ et $\vec{m}$, pour tout j inférieur ou égal à la longueur de $\vec{f}_i$, on a :*

$$(\mathsf{match}\ a.\{\overrightarrow{f_i}\}\ (\mathsf{Co}(j)\ \vec{m})) \in \mathcal{C} \Leftrightarrow (f_j[a \setminus \mathsf{match}\ a.\{\overrightarrow{f_i}\}]\ \vec{m}) \in \mathcal{C}.$$

PREUVE La condition nécessaire est évidente. La condition suffisante se montre simplement, par récurrence sur $\nu(\vec{m}) + \nu(\vec{f_i})$. ■

4.2.4 Demi-Treillis des Candidats d'Ordre Quelconque

La structure de demi-treillis définie ci-dessus peut se généraliser aux candidats de réductibilité d'ordre quelconque. On commence par définir l'ordre.

Définition 4.16 (Ordre sur les Candidats) *Pour tout ordre K avec $Ar(K, \text{Set})$, on définit la relation binaire $\prec$ sur $\rho(K)$ par récurrence sur la structure de K :*

- $\forall C, C' \in \rho(\text{Set}).C \prec C' \Leftrightarrow C \subset C'$.
- $\forall C, C' \in \rho((x : T)K')\ .\ C \prec C' \Leftrightarrow (\forall u \in \Lambda.C(u) \prec C'(u))$.
- $\forall C, C' \in \rho((\alpha : K')K'')\ .\ C \prec C' \Leftrightarrow (\forall u \in \rho(K')\ .\ C(u) \prec C'(u))$.

On peut également définir la généralisation de l'intersection :

Définition 4.17 *Pour tout ordre K avec $Ar(K, \mathrm{Set})$, pour toute famille $(C_a)_{a\in A}$ d'éléments de $\rho(K)$ on définit $\bigwedge_{a\in A} C_a$, élément de $\rho(K)$ par récurrence sur la structure de K :*
- $\forall (C_a)_{a\in A} \in \rho(\mathrm{Set})^A \ . \ \bigwedge_{a\in A} C_a \equiv \bigcap_{a\in A} C_a$.
- $\forall (C_a)_{a\in A} \in \rho((x:T)K)^A \ . \ \bigwedge_{a\in A} C_a \equiv u \in \Lambda \mapsto \bigwedge_{a\in A} C_a(u)$.
- $\forall (C_a)_{a\in A} \in \rho((\alpha:K')K'')^A \ . \ \bigwedge_{a\in A} C_a \equiv u \in \rho(K') \mapsto \bigwedge_{a\in A} C_a(u)$.

Il est facile de voir que si $(C_a)_{a\in A} \in \rho(K)^A$ on a bien $\bigwedge_{a\in A} C_a \in \rho(K)$. La propriété importante est bien sûr :

Lemme 4.9 *Soit un ordre K et une famille $(C_a)_{a\in A}$ d'éléments de $\rho(A)$. Alors :*

$$\forall b \in A. \bigwedge_{a\in A} C_a \prec C_b.$$

On peut voir que $\mathcal{CR}(K)$ est aussi clos par $\wedge$:

Lemme 4.10 *Soit un ordre K et une famille $(C_a)_{a\in A}$ d'éléments de $\rho(A)$. Si tous les C_a sont invariants, alors il va de même de $\bigwedge_{a\in A} C_i$.*

Lemme 4.11 *Soit un ordre K et $(C_a)_{a\in A}$ une famille d'éléments de $\mathcal{CR}(K)^A$. Alors on a aussi $\wedge_{a\in A} C_a \in \mathcal{CR}(K)$.*

Corollaire 4.5 *Pour tout ordre K, $(\mathcal{CR}(K), \prec)$ est un inf-demi-treillis. En particulier, on définit pour tout K, comme $Min(K)$ le plus petit élément de $\mathcal{CR}(K)$ au sens de $\prec$.*

Cette dernière propriété est importante, puisqu'elle nous permettra, dans le paragraphe suivant, de définir le plus petit point fixe de tout opérateur croissant sur $\mathcal{CR}(K)$.

4.2.5 Candidats de Réductibilité Canoniques

Dans le précédent paragraphe, on a construit un candidat de réductibilité minimal pour chaque ordre K. On peut de même construire le candidat maximal.

Définition 4.18 *On appelle $Can(\mathrm{Set})$ l'ensemble des termes purs fortement normalisables.*

Il est facile de voir que $Can(\mathrm{Set})$ est un candidat de réductibilité d'ordre Set (voir par exemple le chapitre précédent). A partir de là on peut construire un candidat de réductibilité canonique pour tout ordre K.

Définition 4.19 (Candidat canonique) *Soit K une arité de sorte* Set. *On définit $Can(K)$ par récurrence sur la structure de K :*
- *$Can(\mathrm{Set})$ est défini ci-dessus.*
- *$Can((x:T)K')$ est l'application constante associant $Can(K')$ à tout terme pur.*
- *$Can((\alpha:K')K'')$ est l'application constante associant $Can(K'')$ à tout élément de $\rho(K')$.*

Lemme 4.12 *Pour tout ordre K, $Can(K)$ est un candidat de réductibilité d'ordre K. De plus, pour tout élément C de $\mathcal{CR}(K)$, on a $C \prec Can(K)$.*

4.2.6 Définitions inductives de Candidats de Réductibilité

Dans cette partie, on définit les outils nécessaires à la construction de l'interprétation des types inductifs. Essentiellement, on construit une petite théorie de définitions inductives en Théorie des Ensembles, adaptée à ce dont on aura besoin par la suite. Il est toutefois apparu plus simple de parler effectivement des familles $\rho(K)$ et $\mathcal{CR}(K)$ plutôt que d'abstraire leurs propriétés utilisées ici. Ce paragraphe utilise quelques propriétés élémentaires sur les ordinaux (voir par exemple [85, 90]).

Tout comme $\bigwedge$ généralisait l'intersection à tous les $\rho(K)$, on peut définir la généralisation de la réunion $\bigvee$.

Définition 4.20 *Pour tout ordre K avec $Ar(K, \mathrm{Set})$, pour toute famille $(C_a)_{a\in A}$ d'éléments de $\rho(K)$ on définit $\bigvee_{a\in A} C_a$, élément de $\rho(K)$ par récurrence sur la structure de K :*

- $\forall (C_a)_{a\in A} \in \rho(\mathrm{Set})^A. \bigvee_{a\in A} C_a \equiv \bigcup_{a\in A} C_a$.
- $\forall (C_a)_{a\in A} \in \rho((x : T)K)^A. \bigvee_{a\in A} C_a \equiv u \in \Lambda \mapsto \bigvee_{a\in A} C_a(u)$.
- $\forall (C_a)_{a\in A} \in \rho((\alpha : K')K'')^A. \bigvee_{a\in A} C_a \equiv u \in \rho(K') \mapsto \bigvee_{a\in A} C_a(u)$.

Lemme 4.13 *Soit un ordre K et deux familles $(C_a)_{a\in A}$ et $(D_a)_{a\in A}$ d'éléments de $\rho(K)$. Si pour tout a élément de A on a $C_a \simeq D_a$, alors on a aussi $\bigvee_{a\in A} C_a \simeq \bigvee_{a\in A} D_a$. En particulier si tous les C_a sont invariants, il en va de même de $\bigvee_{a\in A} C_a$.*

Preuve Par récurrence sur la structure de K. ∎

Il est également facile de voir que $C_b \prec \bigvee_{a\in A} C_a$. En revanche, $\mathcal{CR}(K)$ n'est pas clos pour $\bigvee$, car la réunion de deux candidats de réductibilité n'est pas forcément un candidat de réductibilité. Par contre, Si $\{C_a\}_{a\in A}$ est une partie de $\mathcal{CR}(K)$ totalement ordonnée pour $\prec$, alors $\bigvee_{a\in A} C_a$ est un candidat de réductibilité :

Lemme 4.14 *Soit $(C_a)_{a\in A}$ une famille totalement ordonnée d'éléments de $\mathcal{CR}(K)$. Alors*

$$\bigvee_{a\in A} C_a \in \mathcal{CR}(K).$$

Preuve Par récurrence sur la structure de K. Si $K = \mathrm{Set}$, $\bigcup_{a\in A} C_a$ vérifie évidemment les conditions **CR 1** et **CR 2**. Il nous reste à montrer **CR 3** : soit t neutre, tel que si t **B** t', alors t' appartient à $\bigcup_{a\in A} C_a$. Cela revient à dire que pour tout t', il existe un $a \in A$, tel que $t' \in C_a$ [3]. Les t' étant en nombre fini, on peut choisir le plus grand de ces C_a, qui contiendra donc tous les t'. Ce C_a étant un candidat, on a $t \in C_a$, et donc $t \in \bigcup_{a\in A} C_a$.

Si $K = (a : T)K'$, alors $\bigvee_{a\in A} C_a$ est l'application qui à tout terme pur u associe $\bigvee_{a\in A} C_a(u)$. On a vu qu'elle était invariante. On vérifie ensuite que $(C_a(u))_{a\in A}$ est totalement ordonné. Soient a et b deux éléments de A, on a $C_a \prec C_b$ ou $C_b \prec C_a$, ce qui implique respectivement $C_a(u) \prec C_b(u)$ et $C_b(u) \prec C_a(u)$. Alors, par hypothèse de récurrence, on sait aussi que $\bigvee_{a\in A} C_a(u) \in \mathcal{CR}(K')$. Par ailleurs, si $u =_{\beta\eta\iota} u'$, pour tout $a \in A$, on a bien $C_a(u) = C_a(u')$, et donc $\bigvee_{a\in A} C_a(u) = \bigvee_{a\in A} C_a(u')$. On en conclus $\bigvee_{a\in A} C_a \in \mathcal{CR}(K)$.

Le cas $K = (\alpha_1 : K_1)K_2$ est similaire au précédent. ∎

Lemme 4.15 *Soit $(C_a)_{a\in A}$ une famille d'éléments de $\rho(K)$ et $\mathcal{C} \in \rho(K)$. Si pour tout $a \in A$, $C_a \prec \mathcal{C}$, alors $\bigvee_{a\in A} C_a \prec \mathcal{C}$.*

[3] Remarquons que l'on n'utilise pas l'axiome du choix : pour chaque t', il suffit, par exemple, de choisir le plus petit C_a contenant t'.

PREUVE Par récurrence immédiate sur la structure de K. ■

On considère maintenant une fonction F de $\rho(K)$ vers $\rho(K)$, croissante pour l'ordre $\prec$:

$$\forall \mathcal{C}, \mathcal{C}' \in \rho(K) \ . \ \mathcal{C} \prec \mathcal{C}' \Rightarrow F(\mathcal{C}) \prec F(\mathcal{C}').$$

Par ailleurs on désigne par $\emptyset(K)$ le plus petit élément de $\rho(K)$, c'est-à-dire l'application qui associe $\emptyset$ à tout élément de $\mathrm{dom}(K)$. Pour ne pas avoir à traiter le cas dégénéré, on suppose $F(\emptyset(K)) \neq \emptyset(K)$.

Comme $\rho(K)$ est clos pour $\bigwedge$ et que $(\rho(\mathrm{Set}), \prec)$ est un inf-demi-treillis, il est bien connu que F admet un plus petit point fixe $lfp(F)$. Il suffit de prendre :

$$lfp(F) \equiv \bigwedge \{\mathcal{C} \in \rho(K), F(\mathcal{C}) \prec \mathcal{C}\}.$$

On cherche maintenant à montrer comment la construction inductive de $lfp(F)$ permet de définir un pré-ordre bien-fondé sur $lfp(F)$. On commence par définir l'itérée transfinie de F :

Définition 4.21 *Soit $\mathcal{C} \in \rho(K)$ et $\mathfrak{a}$ un ordinal. On définit l'itérée d'ordre $\mathfrak{a}$ de F sur $\mathcal{C}$, notée $F^{\mathfrak{a}}(\mathcal{C})$ par l'induction transfinie suivante sur $\mathfrak{a}$:*

- $F^0(\mathcal{C}) \equiv \mathcal{C}$
- *pour tout ordinal $\mathfrak{b}$, $F^{\mathfrak{b}+1}(\mathcal{C}) \equiv F(F^{\mathfrak{b}}(\mathcal{C}))$*
- *pour tout ensemble $\mathfrak{A}$ d'ordinaux, $F^{lim(\mathfrak{A})}(\mathcal{C}) \equiv \bigvee_{\mathfrak{b} \in \mathfrak{A}} F^{\mathfrak{b}}(\mathcal{C})$.*

Lemme 4.16 *Soit $\mathcal{K}$ un sous-ensemble de $\rho(K)$ vérifiant la condition du lemme 4.14 : si $(C_a)_{a \in A}$ est une famille totalement ordonnée d'éléments de $\mathcal{K}$, alors $\bigvee_{a \in A} C_a \in \mathcal{K}$. Si de plus F vérifie les conditions suivantes :*

- $F(\emptyset(K)) \in \mathcal{K}$
- $\forall \mathcal{C} \in \mathcal{K}. F(\mathcal{C}) \in \mathcal{K}$

alors $lfp(F) \in \mathcal{K}$. On a vu que c'est le cas si $\mathcal{K} = \mathcal{CR}(K)$, ou si $\mathcal{K}$ est l'ensemble des éléments invariants de $\rho(K)$.

PREUVE On a vu que $lfp(F) = F^{\mathfrak{a}}(\emptyset(K))$. La preuve est alors immédiate par induction transfinie sur $\mathfrak{a}$. ■

Lemme 4.17 *Soit $\mathfrak{a}$ un ordinal ; on a $F^{\mathfrak{a}}(\emptyset(K)) \prec lfp(F)$.*

PREUVE Par induction transfinie sur $\mathfrak{a}$. Le cas 0 est immédiat, car $F^0(\emptyset(K)) = \emptyset(K) \prec lfp(F)$. Par ailleurs, si $F^{\mathfrak{a}}(\emptyset(K)) \prec lfp(F)$, alors

$$F^{\mathfrak{a}+1}(\emptyset(K)) = F(F^{\mathfrak{a}}(\emptyset(K))) \prec F(lfp(F)) = lfp(F).$$

Enfin le cas de l'ordinal limite est une conséquence immédiate du lemme 4.15 et de l'hypothèse de récurrence. ■

Comme pour tout ordinal $\mathfrak{a}$, on a $F^{\mathfrak{a}}(\emptyset(K)) \prec lfp(F)$, pour des raisons de cardinalité évidentes, $F^{\mathfrak{a}}(\emptyset(K))$ reste constant à partir d'un certain ordinal $\mathfrak{a}_0$. On a donc $lfp(F) = F^{\mathfrak{a}}(\emptyset(K))$. Une conséquence est :

Corollaire 4.6 *Si $F(\emptyset(K)) \in \mathcal{CR}(K)$, et si $F(\mathcal{CR}(K)) \subset \mathcal{CR}(K)$, alors $lfp(F) \in \mathcal{CR}(K)$.*

PREUVE Immédiate, en utilisant le lemme précédent et le lemme 4.15. ■

De plus, à tout élément de $lfp(F)$, on peut associer l'ordinal correspondant au plus petit nombre d'itérations nécessaires pour l'atteindre :

Définition 4.22 *Soit $v \in dom(K)$ et $u \in lfp(F)(v)$. On définit* $\deg(u,v)$ *comme le plus petit ordinal tel que $u \in F^{\deg(u,v)}(\emptyset(K))(v)$.*

Lemme 4.18 *Quels que soient u et v,* $\deg(u,v)$ *est toujours un ordinal successeur.*

PREUVE Il est évident que $\deg(u,v) \neq 0$, puisque $F^0(\emptyset(K))(v) = \emptyset$. Supposons que $\deg(u,v)$ soit un ordinal limite ; on aurait

$$u \in (\bigvee_{\mathfrak{a} \in \deg(u,v)} F^{\mathfrak{a}}(\emptyset(K)))(v) = \bigcup_{\mathfrak{a} \in \deg(u,v)} F^{\mathfrak{a}}(\emptyset(K))(v)$$

c'est-à-dire que $u \in F^{\mathfrak{a}}(\emptyset(K))(v)$ pour un certain $\mathfrak{a} < \deg(u,v)$, ce qui est impossible, puisque $\deg(u,v)$ est le plus petit ordinal vérifiant cette condition. ■

Définition 4.23 *À tout $v \in dom(K)$ et tout $u \in lfp(F)(v)$ on associe $pred(u,v) \subset lfp(F)$ défini comme $F^{\deg(u,v)-1}(\emptyset(K))$.*

Définition 4.24 *Soient $\mathcal{C}$ et $\mathcal{C}'$ deux éléments de $\rho(K)$. On dit que $\mathcal{C} <_F \mathcal{C}'$ si $\mathcal{C} = F^{\mathfrak{a}}(\emptyset(K))$ et $\mathcal{C}' = F^{\mathfrak{b}}(\emptyset(K))$ avec $\mathfrak{a} < \mathfrak{b}$.*

L'ordre partiel $<_F$ est évidemment bien fondé. De plus, si $u \in F^{\mathfrak{a}}(\emptyset(K))(v)$, alors $pred(u,v) <_F F^{\mathfrak{a}}(\emptyset(K))$.

4.3 Prédicats bien construits

A priori, on cherche à définir l'interprétation des prédicats bien formés. Pour des raisons pratiques, il est plus facile de définir l'interprétation sur une classe plus large de prédicats, que nous appellerons les prédicats *bien-construits*. Cette notion correspond en fait à une notion faible de typage, où l'on ne tient pas compte du type des termes preuves. Cette notion est suffisante pour déterminer à quel ensemble $\rho(K)$ appartiendra l'interprétation d'un prédicat donné.

Définition 4.25 (Ordre-interprétations) *On appelle ordre-interprétation une application $\mathcal{J}$ qui associe un élément de $\{\rho(K)\}$ à toute variable de prédicat α. On notera $\mathcal{J};\alpha \leftarrow \rho(K)$ l'ordre-interprétation valant $\rho(K)$ en α et identique à $\mathcal{J}$ en tous les autres points. Par abus de langage on s'autorisera aussi à la noter $\mathcal{J};\alpha \leftarrow K$.*

Définition 4.26 (Bonne-construction) *Soit un ordre K, un prédicat T et une ordre-interprétation $\mathcal{J}$. On dit que T* est correctement construit d'ordre K sous $\mathcal{J}$, *si et seulement si :*

- *$T = \alpha$ et $\rho(K) = \mathcal{J}(\alpha)$.*
- *$T = (a : T_1)T_2$ avec $K =_{\beta\eta\iota}$* Set, *et T_1 T_2 tous deux bien construits d'ordre* Set *sous $\mathcal{J}$.*
- *$T = (\alpha : K_1)T_1$ avec $K =_{\beta\eta\iota}$* Set *et T_1 bien construit d'ordre* Set *sous $\mathcal{J};\alpha \leftarrow K_1$.*
- *$T = (T_1\ T_2)$ s'il existe deux ordres K_1 et K_2 avec T_2 bien construit d'ordre K_2 sous $\mathcal{J}$, T_1 bien construit d'ordre $(\alpha : K_2)K_1$ sous $\mathcal{J}$ et $\rho(K) = \rho(K_1[\alpha \setminus T_2])$ (c.à.d. en fait $\rho(K) = \rho(K_1)$).*

- $T = (T_1\ t)$ *s'il existe* T_2 *et* K' *tel que* T_1 *soit bien construit d'ordre* $(a : T_2)K_1$ *sous* $\mathcal{J}$ *et* $\rho(K) = \rho(K_1[x \setminus t])$ *(c.à.d. en fait* T_1 *bien construit d'ordre* $(a : T_1)K$*).*
- $T = [a : T_1]T_2$ *s'il existe* K_2 *tel que* T_2 *soit bien construit d'ordre* K_2 *sous* $\mathcal{J}$ *et* $\rho(K) = \rho((a : T_1)K_2)$*.*
- $T = [\alpha : K_1]T_2$ *s'il existe* K_2 *tel que* T_2 *soit bien construit d'ordre* K_2 *sous* $\mathcal{J}; \alpha \leftarrow K_1$ *et* $\rho(K) = \rho(\alpha : K_1)K_2$*.*
- $T = [a : T_1]T_2$ *s'il existe* K_2 *avec* T_2 *bien construit d'ordre* K_2 *sous* $\mathcal{J}$ *et* $\rho(K) = \rho((a : T_1)K_2)$*.*
- $T = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$ *avec :*
  - A *est de la forme* $(\vec{x} : \vec{A})\mathrm{Set}$
  - $\rho(A) = \rho(K)$
  - *tous les* $C_i(X)$ *sont des types de constructeurs*
  - *Tous les* $C_i(X)$ *sont bien construits d'ordre* Set *sous* $\mathcal{J}; X \leftarrow A$*.*
- $T = \mathsf{Elim}(I, Q, \vec{a}, t)\{\overrightarrow{T_j}\}$ *avec :*
  - $I = \mathsf{Ind}(X : A)\{\overrightarrow{C_j(X)}\}$ *et* I *est bien construit sous* $\mathcal{J}$ *(forcément d'ordre* A*)*
  - $(Q\ \vec{a}\ t)$ *est un ordre et* $K =_{\beta\eta\iota} (Q\ \vec{a}\ t)$*.*
  - *pour tout* i*,* T_i *est bien construit d'ordre* $\Delta\{C_i(I), Q, \mathsf{Constr}(i, I)\}$*.*

Il faut noter que la propriété d'être bien construit d'ordre K ne dépend que de $\rho(K)$.

Remarque Un prédicat T est bien construit d'ordre K sous une ordre-interprétation $\mathcal{J}$ si et seulement si $\|T\|$ est bien-construit d'ordre K sous $\mathcal{J}$.

La propriété de bonne construction nous permettra de définir les interprétations des types et des prédicats en fonction de leur ordre. Avant cela, nous pouvons d'ores et déjà nous assurer que les ordres bien typés sont bien construits :

Définition 4.27 (Ordre-interprétation compatible avec un contexte) *Une ordre interprétation* $\mathcal{J}$ *dite est compatible avec un contexte* Γ *si pour tout couple* (α, K) *dans* Γ*,* $\mathcal{J}(\alpha) = K$*.*

Lemme 4.19 *S'il existe une dérivation d'un jugement de typage* $\Gamma \vdash T : K$*, alors* T *est bien construit d'ordre* K *sous toute ordre-interprétation compatible avec* Γ*.*

4.4 Interprétation des prédicats

Comme expliqué dans le chapitre 3, l'interprétation d'un prédicat est définie modulo la donnée de l'interprétation de chacune de ses variables libres ; aussi bien les variables de preuves que les variables de prédicats. D'où la définition suivante.

Définition 4.28 (Interprétations) *On appelle* interprétation *une application* $\mathcal{I}$ *associant un terme pur à toute variable de preuve, et un élément de* $\bigcup_K \rho(K)$ *à toute variable de prédicat. Si de plus pour tout* α*,* $\mathcal{I}(\alpha)$ *est invariant (respectivement candidat d'ordre* K*), on dira que* $\mathcal{I}$ *est une* interprétation invariante *(respectivement une* interprétation candidat*).*

Remarque Il est facile de voir que les $\rho(K)$ sont deux-à-deux soit égaux, soit disjoints. Aussi toute interprétation définit de manière univoque une ordre-interprétation $\mathcal{J}$ tel que pour tout α on ait : $\mathcal{I}(\alpha) \in \mathcal{J}(\alpha)$. A partir de là, si un prédicat est bien construit d'un ordre K sous $\mathcal{J}$, on dira aussi qu'il est bien construit d'ordre K sous $\mathcal{I}$.

Définition 4.29 (Interprétation des preuves) *Toute interprétation définit une transformation sur les termes purs, par substitution de ses variables libres. On désignera par $|u|_{\mathcal{I}}$ le terme pur u dont toute variable libre x a été substituée par $\mathcal{I}(x)$. Par la suite, on s'autorise à écrire $|t|_{\mathcal{I}}$ pour $|\bar{t}|_{\mathcal{I}}$ pour tout terme de preuve t.*

Définition 4.30 (Interprétation des prédicats) *Soient T, K et une interprétation $\mathcal{I}$ tels que T soit bien construit d'ordre K sous $\mathcal{J}$. On définit alors par récurrence sur T l'élément $|T|_{\mathcal{I}}$ de $\rho(K)$:*

$$\begin{aligned}
|\alpha|_{\mathcal{I}} &= \mathcal{I}(\alpha)\\
|(x:T_1)T_2|_{\mathcal{I}} &= \{u\in\Lambda, \forall u_1\in|T_1|_{\mathcal{I}}.(u\ u_1)\in|T_2|_{\mathcal{I};x\leftarrow u_1}\}\\
|(\alpha:K_1)T_1|_{\mathcal{I}} &= \bigcap_{\mathcal{C}\in\mathcal{CR}(K_1)}|T_1|_{\mathcal{I};\alpha\leftarrow C}\\
|(T_1\ t)|_{\mathcal{I}} &= |T_1|_{\mathcal{I}}(|t|_{\mathcal{I}})\\
|(T_1\ T_2)|_{\mathcal{I}} &= |T_1|_{\mathcal{I}}(|T_2|_{\mathcal{I}})\\
|[x:T_1]T_2|_{\mathcal{I}} &= u\in\Lambda\mapsto|T_2|_{\mathcal{I};x\leftarrow u}\\
|[\alpha:K_1]T_1|_{\mathcal{I}} &= \mathcal{C}\in\mathcal{CR}(K_1)\mapsto|T_1|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}
\end{aligned}$$

Ce qui précède est classique, et correspond à ce que l'on pourrait définir pour le Calcul des Constructions pur. Il nous reste à définir les interprétations des prédicats inductifs :

$$\mathsf{Ind}(X:A)\{\overrightarrow{C(X)}\}$$

et des prédicats définis par élimination forte :

$$\mathsf{Elim}(I,Q,\vec{x},t)\{\vec{F}\}.$$

Par ailleurs, on peut déjà remarquer, par simple récurrence structurelle, que pour la définition partielle ci-dessus, si T est bien construit d'ordre K sous $\mathcal{I}$, alors $|T|_{\mathcal{I}}$ est effectivement élément de $\rho(K)$.

Définition 4.30 (suite) *Soit*

$$I=\mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}$$

avec

$$A\equiv(\vec{y_k}:\vec{A_k})\mathrm{Set},$$

et pour tout i,

$$C_i(X)=(\vec{x}_i^j:\overrightarrow{T_i^j(X)})(X\ \vec{t}_i^k).$$

Soit aussi une interprétation $\mathcal{I}$, telle que I soit bien construit d'ordre A sous $\mathcal{I}$. On définit $|I|_{\mathcal{I}}$ comme le plus petit élément de $\rho(K)$ tel que $\min(K)\prec|I|_{\mathcal{I}}$ et vérifiant la condition suivante : Pour tout $\mathcal{A}\in dom(A)$, tout terme pur u élément de $|I|_{\mathcal{I}}(\mathcal{A})$, pour tout $\mathcal{C}\in\mathcal{CR}((\vec{y_k}:\vec{A_k})\alpha\to\mathrm{Set})$, étant donnée une séquence de termes $\vec{f_i}$ avec pour tout i :

$$f_i\in|\Delta\{C_i(X),Q,c\}|_{\mathcal{I};X\leftarrow|I|_{\mathcal{I}};Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}$$

on a :

$$(\mathsf{match}\ a.\{\overrightarrow{|\Delta[C_i(X),g_i,[\vec{y_k}:\vec{A_k}][c:(I\ \vec{y_k})](a\ c)]|_{\mathcal{I};\forall i.g_i\leftarrow f_i}}\}\ u)\in\mathcal{C}(\mathcal{A})(u)$$

c'est-à-dire aussi, de manière équivalente :

$$(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X),f_i,a]\!]}\}\ t)\in\mathcal{C}(\mathcal{A})(t).$$

La définition ci-dessus est la formulation récursive d'une définition inductive sur le modèle considéré en 4.2.6. La définition équivalente en terme de point-fixe est la suivante :

$|I|_{\mathcal{I}}$ est le plus petit point fixe de l'application F de $\rho(A)$ vers $\rho(A)$ définie par : pour tout $S \in \rho(A)$, pour tout $\mathcal{A} \in dom(A)$, on définit $F(S)(\mathcal{A}) \in \rho(\mathrm{Set})$ comme la réunion de $\min(\mathrm{Set})$ *avec l'ensemble des termes purs u, tels que pour toute séquence de termes purs f_i, avec pour tout i*

$$f_i \in |\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}$$

on ait :

$$(\mathsf{match}\ a.\{|\Delta[C_i(X), g_i, [\vec{y_k} : \vec{A}_k][c : (I\ \vec{y}_k)](a\ c)]|_{\mathcal{I};\forall i.g_i\leftarrow f_i}\overrightarrow{\}\ u}) \in \mathcal{C}(\mathcal{A})(u).$$

On appelle $<_{(I,\mathcal{I})}$ l'ordre bien-fondé sur $\rho(A)$ induit par la définition inductive de $|I|_{\mathcal{I}}$.

Il faut tout de suite vérifier que l'application F est effectivement monotone en S pour l'ordre $\prec$; pour cela, on prouve :

$$S \prec S' \Rightarrow |\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow S';Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)} \subset |\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

La preuve est bien sûr par récurrence sur la structure du type de constructeur $C_i(X)$:

- Si $C_i(X) = (X\ \vec{t'})$ alors $\Delta\{C_i(X), Q, c\} = (Q\ \vec{t'}\ c)$ et donc $|\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}$ ne dépend pas de S.
- Le cas $C_i(X) = (x : T)D(X)$ est une simple conséquence de l'hypothèse de récurrence, puisque $|T|_{\mathcal{I}}$ ne dépend pas de S et

$$S \prec S' \Rightarrow |\Delta\{D(X), Q, c\}|_{\mathcal{I};X\leftarrow S';Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)} \subset |\Delta\{D(X), Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

- Si $C_i(X) = ((\vec{x} : \vec{t})(X\ \vec{t'})) \to D(X)$, alors

$$\Delta\{C_i(X), Q, c\} = (p : (\vec{x} : \vec{t})(X\ \vec{t'}))((\vec{x} : \vec{t})(Q\ \vec{t'}\ (p\ \vec{x}))) \to \Delta\{D(X), Q, c\}.$$

Grâce à l'hypothèse de récurrence, on sait que :

$$S \prec S' \Rightarrow |\Delta\{D(X), Q, c\}|_{\mathcal{I};X\leftarrow S';Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)} \subset |\Delta\{D(X), Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

Par ailleurs on peut vérifier par récurrence sur $\vec{t}$, que

$$S \prec S' \Rightarrow |(\vec{x} : \vec{t})(X\ \vec{t'})|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)} \subset |(\vec{x} : \vec{t})(X\ \vec{t'})|_{\mathcal{I};X\leftarrow S';Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

Le résultat suit alors par simple utilisation de la définition de $|(x : A)B|_{\mathcal{I}}$.

On voit aussi que cette clause de la définition ne respecte pas la récurrence structurelle, puisque $\Delta\{C_i(X), Q, c\}$ n'est pas un sous-terme de $\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$. On peut toutefois corriger cela en définissant $|\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}|_{\mathcal{I}}$ par récurrence sur les différents $C_i(X)$, en remplaçant toutes les occurrences d'un $|(x : A)B|_{\mathcal{J}}$ par sa définition en fonction de $|A|_{\mathcal{J}}$ et $|B|_{\mathcal{J};x\leftarrow\mathcal{C}}$. Comme il me semble que cela nuirait plutôt à la clarté, je préfère faire grâce de la définition rigoureuse au lecteur.

Exemple (Entiers naturels) Lorsque $I = \mathrm{Nat} \equiv ind(X : \mathrm{Set})\{X; X \to X\}$, on retrouve la définition du chapitre 3. L'ensemble de termes $|\mathrm{Nat}|_{\mathcal{I}}$ est alors défini par l'ensemble des termes u tel que pour tout candidat $\mathcal{C}$ d'ordre $\mathrm{Nat} \to \mathrm{Set}$, pour tous f_0 et f_S tels que

$$\begin{aligned} f_0 &\in |(Q\ c)|_{\mathcal{I};Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(1)} \\ f_S &\in |(p : X)(Q\ p) \to (Q\ (c\ p))|_{\mathcal{I};X\leftarrow|\mathrm{Nat}|_{\mathcal{I}};Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(2)} \end{aligned}$$

c'est-à-dire en fait :

$$\begin{array}{rcl} f_0 & \in & |(Q\ O)|_{\mathcal{I};Q\leftarrow\mathcal{C}} \\ f_S & \in & |(p:\mathrm{Nat})(Q\ p)\rightarrow(Q\ (S\ p))|_{\mathcal{I};Q\leftarrow\mathcal{C}} \end{array}$$

on ait :

$$(\mathsf{match}\ a.\{f_0;\lambda p.\lambda x.(f_S\ p\ (\lambda y.(a\ y)\ x))\}\ u)\in(\mathcal{C}\ u).$$

Il est facile de vérifier qu'un terme u est élément de $|\mathrm{Nat}|_{\mathcal{I}}$ si et seulement si :

1. u est fortement normalisable.
2. l'une des conditions suivantes est vérifiée :
   - u se réduit vers $O\equiv\mathsf{Constr}(1,\mathrm{Nat})$
   - u se réduit vers $(S\ v)$ avec $v\in\|\mathrm{Nat}|_{\mathcal{I}}$
   - u ne se réduit pas vers une des formes ci-dessus.

Autrement dit, $|\mathrm{Nat}|_{\mathcal{I}}$ est l'ensemble des termes fortement normalisables, tels que s'ils se réduisent vers $(S\ v)$, alors v est élément de $|\mathrm{Nat}|_{\mathcal{I}}$.

La preuve n'est pas très difficile. Nous ne la donnons pas, car elle serait redondante avec les résultats à venir. Remarquons aussi que $|\mathrm{Nat}|_{\mathcal{I}}$ ne dépend pas de $\mathcal{I}$.

Remarque Si $(\mathsf{Ind}(X:A)\{\overrightarrow{C(X)}\}\ \vec{u})$ est bien construit d'ordre Set sous $\mathcal{I}$, alors on a toujours :

$$|(\mathsf{Ind}(X:A)\{\overrightarrow{C(X)}\}\ \vec{u})|_{\mathcal{I}}=|\mathsf{Ind}(X:A)\{\overrightarrow{C(X)}\}|_{\mathcal{I}}(|\vec{u}|_{\mathcal{I}}).$$

Nous pouvons maintenant définir $|\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{T_i}\}|_{\mathcal{I}}$ dans le cas où les constructeurs de I sont tous petits. Rappelons que l'idée est de prévoir, dans l'interprétation, toutes les ι-réductions qui pourront être effectuées. Dans [150], on définissait l'interprétation par récurrence sur la forme normale de $|t|_{\mathcal{I}}$. Ici, comme le type inductif I peut être plus complexe que les entiers naturels, nous raisonnerons par récurrence sur la preuve que $|t|_{\mathcal{I}}\in|(I\ \vec{u})|_{\mathcal{I}}$. Formellement il s'agit d'une récurrence sur $(|\vec{u}|_{\mathcal{I}},|t|_{\mathcal{I}})$ bien-fondée vis-à-vis de l'ordre $<_{(I,\mathcal{I})}$.

Définition 4.30 (suite et fin) *On considère $I=\mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}$ bien construit d'ordre*

$$A\equiv(\vec{y_k}:\vec{A_k})\mathrm{Set}$$

sous $\mathcal{I}$. On définit

$$|\mathsf{Elim}(I,Q,\vec{u},t)\{\overrightarrow{F_i}\}|_{\mathcal{I}}\equiv G(|I|_{\mathcal{I}})(|\vec{u}|_{\mathcal{I}},|t|_{\mathcal{I}})$$

où

$$G(H)\in\rho((\vec{y_k}:\vec{A_k})(c:(I\ \vec{y_k}))(Q\ \vec{y_k}\ c))$$

est défini pour tout $H\in dom(<_{(I,\mathcal{I})})$ [4] par les clauses ci-dessous.

On se donne $\mathcal{A}\equiv\vec{\mathcal{A}_k}\in dom(A)$; c'est-à-dire tel que pour tout k, $\mathcal{A}_k\in\rho(A_k)$ si y_k est une variable de prédicat et $\mathcal{A}_k\in\Lambda$ si y_k est une variable de preuve. On se donne aussi $v\in\Lambda$. Il nous faut donc définir $G(H)(\mathcal{A})(v)\in\rho(\mathrm{Set})$. On procède par cas :

[4] C'est-à-dire que $H\in\rho(A)$, et il existe $H'\in\rho(A)$ tel que $H<_{(I,\mathcal{I})}H'$. Ou, de manière équivalente, il existe un ordinal $\mathfrak{a}$ tel que $H=F^{\mathfrak{a}}(\emptyset(A))$.

1. *Si v n'admet pas de forme normale, ou si la forme normale de v n'est pas de la forme $(\mathsf{Co}(i)\ \vec{m})$, alors*
$$G(H)(\mathcal{A})(v) \equiv Can(set)$$
dans tous les autres cas, on appellera $w \equiv (\mathsf{Co}(i)\ \vec{m})$ la forme normale de v [5].
2. *Si $w \notin H(\mathcal{A})$, alors on prend de même*
$$G(H)(\mathcal{A})(u) \equiv Can(set)$$
c'est en particulier le cas si $w \notin |I|_{\mathcal{I}}(\mathcal{A})$.
3. *Enfin si $w \in H(\mathcal{A})$, alors*
$$G(H)(\mathcal{A})(u) \equiv |\Delta[C_i(I), F_i, g]|_{\mathcal{I};g \leftarrow G(pred(\mathcal{A},w))}(\vec{m})$$
où g est une variable "fraîche".

C'est la dernière clause 3 qui est bien sûr le cas crucial. En fait on effectue "à la main" la ι-réduction de tête. *L'utilisation de la variable auxiliaire g sert simplement à simuler l'appel récursif du* Elim. Si l'on regarde la structure de $(\Delta[C_i(I), F_i, g]\ \vec{m})$, on voit que g ne se verra appliqué qu'à des éléments de $\vec{m}$ (auxquels on aura éventuellement appliqué des arguments). Cette définition récursive est bien-fondée par rapport au pré-ordre $<_{(I,\mathcal{I})}$.

Comme cela a déjà été le cas précédemment, la dernière clause, telle qu'elle est formulée ici ne respecte pas les conditions de récurrence structurelle, puisque
$$\Delta[C_i(I), F_i, g]$$
n'est pas structurellement plus petit que
$$\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}.$$
Toutefois, en reprenant la définition de $\Delta[C, f, g]$, il est facile de la reformuler de manière correcte. Toute cette partie étant, à mon sens, suffisamment ardue, pour que l'on s'autorise une formulation un peu plus intuitive et, au premier abord, un peu moins rigoureuse.

Remarque Si pour tout i, $|F_i|_{\mathcal{I}} = |G_i|_{\mathcal{I}}$, alors
$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} = |\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{G_i}\}|_{\mathcal{I}}.$$

Remarque Si $|t|_{\mathcal{I}} =_{\beta\eta\iota} |t'|_{\mathcal{I}}$ alors
$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} = |\mathsf{Elim}(I, Q, \vec{u}, t')\{\overrightarrow{F_i}\}|_{\mathcal{I}}.$$

Remarque Si $t_1 =_{\beta\eta\iota} t_2$, alors $|\mathsf{Elim}(I, Q, \vec{u}, t_1)\{\overrightarrow{T_i}\}|_{\mathcal{I}} = |\mathsf{Elim}(I, Q, \vec{u}, t_2)\{\overrightarrow{T_i}\}|_{\mathcal{I}}$.

Lemme 4.20 *Soit T bien construit d'ordre K sous une interprétation $\mathcal{I}$, a une variable de preuve et t un terme de preuve. On a :*
$$|T[a \setminus t]|_{\mathcal{I}} = |T|_{\mathcal{I};a \leftarrow |t|_{\mathcal{I}}}.$$

[5] Remarquons que c'est ici que nous utilisons la propriété de confluence sur les termes purs, ou, si l'on veut, l'unicité de la forme normale dans Λ.

Lemme 4.21 *Soit T bien construit d'ordre K sous une interprétation $\mathcal{I}$, α une variable de preuve et T_1 un prédicat, tels que T_1 est bien construit sous $\mathcal{I}$ du même ordre que $\mathcal{I}(\alpha)$. On a :*

$$|T[\alpha \setminus T_1]|_{\mathcal{I}} = |T|_{\mathcal{I};\alpha \leftarrow |T_1|_{\mathcal{I}}}.$$

PREUVE Par récurrence sur la structure de T. ■

Les propriétés suivantes nous seront utiles par la suite pour traiter l'élimination forte :

Lemme 4.22 *Soit le prédicat $\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}$ bien construit d'ordre $(Q\ \vec{u}\ t)$ une interprétation $\mathcal{I}$ avec*

$$I \equiv \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}.$$

On appelle G la fonction utilisée dans la définition 4.30 pour l'interprétation de l'élimination forte ; c'est-à-dire que l'on a :

$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} = G(|I|_{\mathcal{I}})(|\vec{u}|_{\mathcal{I}}, |t|_{\mathcal{I}}).$$

Si $|t|_{\mathcal{I}} \in |(I\ \vec{u})|_{\mathcal{I}}$, alors pour tout $H \in \rho(A)$ avec $|t|_{\mathcal{I}} \in H(|\vec{u}|_{\mathcal{I}})$, on a :

$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} = G(H)(|\vec{u}|_{\mathcal{I}}, |t|_{\mathcal{I}})$$

et donc aussi

$$G(|I|_{\mathcal{I}})(|\vec{u}|_{\mathcal{I}}, |t|_{\mathcal{I}}) = G(H)(|\vec{u}|_{\mathcal{I}}, |t|_{\mathcal{I}}).$$

La preuve est immédiate.

Lemme 4.23 *Soient $\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}$, $\mathcal{I}$ et G tels que ci-dessus. A curryfication près, on a :*

$$|\mathsf{Fun\_Elim}(I, Q, \vec{F_i})|_{\mathcal{I}} = G(|I|_{\mathcal{I}}).$$

PREUVE C'est une conséquence immédiate de la définition de l'interprétation de l'abstraction et de l'application. ■

4.5 Interprétations invariantes

Une première propriété est que l'interprétation des prédicats ne dépend que des classes d'équivalence des $\mathcal{I}(a)$ modulo $=_{\beta\eta\iota}$.

Lemme 4.24 *Soit $(\mathcal{C}_i)_{i\in I}$ une famille d'éléments de $\rho(K)$, pour un certain ordre K et un ensemble quelconque I. Si pour tout i dans I, C_i est invariant, alors $\bigwedge_{i\in I} C_i$ est aussi invariant.*

La relation d'équivalence partielle $\simeq$ et la notion d'invariance s'étendent naturellement aux interprétations :

Définition 4.31 (Interprétations équivalentes) *Soient $\mathcal{I}$ et $\mathcal{I}'$ deux interprétations engendrant la même ordre-interprétation $\mathcal{J}$. On dit que $\mathcal{I}$ et $\mathcal{I}'$ sont équivalentes si pour tout α, $\mathcal{I}(\alpha) \simeq \mathcal{I}'(\alpha)$ et pour tout a $\mathcal{I}(a) =_{\beta\eta\iota} \mathcal{I}'(a)$. On note $\mathcal{I} \simeq \mathcal{I}'$.*

Définition 4.32 (Interprétations invariantes) *Une interprétation $\mathcal{I}$ est invariante si elle est équivalente à elle même : $\mathcal{I} \simeq \mathcal{I}$. Ou, de manière équivalente, si pour tout α, $\mathcal{I}(\alpha) \in \rho(K)$ est invariant dans $\rho(K)$. En particulier toute interprétation candidat est invariante.*

Lemme 4.25 *Soit K un ordre, T un prédicat bien construit d'ordre K sous deux interprétations équivalentes $\mathcal{I}$ et $\mathcal{I}'$. Alors $|T|_{\mathcal{I}} \simeq_K |T|_{\mathcal{I}'}$. En particulier $|T|_{\mathcal{I}}$ est invariant dans $\rho(K)$.*

PREUVE La preuve se fait par récurrence structurelle sur T.

- Le cas $T = \alpha$ est immédiat.
- Si $T = (a : T_1)T_2$, on a $|T_1|_{\mathcal{I}} = |T_1|_{\mathcal{I}'}$ et pour tout terme pur u, $|T_2|_{\mathcal{I};a\leftarrow u} = |T_2|_{\mathcal{I}';a\leftarrow u}$; donc on a bien $|T|_{\mathcal{I}} = |T|_{\mathcal{I}'}$.
- Si $T = (\alpha : K_1)T_1$, on sait que $K = \text{Set}$ et $|T|_{\mathcal{I}} = |T|_{\mathcal{I}'}$. Par ailleurs pour tout $\mathcal{C}$ invariant d'ordre K_1, $\mathcal{I};\alpha \leftarrow \mathcal{C}$ et $\mathcal{I}';\alpha \leftarrow \mathcal{C}$ sont deux interprétations équivalentes et donc $|T_1|_{\mathcal{I};\alpha\leftarrow\mathcal{C}} = |T_1|_{\mathcal{I}';\alpha\leftarrow\mathcal{C}}$. Le résultat suit.
- Si $T = (T_1\ t)$ on vérifie d'abord aisément $|t|_{\mathcal{I}} =_{\beta\eta\iota} |t|_{\mathcal{I}'}$. Le résultat est alors immédiat, puisqu'on a $|T_1|_{\mathcal{I}} \simeq |T'|_{\mathcal{I}'}$.
- Le cas $T = (T_1\ T_2)$ est similaire au précédent.
- Si $T = [a : T_1]T_2$, quels que soit les termes purs u et u', si $u =_{\beta\eta\iota} u'$, alors $\mathcal{I}; a \leftarrow u \simeq \mathcal{I}'; a \leftarrow u'$, et donc alors $|T_2|_{\mathcal{I};a\leftarrow u} \simeq |T_2|_{\mathcal{I};a\leftarrow u'}$.
- Le cas $T = [\alpha : K_1]T_1$ est similaire au précédent.
- Si $T = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$, avec $A \equiv (\vec{x}_j : \vec{A}_j)\text{Set}$, on commence par remarquer qu'en reprenant les raisonnements ci-dessus, on vérifie que pour tout i et pour tout candidat de réductibilité $\mathcal{C} \in \mathcal{CR}((\vec{x}_j : \vec{A}_j)\alpha \to \text{Set})$ on a :

$$|\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow|T|_{\mathcal{I}};Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)} = |\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow|T|'_{\mathcal{I}};Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

  Comme par ailleurs, si $\mathcal{A}$ et $\mathcal{A}'$, éléments de $\text{dom}(A)$ sont équivalents on a aussi $\mathcal{C}(\mathcal{A})(u) = \mathcal{C}(\mathcal{A}')(u)$ pour tout terme pur u, on voit bien que $|T|_{\mathcal{I}}(\mathcal{A}) = |T|_{\mathcal{I}'}(\mathcal{A}')$, et donc $|T|_{\mathcal{I}} \simeq |T|_{\mathcal{I}'}$.
- Si $T = \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}$, la définition de $|T|_{\mathcal{I}}$ fait intervenir une application G dépendant de $\mathcal{I}$. Ici on distinguera entre :

$$|T|_{\mathcal{I}} = G(|I|_{\mathcal{I}})(|t|_{\mathcal{I}}, |\vec{u}|_{\mathcal{I}}) \text{ et } |T|_{\mathcal{I}'} = G'(|I|_{\mathcal{I}'})(|t|_{\mathcal{I}'}, |\vec{u}|_{\mathcal{I}'}).$$

  L'hypothèse de récurrence nous assure entre autres que

$$|\vec{u}|_{\mathcal{I}} \simeq_{\text{dom}(A)} |\vec{u}|_{\mathcal{I}'} \text{ et } |I|_{\mathcal{I}} \simeq_A |I|_{\mathcal{I}'}$$

  et on a aussi $|t|_{\mathcal{I}} =_{\beta\eta\iota} |t|_{\mathcal{I}'}$.
  Les applications G et G' étant définies comme plus petit point fixe, on a $G = F^{\mathfrak{a}}$ et $G' = F'^{\mathfrak{a}}$ pour certaines fonctionnelles F et F' et un certain ordinal $\mathfrak{a}$. On montre aisément que $F^{\mathfrak{a}} \simeq F^{\mathfrak{a}}$ par induction transfinie sur $\mathfrak{a}$. Il suffit de reprendre la structure de la démonstration du lemme 4.16.

■

4.6 Interprétations Candidats

Lemme 4.26 *Soit T un prédicat de la forme $(\vec{x}_i : \vec{T}_i)T'$, tel que T soit bien construit d'ordre* Set *sous une interprétation $\mathcal{I}$. Un terme pur u est élément de $|T|_{\mathcal{I}}$ si et seulement si pour toute séquence $\vec{\mathcal{A}}_i \in |(\vec{x}_i : \vec{T}_i)|_{\mathcal{I}}$ on a $(u\ \vec{B}) \in |T|_{\mathcal{I};x_i\leftarrow\mathcal{A}_i}$, où $\vec{B}$ est la sous-séquence de $\vec{\mathcal{A}}_i$ ne contenant que les termes purs.*

PREUVE Immédiate par récurrence sur la longueur de $\vec{x}_i$ et $\vec{T}_i$. ■

Lemme 4.27 *Soit $\mathcal{I}$ une interprétation candidat et T un prédicat bien construit d'ordre K sous $\mathcal{I}$. Alors $|T|_{\mathcal{I}}$ est élément de $\mathcal{CR}(K)$.*

PREUVE Comme on a déjà vu que $|T|_{\mathcal{I}}$ était invariant, il ne reste qu'à vérifier que :
- Si $K = \text{Set}$ alors $|T|_{\mathcal{I}}$ vérifie les conditions **CR 1**, **CR 2** et **CR 3**.
- Si $K = (\alpha : K_1)K_2$ alors pour tout $\mathcal{C} \in \mathcal{CR}(K_1)$, $|T|_{\mathcal{I}}(\mathcal{C}) \in \mathcal{CR}(K_2)$
- Si $K = (a : T)K_1$ alors pour tout terme pur u, $|T|_{\mathcal{I}}(u) \in \mathcal{CR}(K_1)$.

On procède une nouvelle fois par récurrence structurelle sur T (ou, si l'on veut, par récurrence sur la preuve que T est bien construit) :
- Si $T = \alpha$, c'est immédiat puisque $\mathcal{I}$ est une interprétation candidat.
- Si $T = (x : T_1)T_2$, on sait que $K = \text{Set}$, et que T_1 est T_2 sont aussi bien construits d'ordre Set. Donc $|T_1|_{\mathcal{I}}$ est un candidat d'ordre Set, de même que $|T_2|_{\mathcal{I};x\leftarrow u}$ pour tout terme u. Alors $|T|_{\mathcal{I}}$ est bien élément de $\mathcal{CR}(\text{Set})$, d'après le lemme 4.6.
- Si $T = (\alpha : K_1)T_1$, on sait que T_1 est bien construit d'ordre Set sous $\mathcal{J}; \alpha \leftarrow S$ pour tout S dans $\rho(K_1)$. Donc si $S \in \mathcal{CR}(K_1)$, alors $T_1|_{\mathcal{I};\alpha\leftarrow S} \in \mathcal{CR}(\text{Set})$. Alors, d'après le lemme 4.5, $|T|_{\mathcal{I}} = \bigcap_{\mathcal{C}\in\mathcal{CR}(K_1)} |T_1|_{\mathcal{I};\alpha\leftarrow S}$ est bien un candidat d'ordre Set.
- Si $T = [a : T_1]T_2$, alors K est de la forme $(a : T_1)K_2$ et T_2 est bien construit d'ordre K_2 sous $\mathcal{I}; a \leftarrow u$ pour tout terme pur u. L'hypothèse de récurrence nous assure alors que $|T_2|_{\mathcal{I};a\leftarrow u} \in \mathcal{CR}(K_2)$, ce qui est équivalent au résultat recherché.
- Le cas $T = [\alpha : K_1]T_2$ est similaire.
- On traite le cas $T = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$ en utilisant le lemme 4.16. Soit F l'application de $\rho(A)$ vers $\rho(A)$ dont $|T|_{\mathcal{I}}$ est le point fixe. Notons $A \equiv (\vec{x}_j : \vec{A}_j)\text{Set}$. On vérifie que $F(\emptyset(A)) \in \mathcal{CR}(A)$ et $F(S) \in \mathcal{CR}(A)$ si $S \in \mathcal{CR}(A)$:
  - $F(\emptyset(A))$. Prenons le cas où au moins l'un des arguments de l'un des constructeurs est récursif ; appellons i_0 son indice. Soient aussi $\mathcal{C}$ un candidat de réductibilité du bon ordre et $\mathcal{A} \in \text{dom\_C}(A)$. On vérifie alors aisément que $|\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow\emptyset(A);Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}$ contient tous les termes purs, y compris ceux qui ne sont pas normalisables. Aussi, si l'on choisi f_{i_0} non normalisable, n'existe-t-il pas de terme pur u tel que

$$(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X), f_i, [\vec{x}_j : \vec{A}_j][c : (I\ \vec{x}_j)](a\ c)]\!]}\}\ u) \in \mathcal{C}(\mathcal{A})(u)$$

    puisque $\mathcal{C}(\mathcal{A})(u)$ est un candidat de réductibilité et le terme de gauche n'est pas fortement normalisable. On a donc $F(\emptyset(A)) = \min(A) \in \mathcal{CR}(A)$.
    Si aucun constructeur n'est récursif, on voit aisément que F est en fait une fonction constante. On peut alors, par exemple, reprendre la preuve du cas suivant.
  - Soit $S \in \mathcal{CR}(A)$. On prouve $F(S) \in \mathcal{CR}(A)$ en vérifiant que pour tout $\mathcal{A} \in \text{dom\_C}(A)$, $F(S)(\mathcal{A})$ vérifie bien les trois conditions **CR 1**, **CR 2** et **CR 3**.
    - $|I|_{\mathcal{I}}(\mathcal{A}) \subset \boldsymbol{\mathcal{SN}}$. Il suffit de vérifier que pour tout i,

$$|\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow c(i)}$$

      est non vide. C'est immédiat puisqu'en utilisant l'hypothèse de récurrence on voit aisément que c'est un candidat de réductibilité.

– Le fait que $|I|_{\mathcal{I}}(\mathcal{A})$ est clos par réduction se vérifie aisément : soit $u \in |I|_{\mathcal{I}}(\mathcal{A})$ et u' avec $u \mathbf{B} u'$. alors soit $u \in \min(A)$, et alors $u' \in \min(A)(\mathcal{A})$ et donc aussi $u' \in |I|_{\mathcal{I}}(\mathcal{A})$, soit pour tout candidat $\mathcal{C}$, toute séquence $\vec{f_i}$ avec

$$f_i \in |\Delta\{C_i(X), Q, c\}|_{\mathcal{I};X \leftarrow |I|_{\mathcal{I}};Q \leftarrow \mathcal{C};c \leftarrow \mathsf{Co}(i)}$$

on a

$$(\mathsf{match}\ a.\{[\![C_i(X), f_i, \overrightarrow{[\vec{x}_j : \vec{A}_j][c : (I\ \vec{x}_j)](a\ c)]\!]}\}\ u) \in \mathcal{C}(\mathcal{A})(u)$$

mais alors comme $\mathcal{C}(\mathcal{A})(u)$ est un candidat de réductibilité, et est égal à $\mathcal{C}(\mathcal{A})(u')$, on a bien :

$$(\mathsf{match}\ a.\{[\![C_i(X), f_i, \overrightarrow{[\vec{x}_j : \vec{A}_j][c : (I\ \vec{x}_j)](a\ c)]\!]}\}\ u') \in \mathcal{C}(\mathcal{A})(u').$$

– Il nous faut encore vérifier que $F(S)(\mathcal{A})$ vérifie **CR 3**. Soit u un terme pur neutre, tel que si $u \mathbf{B} u'$ alors $u \in F(S)(\mathcal{A})$. Soient alors un candidat de réductibilité $\mathcal{C}$ et une séquence de termes $\vec{f_i}$ avec

$$f_i \in |\Delta\{C_i(X), Q, c\}|_{\mathcal{I};x \leftarrow S;Q \leftarrow \mathcal{C};c \leftarrow \mathsf{Co}(i)}$$

pour vérifier $u \in F(S)(\mathcal{A})$, il suffit que

$$(\mathsf{match}\ a.\{[\![C_i(X), f_i, \overrightarrow{[\vec{x}_j : \vec{A}_j][c : (I\ \vec{x}_j)](a\ c)]\!]}\}\ u) \in \mathcal{C}(\mathcal{A})(u)$$

et comme le terme est neutre et que $\mathcal{C}(\mathcal{A})(u)$ est un candidat, il suffit alors de montrer que tous les réduits du terme sont dans $\mathcal{C}(\mathcal{A})(u)$. On procède alors par la méthode habituelle, par récurrence sur le nombre maximal de réductions possibles sur les f_i, en remarquant d'une part que u étant neutre le terme n'est pas un radical, et d'autre part, que les $\Delta\{C_i(X), Q, c\}|_{\mathcal{I};x \leftarrow S;Q \leftarrow \mathcal{C};c \leftarrow \mathsf{Co}(i)}$ sont des candidats de réductibilité et donc clos par réduction.

– Si $T = \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}$, alors

$$|T|_{\mathcal{I}} \equiv G(|I|_{\mathcal{I}})(|\vec{u}|_{\mathcal{I}}, |t|_{\mathcal{I}})$$

où G est une application définie par récursion bien fondée sur son premier élément. On prouve alors par récurrence bien-fondée, que pour tout $H \in \mathrm{dom}(F)$, pour tout $\mathcal{A} \in \mathrm{dom}(A)$, tout terme pur u, on a bien $G(H)(\mathcal{A}, u) \in \mathcal{CR}(A)$. Les cas non-récursifs sont immédiats. Le cas récursif est une simple application de l'hypothèse de récurrence, puisqu'en reprenant les raisonnements faits pour les cas précédents on vérifie aisément que

$$|\Delta[C_i(I), F_i, g]|_{\mathcal{I};g \leftarrow G(pred(\mathcal{B},w))}$$

est un candidat de réductibilité.

■

Définition 4.33 (Interprétations adaptées) *Soit Γ un contexte bien-formé, et $\mathcal{I}$ une interprétation candidat admissible pour Γ. On dit que $\mathcal{I}$ est adaptée à Γ, si :*

$$\forall a \in \Gamma.\mathcal{I}(a) \in |\Gamma(a)|_{\mathcal{I}}.$$

Définition 4.34 (Interprétation canonique) *Soit Γ un contexte bien-formé. On définit l'interprétation $\boldsymbol{\mathcal{C}}(\Gamma)$, qu'on appelle interprétation canonique de Γ par les clauses suivantes :*

- *pour toute variable de preuve* a, $\mathcal{C}(\Gamma)(a) \equiv a$
- *pour toute variable de prédicat* α *apparaissant dans* Γ, $\mathcal{C}(\Gamma)(\alpha) \equiv Can(\Gamma(\alpha))$
- *pour toute variable de prédicat* α *n'apparaissant pas dans* Γ, $\mathcal{C}(\Gamma)(\alpha) \equiv Can(\mathrm{Set})$.

On vérifie aisément que $\mathcal{C}(\Gamma)$ est une interprétation adaptée à Γ.

Lemme 4.28 (Correction de l'interprétation de l'abstraction) *Soit un jugement* $\Gamma \vdash [a : T_0]t : (a : T_0)T$ *et* $\mathcal{I}$ *une interprétation candidat adaptée à* Γ. *On a* $|[a : T_0]t|_{\mathcal{I}} \in |(x : T_0)T|_{\mathcal{I}}$ *si et seulement si pour tout terme pur* $u \in |T_0|_{\mathcal{I}}$ *on a* $|t|_{\mathcal{I};a\leftarrow u} \in |T|_{\mathcal{I};a\leftarrow u}$.

Preuve La condition nécessaire est une conséquence immédiate de la définition de $|(a : T_0)T|_{\mathcal{I}}$, du lemme 4.20 et de la condition **CR 2**. La preuve de la condition suffisante est une conséquence du lemme 4.7 (pour tout $u \in |T_0|_{\mathcal{I}}$, $|[a : T_0]t|_{\mathcal{I}} \in \{u\} \rightarrow |T|_{\mathcal{I};a\leftarrow u}$). ■

Lemme 4.29 (Correction de l'interprétation de l'abstraction polymorphe) *Soit un jugement* $\Gamma \vdash [\alpha : K]t : (\alpha : K)T$ *et une interprétation candidat* $\mathcal{I}$ *adaptée à* Γ. *On a :* $|[\alpha : K]T|_{\mathcal{I}} \in |(\alpha : K)T|_{\mathcal{I}}$ *Si et seulement si pour tout candidat de réductibilité* $\mathcal{C} \in \mathcal{CR}(K)$ *on a :*

$$|t|_{\mathcal{I}} = |t|_{\mathcal{I};\alpha\leftarrow\mathcal{C}} \in |T|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}.$$

Preuve C'est la simple combinaison des définitions des interprétations du terme preuve et du prédicat. ■

Définition 4.35 *Soient* $\vec{x}_i$ *une séquence de variables et* $\vec{u}_i$ *une séquence de preuves et de prédicats, de même longueur, et telles que pour tout* i, x_i *est une variable de preuve (respectivement de prédicat) si et seulement si* U_i *est une preuve (respectivement un prédicat). Soit aussi une interprétation* $\mathcal{I}$. *On définit alors* $|(\vec{x}_i : \vec{U}_i)|_{\mathcal{I}}$, *une partie de* $dom((\vec{x}_i : \vec{U}_i)\mathrm{Set})$ *par les clauses suivantes :*

- $|([] : [])|_{\mathcal{I}} = dom(\mathrm{Set})$
- $(u, \mathcal{A}) \in |(a :: \vec{x} : T :: \vec{U})|_{\mathcal{I}}$ *si et seulement si* T *est bien construit d'ordre* Set *sous* $\mathcal{I}$, $u \in |T|_{\mathcal{I}}$, $|(\vec{x} : \vec{U})|_{\mathcal{I};a\leftarrow u}$ *est défini et* $\mathcal{A} \in |(\vec{x} : \vec{U})|_{\mathcal{I};a\leftarrow u}$.
- $(\mathcal{C}, \mathcal{A}) \in |(\alpha :: \vec{x} : K :: \vec{U})|_{\mathcal{I}}$ *si* $\mathcal{C} \in \mathcal{CR}(K)$, *si* $|(\vec{x} : \vec{U})|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$ *est défini, et* $\mathcal{A} \in |(\vec{x} : \vec{U})|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$.

Remarque Pour tout jugement dérivable $\Gamma \vdash (\vec{x} : \vec{U})M : N$, pour toute interprétation $\mathcal{I}$ admissible pour Γ, $|(\vec{x} : \vec{U})|_{\mathcal{I}}$ est défini.

La précédente définition permet de donner la caractérisation suivante :

Lemme 4.30 (Dé-curryfication de l'interprétation du produit) *Soient* $(\vec{x} : \vec{U})T$ *un prédicat bien construit d'ordre* Set *sous* $\mathcal{I}$ *et* u *un terme pur. On a* $u \in |(\vec{x} : \vec{U})T|_{\mathcal{I}}$ *si et seulement si pour tout* $\mathcal{A} \in |(\vec{x} : \vec{U})|_{\mathcal{I}}$, *on a* $(u\ \overline{\mathcal{A}}) \in |T|_{\mathcal{I};\vec{x}\leftarrow\mathcal{A}}$, *où* $\overline{A}$ *est la sous-séquence de* $\mathcal{A}$ *ne contenant que les termes.*

Preuve C'est bien sûr une récurrence immédiate sur la longueur des séquences $\vec{x}$ et $\vec{U}$. ■

On peut également faire maintenant la remarque suivante sur les types inductifs :

Lemme 4.31 *Soient deux prédicats I et I', respectivement de la forme :*

$$\begin{aligned} I &\equiv \mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\} \\ I' &\equiv \mathsf{Ind}(X:A')\{\overrightarrow{C'_i(X)}\} \end{aligned}$$

où :

$$\begin{aligned} A &\equiv (\vec{y_j}:\vec{A_j})\mathrm{Set} \\ A' &\equiv (\vec{y'}_j:\vec{A_j})\mathrm{Set} \end{aligned}$$

avec $\rho(A)=\rho(A')$ et tel que I et I' soient tous deux bien construits d'ordre A (et donc aussi d'ordre A') sous une interprétation candidat $\mathcal{I}$. On suppose de plus que pour tout candidat de réductibilité S d'ordre A, tout candidat de réductibilité $\mathcal{C}$ d'ordre $(\vec{y_j}:\vec{A_j})(I\ \vec{y}) \to \mathrm{Set}$ et tout indice i, et tout terme pur u on ait :

$$|\Delta\{C_i(X),Q,c\}|_{\mathcal{I};X\to S;Q\to\mathcal{C};c\to u} = |\Delta\{C'_i(X),Q,c\}|_{\mathcal{I};X\to S;Q\to\mathcal{C};c\to u}.$$

Sous ces conditions, on a :

- $|I|_{\mathcal{I}} = |I'|_{\mathcal{I}}$
- *Les ordres partiels $<_{(I,\mathcal{I})}$ et $<_{(I',\mathcal{I})}$ définis dans 4.30 sur $\rho(A)$ sont identiques.*

4.7 Interprétation des constructeurs

Dans cette partie, on cherche essentiellement à s'assurer de la correction de l'interprétation des types inductifs ; à savoir :

- les interprétations de l'élimination faible et des constructeurs font bien partie des interprétations de leurs types respectifs, modulo les conditions adéquates.
- l'ordre bien-fondé défini sur $|I|_{\mathcal{I}}$ correspond bien à la structure des termes.

Dans tout ce paragraphe on considère un prédicat

$$I \equiv \mathsf{Ind}(X:A)\{\overrightarrow{C_i(X)}\}$$

bien construit d'ordre $A \equiv (\vec{x_j}:\vec{U_j})\mathrm{Set}$ sous une interprétation candidat $\mathcal{I}$, un candidat de réductibilité

$$\mathcal{C} \in \mathcal{CR}((\vec{x_j}:\vec{U_j})(I\ \vec{y_j}) \to \mathrm{Set})$$

et une séquence de termes purs $\vec{f_i}$ tels que

$$\forall i. f_i \in |\Delta\{C_i(I),Q,c\}|_{\mathcal{I};Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

Lemme 4.32 *Soit $\mathcal{A} \in dom(A)$ et un terme pur $u \in |I|_{\mathcal{I}}(\mathcal{A})$. On a alors*

$$(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X), f_i, [\vec{x_j}:\vec{U_j}][c:(I\ \vec{x_j})](a\ c)]\!]}\}\ u) \in \mathcal{C}(\mathcal{A})(u).$$

PREUVE C'est la conséquence immédiate de la définition de $|I|_{\mathcal{I}}$. ■

Une première conséquence est que l'interprétation de l'élimination faible est correcte :

Corollaire 4.7 (Correction de l'interprétation de l'élimination) *Soit un jugement dérivable* $\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{g_i}\} : (Q\ \vec{u}\ t)$ *où* Q *est un* prédicat. *Si les conditions suivantes sont vérifiées :*

$$|t|_{\mathcal{I}} \in |I|_{\mathcal{I}}(|\vec{u}|_{\mathcal{I}})$$

$$\forall i.|g_i|_{\mathcal{I}} \in |\Delta\{C_i(I), Q, \mathsf{Constr}(i, I)\}|_{\mathcal{I}}$$

on a alors :

$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{g_i}\}|_{\mathcal{I}} \in |(Q\ \vec{u}\ t)|_{\mathcal{I}}.$$

En utilisant le lemme 4.30 et le corollaire 4.4, on prouve alors aisément :

Corollaire 4.8 *Soit* $\vec{x}_{j_k}$ *la sous-séquence de* $\vec{x}_j$ *ne contenant que des variables de preuves. Le terme*

$$\lambda\vec{x}_{j_k}.\lambda c.(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X), f_i, \lambda\vec{x}_{j_k}.\lambda c.(a\ c)]\!]}\}\ c)$$

est élément de

$$|(Q : (\vec{x}_j : \vec{U}_j)(I\ \vec{x}_j) \to \mathrm{Set})(\vec{x}_j : \vec{U}_j)(c : (I\ \vec{x}_j))(Q\ \vec{x_j}\ c)|_{\mathcal{I}}$$

ou, de manière équivalente :

$$\lambda\vec{x}_{j_k}.\lambda c.(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X), f_i, \lambda\vec{x}_{j_k}.\lambda c.(a\ c)]\!]}\}\ c) \in |(\vec{x}_j : \vec{U}_j)(c : (I\ \vec{x}_j))(Q\ \vec{x_j}\ c)|_{\mathcal{I};Q\leftarrow\mathcal{C}}$$

c'est-à-dire aussi avec les conditions précédentes sur $\vec{g}_i$, *et* t *:*

$$|\mathsf{Fun\_Elim}(I, Q, \vec{g}_i)|_{\mathcal{I}} \in |(Q : (\vec{x}_j : \vec{U}_j)(I\ \vec{x}_j) \to \mathrm{Set})(\vec{x}_j : \vec{U}_j)(c : (I\ \vec{x}_j))(Q\ \vec{x_j}\ c)|_{\mathcal{I}}.$$

Lemme 4.33 *On détaille les types* $C_i(X)$ *ainsi :*

$$C_i(X) \equiv (\vec{y}_k i : \vec{U}^i_k(X))(X\ \vec{t_j}^i).$$

Soit un terme pur $F \in |(\vec{x}_j : \vec{U}_j)(c : (I\ \vec{x}_j))(Q\ \vec{x_j}\ c)|_{\mathcal{I};Q\leftarrow\mathcal{C}}$, *alors on a :*

$$[\![C_i(I), f_i, F]\!] \in |(\vec{y^i_k} : U^i_k(I))(Q\ \vec{t^i_j}\ (\mathsf{Constr}(i, I)\ \vec{y^i_k})|_{\mathcal{I};Q\leftarrow\mathcal{C}}.$$

Preuve Par récurrence sur la structure de $C_i(X)$. ∎

Lemme 4.34 (Correction de l'interprétation des constructeurs) *Pour tout* i, $\mathsf{Co}(i) \in |C_i(I)|_{\mathcal{I}}$.

Preuve Rappellons que $C_i(X) \equiv (\vec{y^i}_k : \vec{U^i}_k)(X\ \vec{t^i_j})$. Soit $\mathcal{A} \equiv \vec{\mathcal{A}}_k \in |(\vec{y^i}_k : \vec{U^i}_k)|_{\mathcal{I};X\leftarrow|I|_{\mathcal{I}}}$. On appelle $\overline{\mathcal{A}}$ la sous-séquence de $\mathcal{A}$ ne contenant que des termes purs. Il nous faut alors simplement vérifier que $(\mathsf{Co}(i)\ \overline{\mathcal{A}}) \in |(I\ \vec{t^i_j})|_{\mathcal{I};\forall k.y^i_k\leftarrow\mathcal{A}_k}$. Pour cela il suffit que

$$(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X), f_i, \overrightarrow{[\vec{x_j} : \vec{U_j}][c : (I\ \vec{x}_j)](a\ c)]}]\!]}\}\ (\mathsf{Co}(i)\ \overline{\mathcal{A}})) \in \mathcal{C}(|\vec{t^i_j}|_{\mathcal{I};\forall k.y^i_k\leftarrow\mathcal{A}_k})(\mathsf{Co}(i)\ \overline{\mathcal{A}})$$

c'est-à-dire aussi :

$$(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(X), f_i, \overrightarrow{[\vec{x_j} : \vec{U_j}][c : (I\ \vec{x}_j)](a\ c)]}]\!]}\}\ (\mathsf{Co}(i)\ \overline{\mathcal{A}})) \in |(Q\ \vec{t^i_j}\ (\mathsf{Constr}(i, I)\ \vec{x^i_k}))|_{\mathcal{I};Q\leftarrow\mathcal{C}}$$

Comme $\mathcal{I}$ est une interprétation candidat, on sait que les $[\![C_i(X), f_i, \overline{[\vec{x_j} : \vec{U_j}][c : (I\ \vec{x_j})](a\ c)}]\!]$ sont tous fortement normalisables puisqu'ils sont éléments respectivement des candidats de réductibilité $|(\vec{y_k^i} : \vec{U_k^i}(I))(Q\ \vec{t_j^i}\ (\mathsf{Constr}(i, I)\ \vec{y_k^i})|_{\mathcal{I}}$.

Grace au lemme 4.8, la condition 1 se ramène alors à

$$(\Delta[C_i(X), f_i, \lambda\vec{x}_{j_k}.\lambda c.(\mathsf{match}\ a.\{\overrightarrow{[\![C_i(I), f_i, \lambda\vec{x}_{j_k}.\lambda c.(a\ c)]\!]}\}\ c)]\ \overline{\mathcal{A}}) \in \mathcal{C}(|\vec{t_j^i}|_{\mathcal{I};\forall j.x_j \to \mathcal{A}_j})(\mathsf{Co}(i)\ \overline{\mathcal{A}})$$

ce qui est une conséquence du corollaire précédent. ■

Exemple (Entiers naturels) Reprenons le cas $I = \mathrm{Nat} \equiv \mathsf{Ind}(X : \mathrm{Set})\{X; X \to X\}$ avec $O \equiv \mathsf{Constr}(1, \mathrm{Nat})$ et $S \equiv \mathsf{Constr}(2, \mathrm{Nat})$.

Nous avons vérifié que les interprétations des constructeurs et des schémas d'élimination sont bien correctes vis-à-vis des règles d'introduction correspondantes. Il nous reste à montrer que l'ordre bien-fondé défini sur chaque interprétation d'un type inductif est bien compatible avec la structure des termes. En simplifiant : les termes qui apparaissent en position d'arguments (récursifs) d'un constructeur dans un terme t qui est élément de l'interprétation d'un type inductif, sont inférieurs à t pour le-dit ordre.

Lemme 4.35 *On suppose que les constructeurs de I sont tous petits (les U_k^i sont tous des prédicats). Soit $\mathcal{A} \in dom(A)$ et une séquence $\vec{m}_k$ telle que*

$$(\mathsf{Co}(i)\ \vec{m}_k) \in |I|_{\mathcal{I}}(\mathcal{A})$$

alors on a

$$\vec{m}_k \in |(\vec{y^i}_k : \vec{U^i}_k(X))|_{\mathcal{I};X \leftarrow pred((\mathsf{Co}(i)\ \vec{m}_k),\mathcal{A})}.$$

Preuve Étant donné un indice k_0, il nous faut vérifier que

$$m_{k_0} \in |U_{k_0}^i|_{\mathcal{I};\forall k<k_0.y_k^i \leftarrow m_k} \tag{1}$$

On distingue deux possibilités : $pred((\mathsf{Co}(i)\ \vec{m}_k), \mathcal{A})$ est vide ou non.

- Si $pred((\mathsf{Co}(i)\ \vec{m}_k), \mathcal{A})$ n'est pas vide, alors c'est un candidat de réductibilité d'ordre A, et donc $|U_{k_0}^i|_{\mathcal{I};\forall k<k_0.y_k^i \leftarrow m_k}$ est un candidat d'ordre Set. Soit $\mathcal{D}$, élément de

  $$\mathcal{CR}((\vec{x}_j : \vec{A}_j)(I\ \vec{x}_j) \to \mathrm{Set})$$

  défini comme l'application constante qui associe $|U_{k_0}^i|_{\mathcal{I};\forall k<k_0.y_k^i \leftarrow m_k}$ à tout élément de

  $$\mathrm{dom}((\vec{x}_j : \vec{A}_j)(I\ \vec{x}_j) \to \mathrm{Set}).$$

  Alors, pour tout indice i',

  $$|\Delta\{C_i'(X), Q, \mathsf{Constr}(i', I)\}|_{\mathcal{I};X \leftarrow pred((\mathsf{Co}(i)\ \vec{m}_k),\mathcal{A});Q \leftarrow \mathcal{D}}$$

  est un candidat de réductibilité d'ordre Set ; c'est en particulier vrai dans le cas $i' = i$. Pour verifier 1, il nous suffit de trouver les termes purs $\vec{f_i}$, tel que pour tout i' :

  $$f_{i'} \in |\Delta\{C_i'(X), Q, \mathsf{Constr}(i', I)\}|_{\mathcal{I};X \leftarrow pred((\mathsf{Co}(i)\ \vec{m}_k),\mathcal{A});Q \leftarrow \mathcal{D}}$$

et que le terme

$$(\Delta[C_i(X), f_i, F]\ \vec{m}_k)$$

se réduise vers m_{k_0} par réduction de tête.
Pour les indices $i' \neq i$, le plus facile est de choisir comme valeur de f'_i une variable quelconque b. Pour f_i, il suffit de prendre la projection qui retournera le k_0-ième argument du constructeur. Par exemple :

- si $C_i(X) = U \to V \to X$, avec U et V non-récursif, si $k_0 = 1$, on prendra $f_i = \lambda z_1.\lambda z_2.z_1$, et si $k_0 = 2$ alors $f_i = \lambda z_1.\lambda z_2.z_2$.
- si $C_i(X) = X \to X \to X$, si $k_0 = 1$, on prendra $f_i = \lambda z_1.\lambda z_2.\lambda z_3.\lambda z_4.z_1$, et si $k_0 = 2$ alors $f_i = \lambda z_1.\lambda z_2.\lambda z_3.\lambda z_4.z_3$.

On pourrait donner la définition formelle de f_i mais cela ne semble pas nécessaire. Il est à peu près évident qu'avec ces définitions $(\Delta[C_i(X), f_i, F]\ \vec{m}_k)$ se réduit bien vers m_{k_0} par réduction de tête.

- Si $pred((\mathsf{Co}(i)\ \vec{m}_k), \mathcal{A}) = \emptyset$, on vérifie très facilement que le type de constructeur $C_i(X)$ n'est pas récursif. On peut alors reprendre la preuve du cas précédent.

■

4.8 Interprétation et β-réduction

A cause de la présence de la règle de conversion, une propriété essentielle des interprétations est qu'elles doivent être invariantes par conversion ; si :

$$\Gamma \vdash T : \text{Set} \quad \Gamma \vdash T' : \text{Set} \quad T =_{\beta\eta\iota} T'$$

alors on veut avoir $|T|_{\mathcal{I}} = |T'|_{\mathcal{I}}$.

Dans les trois sous-parties qui suivent, nous examinons successivement les trois réductions β, ι et η, et leur comportement vis-à-vis de l'interprétation des prédicats. Nous verrons que seule la ι-réduction est un peu problématique et exige quelques précautions. Le cas de la β-réduction est en revanche sans malice.

Lemme 4.36 (Invariance par β-réduction) *Soit T un prédicat bien-construit d'ordre K sous une interprétation $\mathcal{I}$. Si $T\ \mathbf{B}_\beta\ T'$, alors T' est bien construit d'ordre K sous $\mathcal{I}$ et $|T|_{\mathcal{I}} = |T'|_{\mathcal{I}}$.*

PREUVE Par récurrence sur la structure de T. ■

Corollaire 4.9 *Soient le jugement dérivable $\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\} : (Q\ \vec{u}\ t)$ et $\mathcal{I}$ une interprétation adaptée à Γ. Alors*

$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} = |(\mathsf{Fun\_Elim}(I, Q, \vec{F_i})\ \vec{u}\ t)|_{\mathcal{I}}.$$

4.9 Interprétation et η-réduction

Le cas de la règle de η-réduction est juste un peu plus délicat que celui de la β-réduction. On cherche à vérifier que si $\|T_1\|$ et $\|T_2\|$ ont un η-réduit commun, alors leurs interprétations sont égales.

Lemme 4.37 (Invariance par η-réduction) *Soit T, un prédicat bien-construit d'ordre K sous une interprétation $\mathcal{I}$. Si $T\ \mathbf{B}_\eta\ T'$ alors T' est également bien construit d'ordre K sous $\mathcal{I}$, et $|T|_\mathcal{I} = |T'|_\mathcal{I}$.*

PREUVE Elle se fait bien sûr par récurrence sur la structure de T, ou si l'on veut, par récurrence sur la structure de la preuve de $T\ \mathbf{B}_\eta\ T'$. Le cas crucial est bien sûr $T = [x : U](T'\ x)$, avec $x \notin FV(T')$. On a alors $K = (x : U)K'$ et $(T'\ x)$ bien construit d'ordre K' sous $\mathcal{I}; x \leftarrow u$ pour u bien choisi. Alors T est bien construit d'ordre $(x : U)K'$ sous $\mathcal{I}$. Il est aussi évident que $|T|_\mathcal{I} = |T'|_\mathcal{I}$. ■

Lemme 4.38 *Pour tout prédicat bien-construit d'ordre K sous une interprétation $\mathcal{I}$ on a $|T|_\mathcal{I} = |||T|||_\mathcal{I}$.*

Lemme 4.39 (Invariance par η-conversion) *Soient deux prédicats T_1 et T_2 bien construit d'ordre K sous une interprétation $\mathcal{I}$. S'il existe un prédicat T_3 tel que $\|T_1\|\ \mathbf{B}^*_\eta\ T_3$ et $\|T_2\|\ \mathbf{B}^*_\eta\ T_3$, alors $|T_1|_\mathcal{I} = |T_2|_\mathcal{I}$.*

PREUVE D'après les deux lemmes précédents, on a $|T_1|_\mathcal{I} = |||T_1|||_\mathcal{I}$ et $|T_2|_\mathcal{I} = |||T_2|||_\mathcal{I}$ et $|||T_1|||_\mathcal{I} = |T_3|_\mathcal{I} = |||T_2|||_\mathcal{I}$. ■

4.10 Prédicats invariants sur leur domaine

Comme on l'a indiqué dans le chapitre 3, l'interprétation des prédicats peut être modifiée par ι-réduction. Elle reste en revanche constante si l'on considère uniquement la valeur de ceux-ci sur leur domaine "légal". On définit donc celui-ci précisément de manière similaire à la réductibilité.

Définition 4.36 (Invariant sur son domaine) *Soient un jugement dérivable $\Gamma \vdash T : K$ et une interprétation $\mathcal{I}$ adaptée à Γ. On définit par récurrence sur la mesure de la forme ordre-normale de K la proposition "$(T, K, \Gamma, \mathcal{I})$ est invariant sur son domaine" par :*

- *Si $K = \mathrm{Set}$ alors pour tout T' tel que $T\ \mathbf{B}^*\ T'$, $|T|_\mathcal{I} = |T'|_\mathcal{I}$.*
- *Si $K = (x : T_1)K_1$, alors pour tout jugement $\Gamma \vdash t : T_1$ dérivable, si $|t|_\mathcal{I}|T_1|_\mathcal{I}$ alors $((T\ t), K_1[x \setminus t], \Gamma, \mathcal{I})$ est invariant sur son domaine.*
- *Si $K = (\alpha : K_1)K_2$, alors pour tout jugement $\Gamma \vdash T_1 : K_1$ dérivable, on a $((T\ T_1), K_2[\alpha \setminus T_1], \Gamma, \mathcal{I})$ est invariant sur son domaine si $(T_1, K_1, \Gamma, \mathcal{I})$ est invariant sur son domaine.*

On vérifie que la condition d'invariance sur le domaine est suffisante pour traiter le cas de la règle de conversion dans le cas de jugements de la forme $\Gamma \vdash t : T$.

Lemme 4.40 *Soient deux jugements dérivables $\Gamma \vdash T_1 : \mathrm{Set}$ et $\Gamma \vdash T_2 : \mathrm{Set}$ et une interprétation $\mathcal{I}$ adaptée à Γ. Si $(T_1, \mathrm{Set}, \Gamma, \mathcal{I})$ et $(T_2, \mathrm{Set}, \Gamma, \mathcal{I})$ sont invariants sur leurs domaines et $T_1 =_{\beta\eta\iota} T_2$, alors $|T_1|_\mathcal{I} = |T_2|_\mathcal{I}$.*

PREUVE On sait que $\|T_1\|$ et $\|T_2\|$ admettent un réduit commun U :

$$\|T_1\|\ \mathbf{B}^*\ U \text{ et } \|T_2\|\ \mathbf{B}^*\ U$$

d'après le lemme de retard de η et le lemme 2.8 il existe deux prédicats T_1' et T_2' tels que :

$$T_1\ \mathbf{B}^*_{\beta\iota}\ T_1'\ \|T_1'\|\ \mathbf{B}^*_\eta\ U\ T_2\ \mathbf{B}^*_{\beta\iota}\ T_2'\ \|T_2'\|\ \mathbf{B}^*_\eta\ U.$$

Alors les hypothèses et les lemmes 4.38 et 4.39 assurent que

$$|T_1|_\mathcal{I} = |T_1'|_\mathcal{I} = |U|_\mathcal{I} = |T_2'|_\mathcal{I} = |T_2|_\mathcal{I}.$$

■

4.11 Interprétations et ι-réduction

Dans cette partie, on vérifie essentiellement que la règle de formation de l'élimination forte (ELIM-S) est correcte vis-à-vis de l'interprétation. Pour commencer, on montre que la propriété d'invariance sur son domaine est bien close par β puis ι-réduction de tête inverse :

Lemme 4.41 *Soient les termes T, A, V, U, $U_1 \dots U_n$. Si*

$$([x : A]T\ U\ U_1 \dots U_n)\ \mathbf{B}^*\ V$$

alors

$$(T[x \setminus U]\ U_1 \dots U_n)\ \mathbf{B}^*\ V'$$

avec soit $V = V'$, soit $V\ \mathbf{B}^1_\beta\ V'$.

PREUVE La preuve est facile, et similaire à ce qu'on peut faire en λ-calcul pur. On raisonne par récurrence sur la longueur de la chaîne de réductions, en vérifiant à chaque fois si le radical de tête est réduit ou non. ■

Lemme 4.42 *Si $(T[x \setminus A], K, \Gamma, \mathcal{I})$ est invariant sur son domaine, et si $\Gamma \vdash ([x : U]T\ A) : K$, alors $(([x : U]T\ A), K, \Gamma, \mathcal{I})$ est invariant sur son domaine.*

PREUVE Pour un terme T_0, tel que $\Gamma \vdash T_0 : K$, dire que $(T_0, K, \Gamma, \mathcal{I})$ est invariant, c'est équivalent à dire que pour une certaine classe de $U_1 \dots U_n$ qui dépend de K, si

$$(T_0\ U_1 \dots U_n)\ \mathbf{B}^*\ T_1$$

alors

$$|(T_0\ U_1 \dots U_n)|_{\mathcal{I}} = |T_1|_{\mathcal{I}}.$$

Considérons donc une telle séquence $U_1 \dots U_n$. Si $([x : U]T\ A\ U_1 \dots U_n)\ \mathbf{B}^*\ V$, alors d'après le lemme précédent, il existe V' tel que $(T[x \setminus A]\ U_1 \dots U_n)\ \mathbf{B}^*\ V'$, avec $V = V'$ ou $V\ \mathbf{B}_\beta\ V'$. Dans les deux cas, $|V|_{\mathcal{I}} = |V'|_{\mathcal{I}}$. Comme par ailleurs $(T[x \setminus A], K, \Gamma, \mathcal{I})$ est invariant sur son domaine, on a $|(T[x \setminus A]\ U_1 \dots U_n)|_{\mathcal{I}} = |V'|_{\mathcal{I}}$. Enfin comme $([x : U]T\ A)\ \mathbf{B}_\beta\ T[x \setminus A]$, on a $|(T[x \setminus A]\ U_1 \dots U_n)|_{\mathcal{I}} = |([x : U]T\ A\ U_1 \dots U_n)|_{\mathcal{I}}$. Donc $|([x : U]T\ A\ U_1 \dots U_n)|_{\mathcal{I}} = |V|_{\mathcal{I}}$. ■

Lemme 4.43 *Si $((\Delta[C_i(I), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_j})]\ \vec{m}), K, \Gamma, \mathcal{I})$ est invariant sur son domaine, et si*

$$\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, (\mathsf{Constr}(i, I)\ \vec{m}))\{\overrightarrow{F_j}\} : K$$

alors $(\mathsf{Elim}(I, Q, \vec{u}, (\mathsf{Constr}(i, I)\ \vec{m}))\{\overrightarrow{F_j}\}, K, \Gamma, \mathcal{I})$ est aussi invariant sur son domaine.

La preuve est similaire à celle du lemme précédent.

Corollaire 4.10 *Soit le jugement dérivable $\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\} : K$ (c'est-à-dire avec $K =_{\beta\eta\iota} (Q\ \vec{u}\ t)$) et $\mathcal{I}$ une interprétation adaptée à Γ. Alors $(\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}, K, \Gamma, \mathcal{I})$ est invariant sur son domaine si et seulement si $((\mathsf{Fun\_Elim}(I, Q, \vec{F_i})\ \vec{u}\ t), K, \Gamma, \mathcal{I})$ l'est.*

Il est utile d'introduire une variante β-réduite de $(\Delta[C_i(X), f, F]\ \vec{m})$:

Définition 4.37 *Soit deux termes f et F, une séquence de termes $\vec{m}$ et un type de constructeur $C(X) \equiv (\vec{x}_i : \vec{U}_i)(X\ \vec{t})$ tel que $\vec{m}$ soit de même longueur que les séquences $\vec{x}_i$ et $\vec{U}_i$. On définit alors le terme $\Delta_{red}[C_i(X), f, F, \vec{m}]$ par récurrence sur la longueur de $\vec{m}$ par les égalités suivantes :*

$$\begin{array}{rcl}
\Delta_{red}[(X\ \vec{t}), f, F, []] & \equiv & f \\
\Delta_{red}[(x : M)D(X), f, F, m :: \vec{m}] & \equiv & \Delta_{red}[D(X), (f\ m), F, \vec{m}] \\
\Delta_{red}[((\vec{x} : \vec{M})(X\ \vec{N})) \to D(X), f, F, m :: \vec{m}] & \equiv & \Delta_{red}[D(X), (f\ m\ [\vec{x} : \vec{M}](F\ \vec{M}\ (m\ \vec{x}))), F, \vec{m}]
\end{array}$$

On remarque tout de suite que $(\Delta[C_i(X), f, F]\ \vec{m})$ se réduit vers $\Delta_{red}[C_i(X), f, F, \vec{m}]$ par β-réduction de tête inverse. Plus précisément :

Lemme 4.44 *Soit le jugement dérivable $\Gamma \vdash (\Delta[C(X), f, F]\ \vec{m}) : K$ avec $C(X)$ et $\vec{m}$ vérifiant les conditions ci-dessus et $\mathcal{I}$ une interprétation adaptée à Γ. Alors $(\Delta[C(X), f, F]\ \vec{m}), K, \Gamma, \mathcal{I})$ est invariant sur son domaine si et seulement si $(\Delta_{red}[C(X), f, F, \vec{m}], K, \Gamma, \mathcal{I})$ l'est.*

PREUVE C'est une conséquence immédiate du lemme 4.42. ■

Les deux lemmes suivants correpondent au cas le plus délicat du théorème 10, à savoir que la règle de construction d'un prédicat par élimination forte préserve la propriété d'invariance sur le domaine.

Lemme 4.45 *Soit le jugement dérivable $\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\} : (Q\ \vec{u}\ t)$, où $\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}$, ainsi que les F_i sont des* prédicats. *Soit aussi $\mathcal{I}$ une interprétation adaptée à Γ. On suppose que $|t|_{\mathcal{I}} \in |(I\ \vec{u})|_{\mathcal{I}}$ et $t\ \mathbf{B}^*\ (\mathsf{Constr}(i, I)\ \vec{v})$. On a alors :*

$$|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} = |(\Delta[C_i(I), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i})]\ \vec{v})|_{\mathcal{I}}.$$

PREUVE Soit G la fonction utilisée en 4.30 pour la définition de $|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}}$. Soit $(\mathsf{Co}(i)\ \vec{w})$ la forme normale de $|t|_{\mathcal{I}}$. On a

$$\begin{array}{rcl}
|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} & = & G(|I|_{\mathcal{I}})(|\vec{u}|_{\mathcal{I}}, (\mathsf{Co}(i)\ \vec{w})) \\
& = & \Delta_{red}[C_i(I), F_i, g, \vec{w}]|_{\mathcal{I}; g \leftarrow G(pred_{(|I|_{\mathcal{I}}, (\mathsf{Co}(i)\ \vec{w}))})}.
\end{array}$$

Or d'après le lemme 4.35, on sait que $\vec{w} \in |(\vec{y} : \vec{T})|_{\mathcal{I}; X \leftarrow pred_{(|t|_{\mathcal{I}}, |\vec{v}|_{\mathcal{I}})}}$ et donc dans la formule précédente, g n'est appliqué qu'à des éléments de $pred(|I|_{\mathcal{I}}, (\mathsf{Co}(i)\ \vec{w}))$ et on peut donc appliquer le lemme 4.22 :

$$\begin{array}{rcll}
|\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}|_{\mathcal{I}} & = & |\Delta_{red}[C_i(I), F_i, g, \vec{w}]|_{\mathcal{I}; g \leftarrow G(|I|_{\mathcal{I}})} & \\
& = & |\Delta_{red}[C_i(I), F_i, g, \vec{w}]|_{\mathcal{I}; g \leftarrow |\mathsf{Fun_Elim}(I, Q, \vec{F_i})|_{\mathcal{I}}} & \text{(lemme 4.23)} \\
& = & |\Delta_{red}[C_i(I), F_i, \mathsf{Fun_Elim}(I, Q, \vec{F_i}), \vec{w}]|_{\mathcal{I}} & \\
& = & |\Delta[C_i(I), F_i, \mathsf{Fun_Elim}(I, Q, \vec{F_i})]\ \vec{v})|_{\mathcal{I}}. &
\end{array}$$

■

Lemme 4.46 (Correction de l'élimination forte) *Soit un jugement dérivable*

$$\Gamma \vdash \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\} : (Q\ \vec{u}\ t)$$

où les F_i ainsi donc que $\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}$ sont des prédicats. On suppose que pour toute interprétation $\mathcal{I}$, adaptée à Γ, on a :

1. $|t|_{\mathcal{I}} \in |(I\ \vec{u})|_{\mathcal{I}}$
2. *pour tout* i, $(F_i, \Delta\{C_i(I), Q, \mathsf{Constr}(i,I)\}, \Gamma, \mathcal{I})$ *est invariant sur son domaine.*

Soit maintenant $\mathcal{I}$ *une interprétation adaptée à* Γ *donnée ; alors*

$$(\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}, (Q\ \vec{u}\ t), \Gamma, \mathcal{I})$$

est invariant sur son domaine.

PREUVE On raisonne par récurrence sur $(|t|_{\mathcal{I}}, |\vec{u}|_{\mathcal{I}})$ par rapport à l'ordre $>_{(I,\mathcal{I})}$.

Pour abréger, on note :

$$T \equiv \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}.$$

Dire que $(T, (Q\ \vec{u}\ t), \Gamma, \mathcal{I})$ est invariant sur son domaine, revient à dire que pour toute séquence de termes $\vec{U}$ appartenant à un certain ensemble, on a :

$$(T\ \vec{U})\ \mathbf{B}^*\ T' \Rightarrow |T'|_{\mathcal{I}} = |(T\ \vec{U})|_{\mathcal{I}}$$

(pour chacun de ces $\vec{U}$ on a bien sûr $\Gamma \vdash (T\ \vec{U}) : \mathrm{Set}$). Considérons une telle séquence $\vec{U}$. On distingue les cas suivants :

- Le terme $|t|_{\mathcal{I}}$ n'admet pas de réduit de la forme $(\mathsf{Co}(i)\ \vec{m})$ (c'est-à-dire il est minimal pour $<_{(I,\mathcal{I})}$). Alors T' est forcément de la forme

  $$(\mathsf{Elim}(I', Q', \vec{u'}, t')\{\overrightarrow{F_i'}\}\ \vec{U'})$$

  et on a $|(T\ \vec{U})|_{\mathcal{I}} = \mathit{Can}(\mathrm{Set}) = |T'|_{\mathcal{I}}$.
- Le terme t n'admet pas de réduit de la forme $(\mathsf{Constr}(i,I)\ \vec{m})$, mais la forme normale de $|t|_{\mathcal{I}}$ est de la forme $(\mathsf{Co}(i)\ \vec{m})$. Alors T' est encore forcément de la forme

  $$(\mathsf{Elim}(I', Q', \vec{u'}, t')\{\overrightarrow{F_i'}\}\ \vec{U'}).$$

  Par définition, on a :

  $$\begin{aligned} |(T\ \vec{U})|_{\mathcal{I}} &= |(\Delta[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i})]\ \vec{a}\ \vec{U})|_{\mathcal{I};\vec{a}\leftarrow\vec{m}} \\ |T'|_{\mathcal{I}} &= |(\Delta[C_i(X)', F_i', \mathsf{Fun\_Elim}(I', Q', \vec{F_i}')]\ \vec{a}\ \vec{U'})|_{\mathcal{I};\vec{a}\leftarrow\vec{m}} \end{aligned}$$

  où $\vec{a}$ est une séquence de variables de terme "fraîches", de même longueur que $\vec{m}$. Il est facile de voir que le second terme est un réduit du premier. Il suffit donc de vérifier que

  $$(\Delta[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i})]\ \vec{a}\ \vec{U}), \mathrm{Set}, \Gamma(\vec{a}, \vec{V}), \mathcal{I}; \vec{a} \leftarrow \vec{m})$$

  est invariant sur son domaine, où $\vec{V}$ est définit par :

  $$C_i(I) \equiv (\vec{a} : \vec{V})(I\ \vec{n}).$$

  Ceci revient à vérifier (lemme 4.44) que

  $$(\Delta_{red}[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i}), \vec{a}]\ \vec{U}), \mathrm{Set}, \Gamma(\vec{a}, \vec{V}), \mathcal{I}; \vec{a} \leftarrow \vec{m})$$

est invariant sur son domaine, et pour cela il suffit de vérifier que

$$\Delta_{red}[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i}), \vec{a}, (Q\ u\ t), \Gamma(\vec{a}, \vec{V}), \mathcal{I}; \vec{a} \leftarrow \vec{m})$$

est invariant sur son domaine. Pour cela, il suffit d'utiliser la condition 2 et, par récurrence sur la structure de $C_i(X)$, de vérifier à chaque fois que les arguments appliqués à F_i dans $\Delta_{red}[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i}), \vec{a}]$ sont bien dans le domaine. C'est une conséquence du lemme 4.35 ; en particulier, dans le cas d'arguments récursifs, on sait, par hypothèse de récurrence, que l'application de $\mathsf{Fun\_Elim}(I, Q, \vec{F_i})$ à l'argument est un prédicat invariant sur son domaine, car l'argument est inférieur à $|t|_{\mathcal{I}}$ pour l'ordre $<_{(I,\mathcal{I})}$.

– Le terme t admet un réduit de la forme $(\mathsf{Constr}(i, I)\ \vec{v})$, et donc la forme normale de $|t|_{\mathcal{I}}$ est de la forme $(\mathsf{Co}(i)\ \vec{m})$. On suppose qu'en allant de $(T\ \vec{U})$ à T' on ne réduit jamais la racine de T :

$$T' = (\mathsf{Elim}(I', Q', \vec{u'}, t')\ \vec{U'}).$$

on sait alors (lemmes 2.19 et 2.28) que t' admet aussi un réduit de la forme $(\mathsf{Constr}(i, I')\ \vec{v'})$ et alors :

$$\begin{array}{rcl} |(T\ \vec{U})|_{\mathcal{I}} & = & |(\Delta[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i})]\ \vec{v})|_{\mathcal{I}} \\ |T'|_{\mathcal{I}} & = & (\Delta[C'_i(X), F'_i, \mathsf{Fun\_Elim}(I', Q', \vec{F'_i})]\ \vec{v'})|_{\mathcal{I}} \end{array}$$

Il est facile de voir que le second terme est un réduit du second. Par un raisonnement similaire au cas précédent, on vérifie que $((T\ \vec{U}), \mathrm{Set}, \Gamma, \mathcal{I})$ est invariant sur son domaine. Le résultat suit.

– Enfin on fait les mêmes hypothèses sur le terme t que ci-dessus, mais on suppose cette fois que l'on réduit la racine de T en allant de $(T\ \vec{U})$ vers T'. Mais dans ce cas il existe une séquence $\vec{v}$ telle que l'on puisse réordonner les réductions :

$$\begin{array}{rcl} (T\ \vec{U}) & \mathbf{B}^* & (\mathsf{Elim}(I, Q, \vec{u}, (\mathsf{Constr}(i, I)\ \vec{v}))\{\overrightarrow{F_i}\}\ \vec{U}) \\ & \mathbf{B}_\iota & (\Delta[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i})]\ \vec{v}\ \vec{U}) \\ & \mathbf{B}^* & T'. \end{array}$$

Il est clair que la première série de réductions ne change pas la valeur de l'interprétation (on ne réduit que dans un terme de preuve). La ι-réduction centrale non plus (lemme précédent). Enfin de même que ci-dessus, on vérifie que

$$((\Delta[C_i(X), F_i, \mathsf{Fun\_Elim}(I, Q, \vec{F_i})]\ \vec{v}\ \vec{U}), \mathrm{Set}, \Gamma, \mathcal{I})$$

est bien invariant sur son domaine.

■

4.12 Instanciations de contextes

En quantifiant sur l'interprétation $\mathcal{I}$, on quantifie en fait sur toutes les substitutions valides dans les termes purs correspondants aux termes de preuve. Cette quantification correspondant donc, de la manière usuelle, à la notion de réductibilité $|T|_{\mathcal{I}}$ définie sur les prédicats. En revanche elle est insuffisante pour anticiper les substitutions qui auront lieu dans les prédicats. Il nous faut donc

définir quelles seront les substitutions légales vis-à-vis de l'invariance sur le domaine. On exigera cette fois que ces substitutions soient bien-typées. On aboutit ainsi à la notion *d'instanciation de contexte* qui ressemble à l'interprétation "à la Kripke" de Coquand et Gallier [23]. Il s'agit en fait de quantifier sur toutes les dérivations commençant par une dérivation.

C'est, avec les codages des parties 4.15 et 4.16, l'une des lourdeurs dues au passage par les termes purs.

Définition 4.38 (Instanciations de contexte) *Soit Γ un contexte bien formé (c.a.d. qu'il existe un jugement dérivable $\Gamma \vdash A : B$). Soit Δ un contexte et ϕ une application associant un terme à toute variable, tel que*

$$\forall x \notin \Gamma.\phi(x) = x.$$

On dit que (Δ, ϕ) est une instanciation de Γ si Δ est bien formé et si pour toute variable x apparaissant dans Γ, on a $\Delta \vdash \phi(x) : \Gamma(x)[\phi]$, où $t[\phi]$ est le terme t dont toutes les variables libres y ont été substituées par leur valeurs $\phi(y)$.

Lemme 4.47 *Soit un jugement dérivable $\Gamma \vdash A : B$ et (Δ, ϕ) une instanciation de Γ. Alors $\Delta \vdash A[\phi] : B[\phi]$.*

PREUVE Par récurrence sur la structure de A, en utilisant les lemmes d'inversion et de substitution. ■

Définition 4.39 (Instanciations adaptées) *Soient Γ un contexte bien formé et (Δ, ϕ) une instanciation de Γ. On dit que (Δ, ϕ) est une instanciation adaptée à Γ si :*

- $\forall a \in \Gamma.|\phi(a)|_\Delta \in |\Gamma(a)[\phi]|_\Delta$
- $\forall \alpha \in \Gamma.\ (\phi(\alpha), \Gamma(\alpha)[\phi], \Delta, Can(\Delta))$ *est invariant sur son domaine.*

On remarque que la première clause est équivalente à dire que $\mathcal{C}(\Delta) \circ \phi$ est une interprétation adaptée à Γ.

On peut maintenant définir ce qui sera notre hypothèse de récurrence pour les jugements $\Gamma \vdash T : K$.

Définition 4.40 *Soit un jugement dérivable $\Gamma \vdash T : K$. On dit que toute instanciation de (T, K, Γ) est invariante si pour toute instanciation (ϕ, Δ) adaptée à Γ et pour toute interprétation $\mathcal{I}$ adaptée à Δ, $(T[\phi], K[\phi], \Delta, \mathcal{I})$ est invariant sur son domaine.*

4.13 Ordres et schémas d'ordres invariants sur leur domaine

Dans cette partie, on adapte simplement les définitions précédentes pour construire les hypothèses de récurrences correspondant aux jugements $\Gamma \vdash K : E$ et $\Gamma \vdash E : \text{Extern}$.

Définition 4.41 *Soit un jugement dérivable $\Gamma \vdash K : E$, K_0 la forme ordre-normale de K et $\mathcal{I}$ une interprétation adaptée a Γ. On définit la proposition "$(K, E, \Gamma, \mathcal{I})$ est invariant sur son domaine" par récurrence sur la structure de K_0 :*

- *si $K_0 = \text{Set}$ et $E = \text{Type}$, alors $(K, E, \Gamma, \mathcal{I})$ est invariant sur son domaine*
- *si $K_0 = [a : T]K_1$ et $E = (a : T')E_1$, alors $(K, E, \Gamma, \mathcal{I})$ est invariant sur son domaine si et seulement si $(T, \text{Set}, \Gamma, \mathcal{I})$ est invariant sur son domaine, et pour tout terme t tel que $\Gamma \vdash t : T$ soit dérivable, et $|t|_\mathcal{I} \in |T|_\mathcal{I}$, on ait $(K_1[a \setminus a], E_1[a \setminus t], \Gamma, \mathcal{I})$ invariant sur son domaine*

- *si* $K_0 = (a : T)K_1$ *et* $E = \text{Type}$*, alors* $(K, E, \Gamma, \mathcal{I})$ *est invariant sur son domaine si et seulement si* $(T, \text{Set}, \Gamma, \mathcal{I})$ *est invariant sur son domaine, et pour tout terme* t *tel que* $\Gamma \vdash t : T$ *soit dérivable, et* $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$*, on ait* $(K_1[a \setminus a], \text{Type}, \Gamma, \mathcal{I})$ *invariant sur son domaine*
- *si* $K_0 = [\alpha : K_1]K_2$ *et* $E = (\alpha : K_1')E_1$*, alors* $(K, E, \Gamma, \mathcal{I})$ *est invariant sur son domaine si et seulement si* $(K_1, \text{Type}, \Gamma, \mathcal{I})$ *est invariant sur son domaine, et pour tout prédicat* T *tel que* $\Gamma \vdash T : K_1$ *soit dérivable et* $(T_1, K_1, \Gamma, \mathcal{I})$ *invariant sur son domaine, on ait* $(K_2[\alpha \setminus T], E_1[\alpha \setminus T], \Gamma, \mathcal{I})$ *invariant sur son domaine*
- *si* $K_0 = (\alpha : K_1)K_2$ *et* $E = \text{Type}$*, alors* $(K, E, \Gamma, \mathcal{I})$ *est invariant sur son domaine si et seulement si* $(K_1, \text{Type}, \Gamma, \mathcal{I})$ *est invariant sur son domaine, et pour tout prédicat* T *tel que* $\Gamma \vdash T : K_1$ *soit dérivable et* $(T_1, K_1, \Gamma, \mathcal{I})$ *invariant sur son domaine, on ait* $(K_2[\alpha \setminus T], \text{Type}, \Gamma, \mathcal{I})$ *invariant sur son domaine.*

Remarquons que l'on n'utilise pas vraiment E *dans la définition, mais il semble plus clair de ne pas l'omettre dans l'énoncé.*

Définition 4.42 *Soit un jugement dérivable* $\Gamma \vdash E : \text{Extern}$ *et une interprétation* $\mathcal{I}$ *adaptée à* Γ*. On définit la proposition "*$(E, \Gamma, \mathcal{I})$ *est invariant sur son domaine" par :*

- *dans le cas où* $E = \text{Extern}$*,* $(\text{Extern}, \Gamma, \mathcal{I})$ *est invariant sur son domaine*
- *si* $E = (a : T)E_1$*, alors* $(E, \Gamma, \mathcal{I})$ *est invariant sur son domaine si et seulement si* $(T, \text{Set}, \Gamma, \mathcal{I})$ *est invariant sur son domaine, et pour tout jugement dérivable* $\Gamma \vdash t : T$ *avec* $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$*, on ait* $(E_1[a \setminus t], \Gamma, \mathcal{I})$ *invariant sur son domaine*
- *si* $E = (\alpha : K)E_1$*, alors* $(E, \Gamma, \mathcal{I})$ *est invariant sur son domaine si et seulement si*

$$(K, \text{Type}, \Gamma, \mathcal{I})$$

est invariant sur son domaine et pour tout jugement dérivable $\Gamma \vdash T : K$*, avec* $(T, \mathcal{K}, \Gamma, \mathcal{I})$ *invariant sur son domaine, on ait*

$$(E_1[\alpha \setminus T], \Gamma, \mathcal{I})$$

invariant sur son domaine.

On verifie que cette hypothèse suffit à traiter la règle de conversion pour les ordres :

Lemme 4.48 *Soit les jugements dérivables* $\Gamma \vdash T : K$ *et* $\Gamma \vdash K' : \text{Type}$ *et* $\mathcal{I}$ *une interprétaiton adaptée à* Γ*. On suppose* $K =_{\beta\eta\iota} K'$ *et que* $(K, \text{Type}, \Gamma, \mathcal{I})$ *et* $(K', \text{Type}, \Gamma, \mathcal{I})$ *sont invaraints sur leurs domaines. Alors si* $(T, K, \Gamma, \mathcal{I})$ *est invariant sur son domaine,* $(T, K', \Gamma, \mathcal{I})$ *l'est aussi.*

PREUVE Par récurrence sur la structure de la forme ordre-normale K_0 de K. On appelle K_0' la forme ordre-normale de K'.

- Le cas $K_0 = K_0' = \text{Set}$ est trivial.
- Si $K_0 = (a : T_0)K_1$ et $K_0' = (a : T_0')K_1'$, on a forcément $T_0 =_{\beta\eta\iota} T_0'$ et $K_1 =_{\beta\eta\iota} K_1'$. Par hypothèse, $(T_0, \text{Set}, \Gamma, \mathcal{I})$ et $(T_0', \text{Set}, \Gamma, \mathcal{I})$ sont invariants sur leurs domaines, et donc $|T_0|_{\mathcal{I}} = |T_0'|_{\mathcal{I}}$. Pour tout terme t, on a donc $\Gamma \vdash t : T_0$ et $|t|_{\mathcal{I}} \in |T_0|_{\mathcal{I}}$ si et seulement si $\Gamma \vdash t : T_0'$ et $|t|_{\mathcal{I}} \in |T_0'|_{\mathcal{I}}$. Dans ce cas, on a alors $(K_1[a \setminus t], \text{Type}, \Gamma, \mathcal{I})$ et $(K_1'[a \setminus t], \text{Type}, \Gamma, \mathcal{I})$ invariants sur leurs domaines, et donc par hypothèse de récurrence, $((T\ t), K_1[a \setminus t], \Gamma, \mathcal{I})$ est invariant sur son domaine si et seulement si $((T\ t), K_1[a \setminus t], \Gamma, \mathcal{I})$ l'est.
- Le cas $K_0 = (\alpha : K_1)K_2$ et $K_0' = (\alpha : K_1')K_2'$ est similaire.

■

Définition 4.43 *Soit un jugement dérivable* $\Gamma \vdash K : E$ *(respectivement* $\Gamma \vdash E :$ Extern*). On dit que toute instanciation de* (K, E, Γ) *(respectivement de* (E, Γ)*) est invariante si et seulement si pour toute instanciation* (ϕ, Δ) *adaptée à* Γ *et toute interprétation* $\mathcal{I}$ *adaptée à* Δ*,* $(K[\phi], E[\phi], \Delta, \mathcal{I})$ *(respectivement* $(E[\phi], \Delta, \mathcal{I})$*) est invariant sur son domaine.*

Les lemmes suivants permettent de traiter la formation d'abstraction dans les ordres :

Lemme 4.49 *Soit le jugement dérivable* $\Gamma :: (a, T) \vdash K : E$*. On suppose que toute instanciation de* $(T, \mathrm{Set}, \Gamma)$ *et de* $(K, E, \Gamma :: (a, T))$ *est invariante. Alors toute instanciation de* $([a : T]K, (a : T)E, \Gamma)$ *est invariante.*

Preuve Soit (ϕ, Δ) une instanciation de Γ et $\mathcal{I}$ une interpretation adaptée à Δ. Par hypothèse, $(T[\phi], \mathrm{Set}, \Delta, \mathcal{I})$ est invariant sur son domaine. Il nous faut donc encore vérifier, qu'étant donné $\Delta \vdash t : T[\phi]$ avec $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$, $(K[\phi; a \to t], E[\phi; a \to t], \Delta, \mathcal{I})$ est invariant sur son domaine. C'est vrai, puisque $(\phi; a \to t, \Delta)$ est une instanciation adaptée de Γ. ■

Lemme 4.50 *Soit le jugement dérivable* $\Gamma :: (\alpha, K_1) \vdash K_2 : E$*. On suppose que toute instanciation de* $(K_2, E, \Gamma :: (\alpha, K_1))$ *ou de* $(K_2, \mathrm{Type}, \Gamma)$ *est invariante. Alors toute instanciation de* $([\alpha : K_1]K_2, (\alpha : K_1)E, \Gamma)$ *est invariante.*

La preuve est similaire à la précédente.

4.14 Normalisation forte des termes purs

Le décor est maintenant en place pour dérouler la grande preuve par récurrence.

Théorème 10 *Soit un* $\Gamma \vdash t : T$ *un jugement dérivable et* $\mathcal{I}$ *une interprétation adaptée à* Γ*. Alors* $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$*.*

Preuve Cette fois, on raisonne par récurrence nœthérienne sur la taille de la dérivation. Plus exactement pour renforcer l'hypothèse de récurrence, on prouve :

- si $\Gamma \vdash t : T$ est dérivable et si $\mathcal{I}$ est une interprétation adaptée à Γ, alors $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$,
- si $\Gamma \vdash T : K$ est dérivable, alors toute instanciation de (T, K, Γ) est invariante,
- si $\Gamma \vdash K : E$ est dérivable, alors toute instanciation de (K, E, Γ) ou de (E, Γ) est invariante,
- si $\Gamma \vdash E :$ Extern, alors toute instanciation de (E, Γ) est invariante.

Une fois de plus, on passe donc en revue l'ensemble des règles d'inférence. Rappelons que $\nu(u)$ désigne la longueur de la plus longue chaîne de réductions possibles dans un terme pur u fortement normalisable. Pour une meilleure lisibilité, les nombreux cas de la preuve sont regroupés suivant la classe du terme construit ($t : T, T : K$, $K : E$ ou $E :$ Extern), c'est-à-dire suivant la clause de l'hypothèse de récurrence qu'il faut prouver.

4.14.1 Construction des termes de preuves

Les règles susceptibles d'être les dernières utilisées dans la dérivation d'un jugement $\Gamma \vdash t : T$ sont : Lam-Set, W-Set, W-Type, Var, App, Conv, Intro et W-Elim. Nous les considérons dans cet ordre. Les cas délicats ont en fait déjà été traités

La règle Lam-Set. Le cas du jugement $\Gamma \vdash [a : T_1]t : (a : T_1)T_2$ correspond exactement au lemme 4.28. Celui du jugement $\Gamma \vdash [\alpha : K]t : (\alpha : K)T$ au lemme 4.29.

La règle d'affaiblissement W-Set. Elle ne pose pas de problèmes, puisqu'une interprétation adaptée au contexte $\Gamma :: (x, M)$ est aussi adaptée au contexte Γ. Le reste suit par application immédiate de l'hypothèse de récurrence.

La règle d'affaiblissement W-Type. Ici aussi, il n'y a pas de difficultés, puisque toute instanciation du contexte $\Gamma :: (\alpha, K)$ est aussi une instanciation de Γ. Le reste suit par utilisation immédiate de l'hypothèse de récurrence.

La règle Var. Le jugement est de la forme $\Gamma \vdash a : T$; aussi si $\mathcal{I}$ est adaptée à Γ on a effectivement $|a|_{\mathcal{I}} = \mathcal{I}(a) \in |T|_{\mathcal{I}}$.

La règle App. Comme d'habitude, ce cas ne pose pas vraiment de problèmes :

• Si le jugement est $\Gamma \vdash (t_1\ t_2) \in T_1[a \setminus t_2]$, on sait par hypothèse de récurrence que $|t_2|_{\mathcal{I}} \in |T_2|_{\mathcal{I}}$ et $|t_1|_{\mathcal{I}} \in |(a : T_2)T_1|_{\mathcal{I}}$. Aussi a-t-on $(|t_1|_{\mathcal{I}}\ |t_2|_{\mathcal{I}}) = |(t_1\ t_2)|_{\mathcal{I}} \in |T_1|_{\mathcal{I};a\leftarrow|t_2|_{\mathcal{I}}}$. Comme $|T_1|_{\mathcal{I};a\leftarrow|t_2|_{\mathcal{I}}} = |T_1[a \setminus t_2]|_{\mathcal{I}}$, le résultat suit.

• Si le jugement est $\Gamma \vdash (t\ T_1) : T[\alpha \setminus T_1]$, on sait par hypothèse de récurrence que $|t|_{\mathcal{I}} \in |(\alpha : K_1)T|_{\mathcal{I}}$, donc en particulier $|t|_{\mathcal{I}} \in |T|_{\mathcal{I};\alpha\leftarrow|T_1|_{\mathcal{I}}} = |T[\alpha \setminus T_1]|_{\mathcal{I}}$. Comme $|t|_{\mathcal{I}} = |(t\ T_1)|_{\mathcal{I}}$, le résultat suit.

La règle Conv. C'est une conséquence immédiate du lemme 4.40 et des hypothèses de récurrence.

La règle Intro. C'est une conséquence immédiate du lemme 4.34.

La règle W-Elim. C'est une conséquence du lemme 4.7.

4.14.2 Construction des prédicats

Les règles susceptibles d'être les dernières d'une dérivation de $\Gamma \vdash T : K$ sont : Prod-Set, Lam-Type, W-Set, W-Type, Var, App, Conv, Ind, S-Elim. Nous les traitons dans cet ordre.

La règle Prod-Set.

• Si le jugement est de la forme $\Gamma \vdash (a : T_1)T_2 : \mathrm{Set}$, il n'y a pas de difficultés. Étant donnés Δ, ϕ et $\mathcal{I}$, on sait par hypothèse de récurrence que $|T_1[\phi]|_{\mathcal{I}} = |U_1|_{\mathcal{I}}$ pour tout U_1 tel que $T_1[\phi]\ \mathbf{B}^*\ U_1$. De même pour tout $u \in |T_1|_{\mathcal{I}}$, si $T_2[\phi]\ \mathbf{B}^*\ U_2$ alors $|T_2|_{\mathcal{I};a\leftarrow u} = |U_2|_{\mathcal{I};a\leftarrow u}$. Aussi si $(a : T_1)T_2\ \mathbf{B}^*\ U$, alors $|(a : T_1)T_2|_{\mathcal{I}} = |U|_{\mathcal{I}}$.

• Si le jugement est de la forme $\Gamma \vdash (\alpha : K)T : \mathrm{Set}$, on a vu que si $K[\phi]\ \mathbf{B}^*\ K'[\phi]$ alors $\mathcal{CR}(K[\phi]) = \mathcal{CR}(K[\phi]')$. Par ailleurs pour tout $\mathcal{C}$ élément de $\mathcal{CR}(K[\phi])$ et pour tout U tel que $T[\phi]\ \mathbf{B}^*\ U$ on a bien $|T|_{\mathcal{I};\alpha\leftarrow\mathcal{C}} = |U|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$. Donc $|((\alpha : K)T)[\phi]|_{\mathcal{I}}$ est bien invariant par réduction dans $((\alpha : K)T)[\phi]$.

La règle Lam-Type.

• Si le jugement est $\Gamma \vdash [a : T_1]T : (a : T_1)K$, étant donnés Δ, ϕ et une dérivation de $\Delta \vdash t : T_1[\phi]$, il faut vérifier que $(([a : T_1]T[\phi]\ t), K[\phi][a \setminus t], \Delta, \mathcal{I})$ est invariant sur son domaine. Le lemme 4.42 ramène cela a prouver que $(T[\phi][a \setminus t], K[\phi][a \setminus t], \Delta, \mathcal{I})$ est invariant, c'est-à-dire que $(T[\phi; a \leftarrow t], K[\phi; a \leftarrow t], \Delta, \mathcal{I})$ est invariant. Il est alors facile de voir que $(\Delta, \phi; a \leftarrow t)$ est une instanciation adaptée du contexte $\Gamma :: (a, T_1)$; il nous suffit alors d'appliquer l'une des hypothèses de récurrence qui assure que toute instanciation de $(T, K, \Gamma :: (a, T_1), \mathcal{I})$ est invariante.

• Si le jugement est $\Gamma \vdash [\alpha : K_1]T : (\alpha : K_1)K$, étant donnés Δ, ϕ, $\mathcal{I}$, il nous faut vérifier que pour tout jugement $\Gamma\Delta \vdash T_1 : K_1[\phi]$, si $(T_1, K_1[\phi], \Gamma\Delta, \mathcal{I})$ est invariant sur son domaine,

alors $((([\alpha : K_1]T)[\phi]\ T_1), K[\phi][\alpha \setminus T_1], \Gamma\Delta, \mathcal{I})$ est bien invariant sur son domaine. La preuve est quasi-identique à celle du cas précédent.

La règle VAR. Le jugement est $\Gamma \vdash \alpha : K$; l'une des hypothèses est précisément que

$$(\phi(\alpha), K[\phi], \Gamma\Delta, \mathcal{I})$$

est bien invariant sur son domaine.

La règle APP.

• Si le jugement est $\Gamma \vdash (T_1\ T_2) : K_1[\alpha \setminus T_2]$, étant donnés Δ, ϕ et $\mathcal{I}$ il nous faut vérifier que $((T_1\ T_2)[\phi], K_1[\alpha \setminus T_2], \Delta, \mathcal{I})$ est invariant sur son domaine. C'est immédiat, puisque, par hypothèse de récurrence, $(T_1[\phi], (\alpha : K_2)K_1[\phi], \Delta, \mathcal{I})$ et $(T_2[\phi], K_2[\phi], \Delta, \mathcal{I})$ sont invariants sur leurs domaines.

• Si le jugement est $\Gamma \vdash (T\ t_1) : K[a \setminus t_1]$, étant donnés Δ, ϕ et $\mathcal{I}$ il nous faut vérifier que $(T\ t_1)[\phi], K[a \setminus t_1][\phi], \Delta, \mathcal{I})$ est invariant sur son domaine. C'est immédiat, puisque $(T[\phi], (a : T_1)K[\phi], \Delta, \mathcal{I})$ est invariant sur son domaine, et qu'on sait dériver le jugement $\Delta \vdash t[\phi] : T_1[\phi]$.

La règle CONV. C'est une conséquence immédiate du lemme 4.48 et des hypothèses de récurrence.

La règle IND. On peut vérifier que $(\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}[\phi], A[\phi], \Gamma\Delta, \mathcal{I})$ est invariant sur son domaine. Si on regarde la définition de $|\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}[\phi]|_{\mathcal{I}}$, on voit qu'elle ne dépend que de $\mathcal{CR}(A)$, et des :

$$|\Delta\{C_i(X)[\phi], Q, c\}|_{\mathcal{I};X\leftarrow S;Q\leftarrow\mathcal{C};c\leftarrow\mathsf{Co}(i)}.$$

Or ces derniers sont invariants sur leurs domaines. On en déduit donc facilement que si :

$$\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}[\phi]\ \mathbf{B}^*\ I'$$

alors

$$|\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}[\phi]|_{\mathcal{I}} = |I'|_{\mathcal{I}}$$

et donc *a fortiori*, l'hypothèse de récurrence est vérifiée.

La règle S-ELIM. C'est une conséquence à peu près immédiate du lemme 4.46.

4.14.3 Construction des ordres

Les jugements de la forme $\Gamma \vdash K : E$ ne peuvent être dérivés qu'en utilisant ultimement une des règles suivantes : AX1, PROD-TYPE, LAM-EXTERN, W-SET, W-TYPE, APP et CONV. On les considère dans cet ordre.

La règle AX1. C'est immédiat.

La règle PROD-TYPE. Si le jugement est de la forme $\Gamma \vdash (a : T)K$: Type, la prémisse est $\Gamma :: (a, T) \vdash K$: Type. On a donc également une dérivation plus petite du jugement $\Gamma \vdash T$: Set. En appliquant l'hypothèse de récurrence à ces deux jugements, on obtient que $(T, \mathrm{Set}, \Gamma)$ est toujours invariant sur son domaine, et $(K_0, \Gamma :: (a, T))$ est toujours invariant où k_0 est la forme ordre-normale de K. Aussi, comme la forme ordre-normale de $(a : T)K$ est $(a : T)K_0$, on sait alors que $((a : T)K, \Gamma)$ est toujours invariant. Le cas où le jugement est de la forme $\Gamma \vdash (\alpha : K_1)K_2$: Type est tout à fait similaire au précédant.

Dans les deux cas, on a, par définition, que toute instanciation de (Type, Γ) est invariante.

La règle LAM-EXTERN.

• Si le jugement est $\Gamma \vdash [a : T]K : (a : T)E$, le lemme 4.49 assure que toute instanciation de $([a : T]K, (a : T)E, \Gamma)$ est invariante. Par ailleurs, il existe alors une dérivation de $\Gamma \vdash T : \mathrm{Set}$ à laquelle nous pouvons appliquer l'hypothèse de récurrence, et donc toute instanciation de $(T, \mathrm{Set}, \Gamma)$ est invariante. Comme l'une des prémisses est $\Gamma :: (a, T) \vdash K : E$, on sait que toute instanciation de $(K, E, \Gamma :: (a, T))$ est invariante. On peut déduire de ces deux dernières propriétés que toute instanciation de $((a : T)E, \Gamma)$ est invariante.

• Le cas où le jugement est de la forme $\Gamma \vdash [\alpha : K_1]K_2 : (\alpha : K_1)E$ est tout à fait similaire. On utilise le lemme 4.50.

Les règle W-SET **et** W-TYPE.

La règle APP. • Si le jugement est $\Gamma \vdash (K\ t) : E[a \setminus t]$, les prémisses sont $\Gamma \vdash K : (a : T)E$ et $\Gamma \vdash t : T$. Soit $[a : T']K_0$ la forme ordre-normale de K. On a alors $T =_{\beta\eta\iota} T'$. Etant donnés ϕ, Δ et $\mathcal{I}$, on sait que $|t[\phi]|_{\mathcal{I}} \in |T[\phi]|_{\mathcal{I}}$. Comme $(T[\phi], \mathrm{Set}, \Gamma, \mathcal{I})$ et $(T'[\phi], \mathrm{Set}, \Gamma, \mathcal{I})$ sont invariants sur leurs domaines, on a $|t[\phi]|_{\mathcal{I}} \in |T'|_{\mathcal{I}}$. Aussi, $(\phi; a \to t[\phi], \Delta)$ est une interprétation adaptée à $\Gamma :: (a, T)$, et donc $(K[\phi; a \to t], E[\phi; a \to t], \Delta, \mathcal{I})$ et $(E[\phi; a \to t], \Delta, \mathcal{I})$ sont invariants sur leurs domaines.

• Le cas d'un jugement $\Gamma \vdash (K\ T) : E$ est similaire.

La règle CONV. Ce cas est immédiat puisque l'hypothèse de récurrence ne dépend que de K et pas de son type E.

4.14.4 Construction des schémas d'ordres

Il ne faut traiter que le cas de la formation de produit. Les preuves sont exactement les mêmes que celles utilisées dans le cas de la règle LAM pour les ordres.

■

Ceci clos la preuve du théorème 10.

Corollaire 4.11 *Si t un terme preuve bien-formé, alors $\bar{t}$ est fortement normalisable.*

PREUVE Un terme preuve t est bien formé si et seulement s'il existe un contexte Γ et un prédicat T tel que le jugement $\Gamma \vdash t : T$ soit dérivable. Il suffit de définir une interprétation $\mathcal{I}$ admissible pour Γ, telle que pour tout a on ait $\mathcal{I}(a) = a$. En effet, alors $|t|_{\mathcal{I}} = \bar{t}$ et $|t|_{\mathcal{I}} \in |T|_{\mathcal{I}}$, et comme $|T|_{\mathcal{I}}$ est un candidat de réductibilité $\bar{t}$ est fortement normalisable.

On a dit que $\mathcal{I}(a) = a$ pour toute variable de preuve a. On définit $\mathcal{I}(\alpha)$ par récurrence structurelle sur Γ :

- Si $\Gamma = []$ on choisit $\mathcal{I}(\alpha) = \mathit{Can}(\mathrm{Set})$ pout tout α.
- Si $\Gamma = \Gamma' :: (a, T)$, on prend pour $\mathcal{I}$ l'interprétation associée à Γ'.
- Si $\Gamma = \Gamma' :: (\alpha, K)$, on sait que $\Gamma' \vdash K : \mathrm{Type}$, et donc si $\mathcal{J}$ est le contexte admissible pour Γ', on prend $\mathcal{I} \equiv \mathcal{J}; \alpha \leftarrow |K|_{\mathcal{J}}$.

■

4.15 Normalisation forte des termes démarqués

Il nous reste à vérifier que la normalisation forte des termes purs associés aux termes preuves implique bien la normalisation forte de tout les termes bien formés. Cette dernière étape, bien qu'elle utilise un codage un peu pénible, est conceptuellement bien plus facile que ce qui précède. Le seul

problème, est que la propriété de normalisation des termes purs ne nous assure pas a priori qu'il n'y a pas de séquence infinie de réductions, faisant intervenir des radicaux qui n'apparaissent pas dans le terme pur extrait. C'est le cas des radicaux de la forme :

$$([\alpha : K]t\ T)$$

ou

$$\mathsf{Elim}(I, Q, \vec{u}, (\mathsf{Constr}(i, I)\ \vec{m}))\{\overrightarrow{F_j}\} \quad \text{où } Q \text{ est un ordre.}$$

Toutefois, le résultat précédent se révèle suffisant pour capturer également le cas des radicaux ci-dessus. Concrètement on adaptera à notre système une idée de Harper, Honsell et Plotkin [132] : pour chaque terme bien formé u, on définit un terme preuve bien formé $\lceil u \rceil$ tel que si $u\ \mathbf{B}\ u'$, alors $\overline{\lceil u \rceil}\ \mathbf{B}^+\ \overline{\lceil u' \rceil}$, ce qui assurera la normalisation forte. Pour simplifier un peu l'écriture du codage, et à cause de difficultés annexes liées à notre formulation de la ι-réduction, nous choisissons de décomposer cette étape en deux : dans la présente partie, nous démontrons la normalisation forte des termes démarqués, c'est-à-dire si u est un terme bien-formé, alors $\|u\|$ est fortement normalisable, et ce quelle que soit la classe de u : terme preuve, prédicat, ordre, etc. Dans la partie suivante, nous utiliserons ce résultat pour vérifier que u lui-même est effectivement fortement normalisable. Soulignons que le résultat de normalisation forte de $\|u\|$ est en lui-même intéressant, puisqu'il suffit à établir les deux propriétés essentielles recherchées, à savoir la cohérence logique du système, et la décidabilité du typage.

4.15.1 A propos du codage

Bien que la définition exacte du codage soit très rébarbative, l'idée générale est très simple. Pour être capable de simuler toutes les réductions, y compris celles sur les prédicats par des réductions sur des termes preuves, puis des termes purs, il nous faut introduire une variable de preuve chaque fois que nous introduisons une variable de prédicats. A partir de maintenant, on suppose donc :

> *Pour chaque variable de prédicat α, on se donne une unique variable de preuve $\lceil \alpha \rceil$. On supposera que les $\lceil \alpha \rceil$ sont deux-à-deux distinctes, et qu'elles ne sont pas utilisées par ailleurs.*

Ainsi, $[\alpha : K]T$ se verra donc traduit en un terme de la forme $[\lceil \alpha \rceil : T']T''$. Comme toutefois la variable α peut être utilisée dans T, on introduit également cette dernière, ce qui donnera en définitive $\lceil [\alpha : K]T \rceil = [\alpha : K'][\lceil \alpha \rceil : T']\lceil T \rceil$. Il reste alors à définir K' et T'. Pour ce faire, on va définir également les fonctions τ, χ, ξ qui correspondront aux types des termes codés. On aura :

$$\begin{array}{rll} \lceil U \rceil : \tau(V) & \text{si} & U : V \\ \tau(U) : \chi(V) & \text{si} & U : V \text{ et } U \text{ n'est pas un terme preuve} \\ \chi(V) : \xi(V) & \text{si} & U : V \text{ et } U \text{ n'est ni un terme preuve ni un prédicat} \end{array}$$

Pour coder les sortes Set, Type et Extern, on rajoutera systématiquement dans le contexte une variable de prédicat o : Set. On se donne également une variable de preuve $b_\perp$ de type $(\alpha : \text{Set})\alpha$, qui nous aidera en particulier lors de la définition des codages des produits ($\lceil (x : T_1)T_2 \rceil$ par exemple). Enfin on se donne une nouvelle variable de preuve "muette", $\jmath$, que l'on suppose non utilisée par ailleurs, et qui jouera un rôle similaire à _ dans la définition des termes démarqués.

Toute ces fonctions sont définies simultanément par récurrence sur la structure des termes. Les définitions sont données dans les figures qui suivent.

$$
\begin{array}{|l|}
\hline
\tau(\text{Extern}) \equiv o \\
\chi(\text{Extern}) \equiv \text{Set} \\
\xi(\text{Extern}) \equiv \text{Type} \\
\\
\tau((a:T)E) \equiv (a:\tau(T))\tau(E) \\
\chi((a:T)E) \equiv (a:\tau(T))\chi(E) \\
\xi((a:T)E) \equiv (a:\tau(T))\xi(E) \\
\\
\tau((\alpha:K)E) \equiv (\alpha:\chi(K))(\lceil\alpha\rceil:\tau(K))\tau(E) \\
\chi((\alpha:K)E) \equiv (\alpha:\chi(K))(\lceil\alpha\rceil:\tau(K))\chi(E) \\
\xi((\alpha:K)E) \equiv (\alpha:\chi(K))(\lceil\alpha\rceil:\tau(K))\xi(E) \\
\hline
\end{array}
$$

FIG. 4.1 – Codage des schémas d'ordres

4.15.2 Propriétés du codage

On commence par établir que le codage est compatible avec la substitution.

Lemme 4.51 *Pour tout terme A bien typé dans un contexte Γ, pour toute variable de preuve a liée dans Γ et tout terme de preuve t tel que $\Gamma \vdash t : \Gamma(a)$, on a :*

$$
\begin{array}{lcl}
\lceil A[a \setminus t]\rceil & = & \lceil A\rceil[a \setminus \lceil t\rceil] \\
\tau(A[a \setminus t]) & = & \tau(A)[a \setminus \lceil t\rceil] \textit{ si } \tau(A) \textit{ est défini} \\
\chi(A[a \setminus t]) & = & \chi(A)[a \setminus \lceil t\rceil] \textit{ si } \chi(A) \textit{ est défini}
\end{array}
$$

Lemme 4.52 *Pour tout terme A bien typé dans un contexte Γ, pour toute variable de prédicat α liée dans Γ et tout prédicat T tel que $\Gamma \vdash T : \Gamma(\alpha)$, on a :*

$$
\begin{array}{lcl}
\lceil A[\alpha \setminus T]\rceil & = & \lceil A\rceil[\lceil\alpha\rceil \setminus \lceil T\rceil][\alpha \setminus \tau(T)] \\
\tau(A[\alpha \setminus T]) & = & \tau(A)[\lceil\alpha\rceil \setminus \lceil T\rceil][\alpha \setminus \tau(T)] \textit{ si } A \textit{ n'est pas un terme preuve} \\
\chi(A[\alpha \setminus T]) & = & \chi(A)[\lceil\alpha\rceil \setminus \lceil T\rceil][\alpha \setminus \tau(T)] \textit{ si } A \textit{ est un ordre}
\end{array}
$$

PREUVE A chaque fois par récurrence sur la structure des termes. ■

En conséquence, la réduction passe bien au codage :

Lemme 4.53 (codage et β-réduction) *Pour tout terme A bien typé dans un contexte Γ, si $A\ \mathbf{B}_\beta\ A'$, on a :*

$$
\begin{array}{lll}
\lceil A\rceil & \mathbf{B}_\beta^+ & \lceil A'\rceil \\
\tau(A) & \mathbf{B}_\beta^* & \tau(A') \textit{ si } A \textit{ n'est pas un terme preuve} \\
\chi(A) & \mathbf{B}_\beta^* & \chi(A') \textit{ si } A \textit{ est un ordre.}
\end{array}
$$

De plus si $\|A\|\ \mathbf{B}_\beta\ A'$, alors il existe A'', tel que $|\lceil A\rceil|\ \mathbf{B}^+\ |\lceil A''\rceil|$ et $\|A''\| = A'$.

PREUVE Là encore par récurrence structurelle sur A. Considérons quelques cas clés :

$$\lceil \mathrm{Set} \rceil \equiv (b_\perp\ o)$$
$$\tau(\lceil \mathrm{Set} \rceil) \equiv o$$
$$\chi(\lceil \mathrm{Set} \rceil) \equiv \mathrm{Set}$$

$$\lceil (A : T)K \rceil \equiv (b_\perp\ \tau(T) \to ((a : \tau(T))\tau(K)) \to o\ \lceil T \rceil\ [a : \tau(T)]\lceil K \rceil)$$
$$\tau((A : T)K) \equiv (a : \tau(T))\tau(K)$$
$$\chi((A : T)K) \equiv (a : \tau(T))\chi(K)$$

$$\lceil (\alpha : K_1)K_2 \rceil \equiv (b_\perp\ \tau(K_1) \to ((\alpha : \chi(K_1))(\lceil \alpha \rceil : \tau(K_1))\tau(K_2)) \to o\ \lceil K_1 \rceil\ [\alpha : \chi(K_1)][\lceil \alpha \rceil : \tau(K_1)]\lceil K_2 \rceil)$$
$$\tau((\alpha : K_1)K_2) \equiv (\alpha : \chi(K_1))(\lceil \alpha \rceil : \tau(K_1))\tau(K_2)$$
$$\chi((\alpha : K_1)K_2) \equiv (\alpha : \chi(K_1))(\lceil \alpha \rceil : \tau(K_1))\chi(K_2)$$

$$\lceil [a : T]K \rceil \equiv [a : \tau(T)]\lceil K \rceil$$
$$\tau([a : T]K) \equiv [a : \tau(T)]\tau(T)$$
$$\chi([a : T]K) \equiv [a : \tau(T)]\chi(K)$$

$$\lceil [\alpha : K_1]K_2 \rceil \equiv [\alpha : \chi(K_1)][\lceil \alpha \rceil : \tau(K_1)]\lceil K_2 \rceil$$
$$\tau([\alpha : K_1]K_2) \equiv [\alpha : \chi(K_1)][\lceil \alpha \rceil : \tau(K_1)]\tau(K_2)$$
$$\chi([\alpha : K_1]K_2) \equiv [\alpha : \chi(K_1)][\lceil \alpha \rceil : \tau(K_1)]\chi(K_2)$$

$$\lceil (K\ t) \rceil \equiv (\lceil K \rceil\ \lceil t \rceil)$$
$$\tau((K\ t)) \equiv (\tau(K)\ \lceil t \rceil)$$
$$\chi((K\ t)) \equiv (\chi(K)\ \lceil t \rceil)$$

$$\lceil (K\ T) \rceil \equiv (\lceil K \rceil\ \tau(T)\ \lceil T \rceil)$$
$$\tau((K\ T)) \equiv (\tau(K)\ \tau(T)\ \lceil T \rceil)$$
$$\chi((K\ T)) \equiv (\chi(K)\ \tau(T)\ \lceil T \rceil)$$

FIG. 4.2 – Codage des ordres

$\tau(\alpha) \equiv \alpha$
$\lceil \alpha \rceil \equiv \lceil \alpha \rceil$

$\tau((a : T_1)T_2) \equiv (a : \tau(T_1))\tau(T_2)$
$\lceil (a : T_1)T_2 \rceil \equiv (b_\perp\ (o \to (\tau(T_1) \to o) \to o)\ \lceil T_1 \rceil\ [a : \tau(T_1)]\lceil T_2 \rceil)$

$\tau((\alpha : K)T) \equiv (\alpha : \chi(K))(\lceil \alpha \rceil : \tau(K))\tau(T)$
$\lceil (\alpha : K)T \rceil \equiv (b_\perp\ (o \to (\chi(K) \to \tau(K) \to o) \to o)\ \lceil K \rceil\ [\alpha : \chi(K)][\lceil \alpha \rceil : \tau(K)]\lceil T \rceil)$

$\tau([a : T_1]T_2) \equiv [a : \tau : (T_1)]\tau(T_2)$
$\lceil [a : T_1]T_2 \rceil \equiv [a : \tau(T_1)]\lceil T_2 \rceil$

$\tau([\alpha : K]T) \equiv [\alpha : \chi(K)][\lceil \alpha \rceil : \tau(K)]\tau(T)$
$\lceil [\alpha : K]T \rceil \equiv [\alpha : \chi(K)][\lceil \alpha \rceil : \tau(K)]\lceil T \rceil$

$\tau(T\ t) \equiv (\tau(T)\ \lceil t \rceil)$
$\lceil (T\ t) \rceil \equiv (\lceil T \rceil\ \lceil t \rceil)$

$\tau((T_1\ T_2)) \equiv (\tau(T_1)\ \tau(T_2)\ \lceil T_2 \rceil)$
$\lceil (T_1\ T_2) \rceil \equiv (\lceil T_1 \rceil\ \tau(T_2)\ \lceil T_2 \rceil)$

$\tau(\mathsf{Ind}(X : A)\{\overrightarrow{C_i}\}) \equiv \mathsf{Ind}(X : \chi(A))\{\overrightarrow{\tau(C_i)}\}$
$\lceil \mathsf{Ind}(X : A)\{\overrightarrow{C_i}\} \rceil \equiv (b_\perp\ (o \to (\chi(A) \to \tau(A) \to (o \to o \to \ldots \to o) \to o) \to o)\ \lceil A \rceil$
$[X : \chi(A)][\lceil X \rceil : \tau(A)][\jmath : o \to o \to \ldots \to o](\jmath\ \lceil \vec{C_i} \rceil))$

$\tau(\mathsf{Elim}(\mathsf{Ind}(X : A)\{\}, Q, \vec{u}, t)\{\}) \equiv \mathsf{Elim}(\mathsf{Ind}(X : \tau(A))\{\}), \chi(Q), \tau(\vec{u}), \lceil t \rceil)\{\}$
$\lceil \mathsf{Elim}(\mathsf{Ind}(X : A)\{\}, Q, \vec{u}, t)\{\} \rceil \equiv ([\jmath : o][\jmath : o]\mathsf{Elim}(\tau(\mathsf{Ind}(X : A)\{\}), \chi(Q), \tau(\vec{u}), \lceil t \rceil)\{\}\ \lceil A \rceil\ \lceil Q \rceil)$

$\tau(\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\}) \equiv \mathsf{Elim}(\tau(I), \chi(Q), \tau(\vec{u}), \lceil t \rceil)\{\overrightarrow{\tau(F_i)}\}$
$\lceil \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_i}\} \rceil \equiv \mathsf{Elim}(\tau(I), \chi(Q), \tau(\vec{u}), \lceil t \rceil)\{\overrightarrow{([\jmath : \tau(A)][\jmath : o]\lceil F_i \rceil\ \lceil I \rceil\ (\lceil Q \rceil\ \tau(\vec{u})\ \lceil t \rceil))}\}$

FIG. 4.3 – Codage des prédicats

$$
\begin{array}{|l|}
\hline
\ulcorner a \urcorner \equiv a \\
\ulcorner [a : T]t \urcorner \equiv [a : \tau(T)]\ulcorner t \urcorner \\
\ulcorner [\alpha : K]t \urcorner \equiv [\alpha : \chi(K)][\ulcorner \alpha \urcorner : \tau(K)]\ulcorner t \urcorner \\
\ulcorner (t_1\ t_2) \urcorner \equiv (\ulcorner t_1 \urcorner\ \ulcorner t_2 \urcorner) \\
\ulcorner (t\ T) \urcorner \equiv (\ulcorner t \urcorner\ \tau(T)\ \ulcorner T \urcorner) \\
\ulcorner \mathsf{Constr}(i, \mathsf{Ind}(X : A)\{\overrightarrow{C_j}\}) \urcorner \equiv ([\jmath : \tau(A)]\mathsf{Constr}(i, \tau(\mathsf{Ind}(X : A)\{\overrightarrow{C_j}\}))\ \ulcorner \mathsf{Ind}(X : A)\{\overrightarrow{C_j}\} \urcorner) \\
\ulcorner \mathsf{Elim}(\mathsf{Ind}(X : A)\{\}, Q, \vec{u}, t)\{\} \urcorner \equiv ([\jmath : o][\jmath : o]\mathsf{Elim}(\mathsf{Ind}(X : \tau(A))\{\}, \tau Q, \tau(\vec{u}), \ulcorner t \urcorner)\{\}\ \ulcorner A \urcorner\ (\ulcorner Q \urcorner\ \tau(\vec{u})\ \ulcorner t \urcorner)) \\
\ulcorner \mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{f_i}\} \urcorner \equiv \mathsf{Elim}(\tau(I), \chi(Q), \tau(\vec{u}), \ulcorner t \urcorner)\{\overrightarrow{([\jmath : \tau(A)][\jmath : o]\ulcorner f_i \urcorner\ \ulcorner I \urcorner\ (\ulcorner Q \urcorner\ \tau(\vec{u})\ \ulcorner t \urcorner))}\} \\
\hline
\end{array}
$$

FIG. 4.4 – Codage des termes preuves

$$
\begin{array}{|l|}
\hline
\tau([]) \equiv [] :: (o, \mathrm{Set}) :: (b_\perp, (\alpha : \mathrm{Set})\alpha) \\
\tau(\Gamma :: (a, T)) \equiv \tau(\Gamma) :: (a, \tau(T)) \\
\tau(\Gamma :: (\alpha, K)) \equiv \tau(\Gamma) :: (\alpha, \chi(K)) :: (\ulcorner \alpha \urcorner, \tau(K)) \\
\hline
\end{array}
\qquad
\begin{array}{|l|}
\hline
\tau([]) \equiv [] :: (o, \mathrm{Set}) :: (b_\perp, (\alpha : \mathrm{Set})\mathrm{Set}) \\
\tau(t :: \vec{u}) \equiv \ulcorner t \urcorner :: \tau(\vec{u}) \\
\tau(T :: \vec{u}) \equiv \ulcorner T \urcorner :: \tau(T) :: \tau(\vec{u}) \\
\hline
\end{array}
$$

FIG. 4.5 – Codage des contextes et des séquences de termes

– Si $A = ([a : T_1]t_2\ t_1)$ avec $A' = t_2[a \setminus t_1]$, alors on a

$$
\begin{array}{rcl}
\ulcorner A \urcorner & = & ([a : \tau(T_1)]\ulcorner t_2 \urcorner\ \ulcorner t_1 \urcorner) \\
& \mathbf{B}_\beta & \ulcorner t_2 \urcorner[a \setminus \ulcorner t_1 \urcorner] \\
& = & \ulcorner t_2[a \setminus t_1] \urcorner \\
& = & \ulcorner A' \urcorner
\end{array}
$$

et de même

$$
\begin{array}{rcl}
\|\ulcorner A \urcorner\| & = & (\lambda a.\|\ulcorner t_2 \urcorner\|\ \|\ulcorner t_1 \urcorner\|) \\
& \mathbf{B}_\beta & \|\ulcorner t_2 \urcorner\|[a \setminus \|\ulcorner t_1 \urcorner\|] \\
& = & \|\ulcorner t_2 \urcorner[a \setminus \ulcorner t_1 \urcorner]\| \\
& = & \|\ulcorner t_2[a \setminus t_1] \urcorner\|
\end{array}
$$

– Si $A = ([\alpha : K]T_1\ T_2)$ avec $A' = T_1[\alpha \setminus T_2]$, alors :

$$
\begin{array}{rcl}
\ulcorner A \urcorner & = & ([\alpha : \chi(K)][\ulcorner \alpha \urcorner : \tau(K)]\ulcorner T_1 \urcorner\ \tau T_2\ \ulcorner T_2 \urcorner) \\
& \mathbf{B}_\beta & \ulcorner T_1 \urcorner[\alpha \setminus \tau(T_2)][\ulcorner \alpha \urcorner \setminus \ulcorner T_2 \urcorner]
\end{array}
$$

et

$$
\begin{array}{rcl}
\|\ulcorner A \urcorner\| & = & (\lambda \ulcorner \alpha \urcorner.\|\ulcorner T_1 \urcorner\|\ \|\ulcorner T_2 \urcorner\|) \\
& \mathbf{B}_\beta & \|\ulcorner T_1 \urcorner\|[\ulcorner \alpha \urcorner \setminus \|\ulcorner T_2 \urcorner\|] \\
& = & \|\ulcorner T_1 \urcorner[\ulcorner \alpha \urcorner \setminus \ulcorner T_2 \urcorner]\| \\
& = & \|\ulcorner T_1[\alpha \setminus T_2] \urcorner\|
\end{array}
$$

■

Lemme 4.54 (codage et ι-réduction) *Pour tout terme A bien typé dans un contexte Γ, si $A\ \mathbf{B}_\iota\ A'$, on a :*

$$\begin{array}{rll} \lceil A \rceil & \mathbf{B}_\iota^+ & \lceil A' \rceil \\ \chi(A) & \mathbf{B}_\iota^* & \chi(A') \textit{ si } A \textit{ est un ordre.} \end{array}$$

De plus, si $\|\lceil A \rceil\|\ \mathbf{B}_{\iota_0}\ A'$[6], il existe A'' tel que $|\lceil A \rceil|\ \mathbf{B}^+\ |\lceil A'' \rceil|$ et $\|\lceil A'' \rceil\| = A'$

Lemme 4.55 (codage et variables libres) *Pour tout terme A bien typé dans un contexte Γ, les variables libres de $\lceil A \rceil$, $\tau(A)$, $\chi(A)$ et $\xi(A)$ (si ces termes existent) appartiennent toutes à $FV(A) \cup \{o, b_\perp\}$.*

PREUVE La preuve est immédiate par récurrence structurelle sur A. ■

Lemme 4.56 (codage et η-réduction) *Pour tout terme A bien typé dans un contexte Γ, si $A\ \mathbf{B}_\eta\ A'$, on a :*

$$\begin{array}{rll} \lceil A \rceil & \mathbf{B}_{\beta\iota}^+ & \lceil A' \rceil \\ \tau(A) & \mathbf{B}_{\beta\iota}^* & \tau(A') \textit{ si } A \textit{ n'est pas un terme preuve} \\ \chi(A) & \mathbf{B}_{\beta\eta}^* & \chi(A') \textit{ si } A \textit{ est un ordre} \end{array}$$

PREUVE On raisonne par récurrence sur la structure de A ; il suffit bien sûr de considérer les cas où A est lui-même le radical réduit (où on réduit la racine du terme).

- Si $A = [a : T](U\ a)$ avec $a \notin FV(U)$, on a $a \notin FV(\lceil U \rceil)$, $a \notin FV(\tau(U))$ si $\tau(U)$ est défini, etc. Et donc :

$$\begin{array}{rcl} \lceil [a : T](U\ a) \rceil & = & [a : \tau(T)](\lceil U \rceil\ a) \\ & \mathbf{B}_\eta & \lceil U \rceil \end{array}$$

$$\begin{array}{rcl} \tau([a : T](U\ a)) & = & [a : \tau(T)](\tau(U)\ a) \\ & \mathbf{B}_\eta & \tau(U) \end{array}$$

  la preuve pour χ étant identique à celle pour τ.
- Si $A = [\alpha : K](U\ \alpha)$, on a $\alpha \notin FV(U)$, $\alpha \notin FV(\tau(U)$ si $\tau(U)$ est défini, etc. Et donc :

$$\begin{array}{rcl} \lceil [\alpha : K](U\ \alpha) \rceil & = & [\alpha : \chi(K)][\lceil \alpha \rceil : \tau(K)](\lceil U \rceil\ \alpha\ \lceil \alpha \rceil) \\ & \mathbf{B}_\eta & [\alpha : \chi(K)](\lceil U \rceil\ \alpha) \\ & \mathbf{B}\ \eta & \lceil U \rceil \end{array}$$

$$\begin{array}{rcl} \tau([\alpha : K](U\ \alpha)) & = & [\alpha : \chi(K)][\lceil \alpha \rceil : \tau(K)](\tau(U)\ \alpha\ \lceil \alpha \rceil) \\ & \mathbf{B}_\eta & [\alpha : \chi(K)](\tau(U)\ \alpha) \\ & \mathbf{B}_\eta & \tau(U) \end{array}$$

  la preuve pour χ étant identique à celle pour τ.

[6] Rappellons que la ι_0-réduction (définition 2.20) correspond simplement à la ι-réduction pour les termes $\|A\|$.

■

Pour vérifier que le codage préserve le typage, une condition essentielle est que si $A =_{\beta\eta\iota} A'$, alors $\lceil A \rceil =_{\beta\eta\iota} \lceil A' \rceil$, $\tau A =_{\beta\eta\iota} \tau A'$, etc. Malheureusement, les lemmes ci-dessus ne suffisent pas tout à fait pour la prouver, toujours à cause de la non-confluence. On va donc repasser une dernière fois par les termes démarqués 2.19. On peut aisément étendre les définitions de $\lceil\rceil, \tau, \chi$ et ξ aux termes démarqués $\|A\|$, avec A bien formé. Il suffit pour cela de prendre :

Définition 4.44 *On étend les définitions précédentes aux termes $\|A\|$ avec A bien formé en reprenant exactement les définitions précédentes avec de plus :*

$$\lceil \_ \rceil = \tau(\_) = \chi(\_) = \xi(\_).$$

On aura par exemple :

$$\lceil \|[a : T]t\| \rceil = [a : \_]\lceil \|t\| \rceil.$$

Les deux résultats dont nous avons besoin sont alors :

Lemme 4.57 *Soit A un terme bien formé. Si $\|A\|\ \mathbf{B}_\eta\ A'$, alors :*

$$\begin{array}{rcl}
\lceil \|A\| \rceil & \mathbf{B}^* & \lceil A' \rceil \\
\tau(\|A\|) & \mathbf{B}^* & \tau(A') \textit{ si } \tau(A) \textit{ est défini} \\
\chi(\|A\|) & \mathbf{B}^* & \chi(A') \textit{ si } \chi(A) \textit{ est défini} \\
\xi(\|A\|) & \mathbf{B}^* & \xi(A') \textit{ si } \xi(A) \textit{ est défini.}
\end{array}$$

La démonstration est quasiment identique à celle du lemme 4.56.

Lemme 4.58 *Soit A un terme bien formé. On a :*

$$\begin{array}{rcl}
\lceil A \rceil & =_{\beta\eta\iota} & \lceil \|A\| \rceil \\
\tau(A) & =_{\beta\eta\iota} & \tau(\|A\|) \textit{ si } \tau(A) \textit{ est défini} \\
\chi(A) & =_{\beta\eta\iota} & \chi(\|A\|) \textit{ si } \chi(A) \textit{ est défini} \\
\xi(A) & =_{\beta\eta\iota} & \xi(\|A\|) \textit{ si } \xi(A) \textit{ est défini.}
\end{array}$$

PREUVE Par simple récurrence sur la structure de A. Les seuls cas pertinents sont bien sûr ceux où A est de la forme $[x : B]C$:

– si $A = [a : T]t$, alors :

$$\begin{array}{rll}
\lceil A \rceil & = [a : \tau(T)]\lceil t \rceil & \\
& =_{\beta\eta\iota} & [a : _]\lceil t \rceil \\
& =_{\beta\eta\iota} & [a : _]\lceil \|t\| \rceil \\
& = & \lceil \|A\| \rceil
\end{array}$$

– si $A = [\alpha : K]t$ alors :

$$\begin{array}{rll}
\lceil A \rceil & =_{\beta\eta\iota} & [\alpha : _][\lceil \alpha \rceil : _]\lceil \|t\| \rceil \\
& = & \lceil \|A\| \rceil
\end{array}$$

– si $A = [a : T_1]T_2$ alors :

$$\begin{array}{rcl} \lceil A \rceil & = & [a : \tau T_1]\lceil T_1 \rceil \\ & =_{\beta\eta\iota} & [a : \_]\lceil \|T_2\| \rceil \\ & = & \lceil \|A\| \rceil \end{array}$$

et :

$$\begin{array}{rcl} \tau(A) & = & [a : \tau(T_1)]\tau(T_2) \\ & =_{\beta\eta\iota} & [a : \_]\tau(\|T_2\|) \\ & = & \tau(\|A\|) \end{array}$$

– si $A = [\alpha : K]T$ alors :

$$\begin{array}{rcl} \lceil A \rceil & = & [\alpha : \chi(K)][\lceil \alpha \rceil : \tau(K)]\lceil T \rceil \\ & =_{\beta\eta\iota} & [\alpha : \_][\lceil \alpha \rceil : \_]\lceil \|T\| \rceil \\ & = & \lceil \|A\| \rceil \end{array}$$

Les autres cas sont tous très similaires. ■

Lemme 4.59 (codage et conversion) *Soient deux jugements dérivables* $\Gamma \vdash A : s$ *et* $\Gamma \vdash A' : s'$ *avec* $s, s' \in \{\text{Set}, \text{Type}, \text{Extern}\}$. *Si* $A =_{\beta\eta\iota} A'$, *alors :*

$$\tau(A) =_{\beta\eta\iota} \tau(A')$$

et si $s, s' \in \{\text{Type}, \text{Extern}\}$ *:*

$$\chi(A) =_{\beta\eta\iota} \chi(A')$$

et si $s = s' = \text{Extern}$ *:*

$$\xi(A) =_{\beta\eta\iota} \xi(A').$$

Lemme 4.60 (codage et typage) *Soit un jugement dérivable* $\Gamma \vdash A : B$. *Si* $B \neq \text{Extern}$, *les jugements suivants sont dérivables :*

$\tau(\Gamma) \vdash \lceil A \rceil : \tau(B)$ *et* $\tau(\Gamma) \vdash \tau(B) : \text{Set}$
$\tau(\Gamma) \vdash \tau(A) : \chi(B)$ *et* $\tau(\Gamma) \vdash \chi(B) : \text{Type}$ *si* A *n'est pas un terme preuve*
$\tau(\Gamma) \vdash \chi(A) : \xi(B)$ *et* $\tau(\Gamma) \vdash \xi(B) : \text{Extern}$ *si* A *est un ordre*

Et si $B = \text{Extern}$ *on peut dériver :*

$\tau(\Gamma) \vdash \lceil A \rceil : o$
$\tau(\Gamma) \vdash \tau(A) : \text{Set}$
$\tau(\Gamma) \vdash \chi(A) : \text{Type}$
$\tau(\Gamma) \vdash \xi(A) : \text{Extern}.$

PREUVE C'est bien sûr une récurrence sur la structure de la dérivation. ■

Lemme 4.61 *Pour tout terme* u *bien typé,* $\|u\|$ *est fortement normalisable pour la* $\beta\eta\iota_0$*-réduction.*

PREUVE Il est évident qu'il ne peut y avoir de séquence infinie de η-réductions (la taille du terme décroît strictement). En utilisant le lemme de retard de η, on peut aisément en conclure que $\|u\|$ est fortement $\beta\eta\iota_0$-normalisable si et seulement s'il est fortement $\beta\iota_0$-normalisable. Supposons que nous ayons une suite infinie de termes $(u_i)_{i\in\mathsf{N}}$ avec $u_0 = \|u\|$ et $u_i \ \mathbf{B}^1_{\beta\iota_0} \ u_{i+1}$. Alors d'après les lemmes précédents, il existe une suite de termes $(v_i)_{i\in\mathsf{N}}$, avec $\|\lceil v_i \rceil\| = u_i$, et $|\lceil v_i \rceil| \ \mathbf{B}^+ \ |\lceil v_i + 1 \rceil|$. Ceci est bien sûr en contradiction avec le fait que $|\lceil v_0 \rceil|$ (c'est-à-dire $|\lceil u \rceil|$) est fortement normalisable. ■

4.16 Normalisation forte des termes bien-formés

Pour montrer maintenant que tout terme bien typé est fortement normalisable, il nous suffit d'émuler les réductions pouvant apparaître dans les annotations de type dans les λ-abstractions. L'idée est d'utiliser un codage, où $[x : U]V$ est remplacé par quelque chose comme $[x : U]([\_ : \mathrm{Set}]V\ U)$. On a ainsi une occurrence de U qui n'est pas en annotation, et qui peut être effacée à volonté (si l'abstraction disparaît lors d'une réduction). En construisant le codage, il faut juste faire attention aux deux points suivants :

- on ne peut pas construire un terme $([\_ : \mathrm{Type}]U\ K)$ (pour coder $[\alpha : K]U$) ; il faut donc un codage de K sous forme de prédicat
- la ι-réduction crée des λ-abstractions dont il faut tenir compte.

Le premier problème se pose déjà avec les PTS et est aisément résolu [56, 40]. La solution du second est un peu plus technique. On commence par spécifier le codage :

Définition 4.45 *On définit inductivement le prédicat binaire "U code V", noté $U \mathsf{c} V$, sur les termes, par les clauses suivantes :*

$$
\begin{array}{lcl}
 & & x \mathsf{c} x \\
 & & s \mathsf{c} s \\
 & & o \mathsf{c} \mathrm{Set} \\
\forall i. U'_i \mathsf{c} U_i \wedge V' \mathsf{c} V & \Rightarrow & (U'\ V') \mathsf{c} (U\ V) \\
 & \Rightarrow & [\vec{x_i} : \vec{U_i}]([\vec{\_}_i : \vec{\mathrm{Set}}]V'\ \vec{U'_i}) \mathsf{c} [\vec{x_i} : \vec{U_i}]V \\
 & \Rightarrow & (x : U')V' \mathsf{c} (X : U)V \\
V' \mathsf{c} V \wedge \forall i. (U'_i \mathsf{c} U_i \wedge W'_i \mathsf{c} W_i) & \Rightarrow & ([\vec{x_i} : \vec{U_i}]([\vec{\_} : \vec{o}][\vec{x_i} : \vec{U_i}]W'\ \vec{V'_i})\ \vec{U'_i})\ \vec{V'_i}) \mathsf{c} \\
 & & \qquad ([\vec{x_i} : \vec{U_i}]W\ \vec{V_i}) \\
A' \mathsf{c} A \wedge \forall i. C_i(X)' \mathsf{c} C_i(X) & \Rightarrow & \mathsf{Ind}(X : A')\{\overrightarrow{C_i(X)}\} \mathsf{c} \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\} \\
I' \mathsf{c} I & \Rightarrow & \mathsf{Constr}(i, I') \mathsf{c} \mathsf{Constr}(i, I) \\
U\ \mathbf{B}_\beta\ V' \wedge V' \mathsf{c} V & \Rightarrow & U \mathsf{c} V \\
\mathsf{Ind}(X : (\vec{z_k} : \vec{A'_k}\mathrm{Set})\{\overrightarrow{C'_i(X)}\} & & \\
\qquad \mathsf{c}\ \mathsf{Ind}(X : (\vec{z_k} : \vec{A_k})\mathrm{Set})\{\overrightarrow{C_i(X)}\} & & \\
\wedge\ Q' \mathsf{c} Q & & \\
\wedge\ \forall k. u'_k \lceil u \rceil_k & & \\
\wedge\ t' \mathsf{c} t & & \\
\wedge\ \forall i. f'_i \mathsf{c} f_i & & \\
\wedge\ \forall i. C_i(I') \equiv (\vec{y_m^i} : \vec{T_m^i})(I\ \vec{V_k^i}) & & \\
\wedge\ \forall i, m. T'^i_m \mathsf{c} T^i_m & & \\
\forall i. \Delta\{C_i(I), Q, \mathsf{Constr}(i, I)\} \equiv (\vec{y_j^{i\prime}} : \vec{U_j^i})\vec{W_j^i} & \Rightarrow & \mathsf{Elim}(I', Q', \vec{u'_k}, t')\{\overrightarrow{f''_i}\} \mathsf{c} \mathsf{Elim}(I, Q, \vec{u_k}, t)\{\overrightarrow{f_i}\} \\
 & & \textit{avec } \forall i. f''_i = ([\vec{\_}^{\,i}_m : \mathrm{Set}]f'''_i\ \vec{T^i_m}') \\
 & & \textit{et } f'''_i = [\vec{y_j^{i\prime}} : \vec{U_j^i}](f'_i\ \vec{W_j^i})
\end{array}
$$

où, dans la dernière clause, $W_j^i = y_j^{i\prime}$ si le terme correspond à un argument de constructeur (U_j^i est un T_m^i), et s'il correspond à un appel récursif avec $T_m^i \equiv (\vec{w_p} : \vec{M_p})(c : (X\ \vec{t}))(Q\ \vec{t}\ c)$ alors $W_j^i \mathsf{c} [\vec{w_p} : \vec{M_p}][c : (X\ \vec{t})](y_j^{i\prime}\ \vec{w_p})$.

Lemme 4.62 *Si U_1 $\mathbf{B}_{\beta\iota}$ U_2 et U_3 c U_1, alors il existe un terme U_4 tel que U_3 $\mathbf{B}^+_{\beta\iota}$ U_4 et U_4 c U_2. De plus, $\|U_3\|$ $\mathbf{B}^+_{\beta\iota_0}$ $\|U_4\|$.*

PREUVE Par récurrence sur la structure de la preuve de U_1 $\mathbf{B}_{\beta\iota}$ U_2. Tous les cas sont à peu près immédiats, sauf celui où U_1 est le ι-redex réduit qui semble malheureusement impossible à détailler formellement. On se contente donc d'examiner l'exemple des entiers naturels ; la réduction est alors :

$$\mathsf{Elim}(\mathrm{Nat}, Q, [], (S\ p))\{f_0, f_S\}\ \mathbf{B}_\iota\ ([n : \mathrm{Nat}](f_S\ n\ (\mathsf{Fun\_Elim}(\mathrm{Nat}, Q, [f_0; f_S])\ n))\ p).$$

Soit U tel que U c $\mathsf{Elim}(\mathrm{Nat}, Q, [], (S\ p))\{f_0, f_S\}$. On a forcément

$$U = \mathsf{Elim}(\mathrm{Nat}, Q', [], (S\ p'))\{f''_0, f''_S\}$$

avec Q' c Q, p' c p et les conditions ci-dessus sur f''_0 et f''_S, à savoir :

$$\begin{array}{lcl} f''_0 & \mathsf{c} & f_0 \\ f''_S & = & ([\_ : \mathrm{Set}]f'''_S\ \mathrm{Nat}) \\ f'''_S & = & [n : \mathrm{Nat}][p : (Q\ n)](f'_S\ n\ p) \\ f'_S & \mathsf{c} & f_S \end{array}$$

Alors

$$U\ \mathbf{B}_\iota\ ([n : \mathrm{Nat}](f''_S\ n\ (\mathsf{Fun\_Elim}(\mathrm{Nat}, Q', [f'_0; f'_S])\ n))\ p')$$

or

$$\begin{array}{ll} & ([n : \mathrm{Nat}](f''_S\ n\ (\mathsf{Fun\_Elim}(\mathrm{Nat}, Q', [f'_0; f'_S])\ n))\ p') \\ = & ([n : \mathrm{Nat}]([\_ : \mathrm{Set}](f'''_S\ n\ (\mathsf{Fun\_Elim}(\mathrm{Nat}, Q', [f'_0; f'_S])\ n))\ \mathrm{Nat})\ p') \\ \mathbf{B}_\beta & ([n : \mathrm{Nat}]([\_ : \mathrm{Set}](f'_S\ n\ F)\ \mathrm{Nat})\ p') \end{array}$$

où F est un terme qui code bien $(\mathsf{Fun\_Elim}(\mathrm{Nat}, Q', [f'_0; f'_S])\ n)$. ■

Corollaire 4.12 *Si U_1 c U_2 et $\|U_1\|$ est fortement $\beta\eta\iota_0$-normalisable, alors U_2 est fortement normalisable.*

PREUVE Encore une fois, le lemme de retard de η nous permet de dire qu'il suffit de vérifier que U_2 est fortement $\beta\iota$-normalisable. Le lemme précédent nous permet de coder toutes les séquences de $\beta\iota$-réductions sur U_2 par des séquences de $\beta\iota_0$-réductions au moins aussi longues sur $\|U_1\|$. ■

Il ne nous reste alors plus qu'à vérifier qu'il existe bien un codage vérifiant les caractéristiques ci-dessus. C'est à peu près immédiat. De fait c est presque une définition du codage. On dispose juste d'une liberté supplémentaire lors du traitement des séquences de λ-abstractions, et il faut vérifier la sorte des types de chacune de ces abstractions.

Lemme 4.63 *Soit les jugements dérivables $\Gamma \vdash t : T$, $\Gamma \vdash T : K$, $\Gamma \vdash K : \mathrm{Type}$. Alors il éxiste des termes $\underline{t}$, $\underline{T}$, $\underline{K}$ et $\underline{K}_K$ tel que l'on puisse dériver : $[o, \mathrm{Set}]\Gamma \vdash \underline{t} : \underline{T}$, $[o, \mathrm{Set}]\Gamma \vdash \underline{T} : \underline{K}$, $[o, \mathrm{Set}]\Gamma \vdash \underline{K}_K : \mathrm{Set}$.*

PREUVE Respectivement par récurrence sur les structures de t, T et K. ■

Théorème 11 (Normalisation forte) *Tout terme bien typé est fortement normalisable.*

Chapitre 5

La propriété de Church-Rosser

A partir du résultat de normalisation forte, nous allons prouver la propriété de confluence pour les termes bien-typés. Tous les termes étant normalisables, ce résultat se confond avec l'unicité de la forme normale. Notre preuve est très inspirée de celle de Geuvers [54], mais nous semble plus simple dans le détail. Ensuite nous envisagerons des méthodes alternatives.

5.1 Remarques préliminaires

Strictement parlant, la propriété de confluence correspond à la fermeture du diagramme bien connu ; deux réduits d'un même terme ont un réduit commun. Nous avons vu que lorsque l'on travaille avec des annotations de types sur les λ-abstractions, cette propriété devient fausse. Nous avons annoncé que nous démontrerons cette propriété pour les termes bien-typés ; c'est-à-dire : si $\Gamma \vdash A : B$ est dérivable, et si $A\ \mathbf{B}^*\ A'$ et $A\ \mathbf{B}^*\ A''$, alors il existe A''' tel que $A'\ \mathbf{B}^*\ A'''$ et $A''\ \mathbf{B}^*\ A'''$. Nous appellons cette propriété $\mathbf{CON}_{\beta\eta\iota}$[1]. La propriété de Church-Rosser est plus forte, et se formule ainsi : Si $\Gamma \vdash A : B$ et $\Gamma \vdash A' : B$ sont dérivables, et si $A =_{\beta\eta\iota} A'$, alors il existe A'' réduit commun de A et A' : $A\ \mathbf{B}^*\ A''$ et $A'\ \mathbf{B}^*\ A''$. Nous appellerons cette propriété $\mathbf{CR}_{\beta\eta\iota}$.

Dans le cadre de systèmes comme le λ-calcul pur, où la confluence est vérifiée pour tous les termes, indépendemment de tout jugement de bonne-formation, on montre facilement que $\mathbf{CON}_{\beta\eta\iota}$ et $\mathbf{CR}_{\beta\eta\iota}$ sont équivalents. En revanche, lorsque, comme ici, la propriété $\mathbf{CON}_{\beta\eta\iota}$ n'est vraie que sur une partie des termes, on ne peut plus montrer directement que $\mathbf{CON}_{\beta\eta\iota}$ implique $\mathbf{CR}_{\beta\eta\iota}$ (la reciproque reste facile). Dans la pratique, $\mathbf{CR}_{\beta\eta\iota}$ est au moins aussi importante que $\mathbf{CON}_{\beta\eta\iota}$, car c'est $\mathbf{CR}_{\beta\eta\iota}$ qui assure le résultat essentiel de *décidabilité du typage*. En effet, si $\Gamma \vdash A : B$ et $\Gamma \vdash A : B'$, on sait que $B =_{\beta\eta\iota} B'$, et nous avons besoin de $\mathbf{CR}_{\beta\eta\iota}$ pour assurer que B et B' ont un réduit commun, c'est-à-dire ont même forme normale.

Nous verrons tout à l'heure qu'il existe des présentations alternatives du système de types, où l'équivalence entre ces deux propriétés se montre plus facilement, sans utiliser le résultat de normalisation forte mais au prix d'une mise en œuvre plus coûteuse sur machine.

[1] Attention au risque de confusion : dans [54], Geuvers semble inverser les appellations de confluence et Church-Rosser.

5.2 La preuve de confluence

Il y a deux idées essentielles dans [54]. La première est le lemme de confluence restreinte 2.19 qui permet de faire le début de la métathéorie (unicité de type, correction des réductions vis-à-vis du typage) pour un système non-confluent. La seconde est de remarquer que si un terme bien typé A est en forme β-normale, alors on peut effectuer les mêmes η-réductions dans A que dans $\|A\|$. C'est-à-dire qu'il suffit d'être en forme β-normale (dans CCI, en forme $\beta\iota$-normale) pour effacer le maximum de variables libres. C'est ce résultat qui suit.

Lemme 5.1 *Soit un jugement dérivable* $\Gamma :: (z, U)\Gamma' \vdash A : B$ *avec* A *en* $\beta\iota$*-forme normale. Si*

$$z \notin FV(B) \cup FV(\Gamma') \cup FV(\|A\|)$$

alors $z \notin FV(A)$.

PREUVE A nouveau, on procède par récurrence sur la taille de A. Comme A est, en particulier, en forme $\beta\iota$-normale, on peut utiliser le lemme 2.37.

- Les cas $A = \alpha$, $A = \mathrm{Set}$ et $A = \mathrm{Type}$ sont immédiats.
- Si $A = (x : A_1)A_2$, on sait que $\Gamma :: (z, U)\Gamma' \vdash A_1 : s_1$, et donc $z \notin FV(A_1)$; on peut alors appliquer l'hypothèse de récurrence au jugement $\Gamma :: (z, U)\Gamma' :: (x, A_1) \vdash A_2 : s_2$ et en déduire $z \notin FV(A_2)$.
- Si $A = [x : A_1]A_2$, le raisonnement est similaire : on sait que $\Gamma :: (z, U)\Gamma' \vdash A_1 : s_1$ et donc $z \notin FV(A_1)$. Par ailleurs on sait que $B\ \mathbf{B}^*\ (x : A'_1)B'$, avec $A_1 =_{\beta\eta\iota} A'_1$ et $\Gamma :: (z, U)\Gamma' :: (x, A_1) \vdash A_2 : B'$. Comme $z \notin FV(B') \cup FV(A_1)$, on peut appliquer l'hypothèse de récurrence et donc $z \notin FV(A_2)$.
- Si $A = (x\ \vec{A_i})$, on remarque tout de suite que $x \neq z$. Par ailleurs, si C est le type de x dans $\Gamma :: (z, U)\Gamma'$, on a $C =_{\beta\eta\iota} (y : D)E$. D'après le lemme 2.32 et le lemme de Geuvers, on peut prendre $x \notin FV((y : D)E)$. On a donc $\Gamma :: (z, U)\Gamma' \vdash A_1 : D$, et par hypothèse de récurrence on en déduit $x \notin FV(A_1)$. On en déduit $x \notin E[x \setminus A_1]$; en itérant le raisonnement on vérifie ainsi que $x \notin FV(A_2), \ldots, x \notin FV(A_i)$ [2]
- Si $A = (\mathsf{Ind}(X : E)\{\overrightarrow{C_i}\}\ \vec{D})$, on applique d'abord l'hypothèse de récurrence au jugement $\Gamma :: (z, U)\Gamma' \vdash E : \mathrm{Type}$ pour vérifier que $z \notin FV(E)$. A partir de là, comme pour tout i, $\Gamma :: (z, U)\Gamma' :: (X, E) \vdash C_i : \mathrm{Set}$ on a $z \notin FV(C_i)$. Enfin on suit le même raisonnement que ci-dessus pour vérifier que $z \notin FV(\vec{D})$.
- Le cas $A = (\mathsf{Constr}(i, I)\ \vec{C})$ est à peu près identique au précédent. On montre comme ci-dessus que $z \notin FV(I)$, en remarquant que $I = \mathsf{Ind}(X : E)\{\overrightarrow{D_j}\}$. Ensuite, comme $z \notin FV(D_i)$, on a aussi $z \notin FV(D_i[X \setminus I])$ qui est le type de $\mathsf{Constr}(i, I)$. Enfin on vérifie que $z \notin FV(\vec{C})$ de la même manière que précédemment.
- Si $A = (\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{F_k}\}\ \vec{D})$, on commence par raisonner comme ci-dessus pour vérifier $z \notin FV(I)$. Puis on applique l'hypothèse de récurrence au jugement

  $$\Gamma :: (z, U)\Gamma' \vdash Q : (\vec{x} : \vec{E})(I\ \vec{x}) \to s$$

  où $I \equiv \mathsf{Ind}(X : (\vec{x} : \vec{E})\mathrm{Set})\{\overrightarrow{C}\}$, ce qui montre $z \notin FV(Q)$.

[2] A noter que, comme le système est polymorphe, on n'a pas nécessairement $x : (\vec{y_i} : \vec{D_j})E$ comme on pourrait le croire à première vue. Cette remarque est due à Gilles Dowek.

On peut alors dériver le jugement

$$\Gamma :: (z, U)\Gamma' :: (Y, (\vec{x} : \vec{E})(I\ \vec{x}) \rightarrow \text{Set}) \vdash (Y\ \vec{u}\ t) : \text{Set}$$

et comme la taille de $(Y\ \vec{u}\ t)$ est strictement plus petite que celle de A, l'hypothèse de récurrence assure $z \notin FV(\vec{u}) \cup FV(t)$.
Enfin, si C_k est le type du k-ième constructeur de I, comme $z \notin FV(C_k)$, on a aussi $z \notin FV(\Delta\{C_k, Q, \mathsf{Constr}(k, I)\})$, et comme c'est le type de F_k, on a aussi $z \notin FV(F_k)$. Finalement, en reprenant le même raisonnement que dans les cas précédents, on montre que $z \notin FV(\vec{D})$.

■

Corollaire 5.1 *Soit un jugement dérivable $\Gamma \vdash A : B$. Si A est en forme normale, alors $\|A\|$ l'est aussi.*

Bien sûr, comme A est $\beta\iota$-normal, alors $\|A\|$ l'est aussi. Le point important est que si de plus A est η-normal, alors $\|A\|$ l'est aussi ; c'est-à-dire que si un sous terme de A est de la forme $[x : C](D\ x)$ avec $x \notin FV(\|D\|)$, alors $x \notin FV(D)$.
PREUVE Il nous suffit de vérifier qu'il n'y a pas de η-radicaux dans $\|A\|$. Supposons que ce soit le cas, un sous-terme de A étant de la forme $[x : C](D\ x)$. Ce terme est bien-typé et il existe donc Γ' et E tel que :

$$\Gamma' \vdash [x : C](D\ x) : (x : C)E$$

comme x n'est pas libre dans $[x : C](D\ x)$ on peut supposer que x n'apparaîsse pas dans Γ'. On montre maintenant par récurrence sur la structure de D que x n'est pas libre dans les domaines des λ-abstractions apparaissant dans D.

- Si $D = [y : U]V$, on sait que $\Gamma' \vdash U : s$. Comme U est en forme normale, le lemme précédent assure que $x \notin FV(U)$. On peut alors appliquer l'hypothèse de récurrence à V qui est bien-typé dans le contexte $\Gamma' :: (y, U)$.
- Le cas $D = (x : U)V$ est similaire.
- Si $D = \mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$, on applique le lemme précédent aux jugements $\Gamma' \vdash A : \text{Type}$ puis $\Gamma' :: (X, A) \vdash C_i(X) : \text{Set}$.
- Tous les autres cas sont triviaux, puisqu'ils ne correspondent pas à la liaison d'une variable. Par exemple, si $D = (U\ V)$, les termes U et V sont tous deux normaux et bien typés dans le contexte Γ'. Il suffit de leur appliquer l'hypothèse de récurrence.

■

Church-Rosser

On peut maintenant terminer la preuve de Church-Rosser. Quand on considère deux termes A et A' normaux, de même type et convertibles, on sait d'après ce qui précède, que $\|A\|$ et $\|A'\|$ sont normaux et donc égaux. Il reste donc, en substance, à vérifier que les annotations de type dans les λ-abstractions sont égales, ce qui se fait par récurrence sur la structure des termes, en utilisant l'unicité du type. Les cas correspondants aux types inductifs demandent un petit peu d'astuce.

Lemme 5.2 *Soient deux jugements dérivables $\Gamma \vdash A : B$ et $\Gamma \vdash A' : B$. Si $A =_{\beta\eta\iota} A'$, et si A et A' sont en forme normale, alors $A = A'$.*

PREUVE On sait que $\|A\| =_{\beta\eta\iota} \|A'\|$. Comme, d'après le corollaire précédent $\|A\|$ et $\|A'\|$ sont normaux, et d'après la propriété de Church-Rosser sur $\|\mathcal{L}\|$, on a $\|A\| = \|A'\|$. La preuve se fait alors par récurrence sur les structures de A et A'.

- Le cas $A = A' = s$ est trivial.
- Si $A = [x : C]D$ et $A' = [x : C']D'$, le lemme d'inversion assure que $B =_{\beta\eta\iota} (x : C)E =_{\beta\eta\iota} (x : C')E'$ ce qui implique $C =_{\beta\eta\iota} C'$ et donc $C = C'$. On applique alors l'hypothèse de récurrence à D et D' qui sont bien formés du même type dans le contexte $\Gamma :: (x, C)$.
- Si $A = (x : C)D$ et $A' = (x : D')C'$, on applique d'abord l'hypothèse de récurrence à C et C', puis à D et D' qui sont bien formés du même type dans le contexte $\Gamma :: (x, C)$.
- Si $A = (x\ \vec{C_i})$ et $A' = (x\ \vec{C_i}')$, on sait que la forme normale de $\Gamma(x)$ est de la forme $(\vec{y_i} : \vec{D_i})E$ avec
$$\Gamma \vdash \vec{C_i} : (\vec{y_i} : \vec{D_i}) \text{ et } \Gamma \vdash \vec{C_i}' : (\vec{y_i} : \vec{D_i}).$$
On vérifie alors que pour tout i, $C_i = C'_i$ en appliquant i fois l'hypothèse de récurrence à C_1 et C'_1, puis C_2 et C'_2, etc, qui sont respectivement bien formés de même type dans les contextes Γ, $\Gamma :: (y_1, D_1)$, etc.
- Si $A = (\mathsf{Ind}(X : D)\{\overrightarrow{D}\}\ \vec{E})$ et $A' = (\mathsf{Ind}(X : D')\{\overrightarrow{D'}\}\ \vec{E'})$, on fait de même, après avoir vérifié que $D = D'$ car ils sont bien typés de type Type dans Γ, puis que $\vec{C} = \vec{C'}$, puisque les composantes sont bien formées de type Set dans $\Gamma :: (X, D)$.
- Si $A = (\mathsf{Constr}(i, I)\ \vec{C})$ et $A' = (\mathsf{Constr}(i, I')\ \vec{C'})$, on vérifie comme dans le cas précédent que $I = I'$, puis on montre, comme dans les deux cas précédents que $\vec{C} = \vec{C'}$.
- Si $A = \mathsf{Elim}(I, Q, \vec{u}, C)\{\overrightarrow{F}\}$ et $A' = \mathsf{Elim}(I', Q', \vec{u'}, C')\{\overrightarrow{F'}\}$, on montre d'abord que $I = I'$, comme ci-dessus.
Soit ensuite $s \in \{\text{Set}, \text{Type}\}$, le type de B. Si
$$I = \mathsf{Ind}(X : (\vec{x} : \vec{E})\text{Set})\{\overrightarrow{D}\}$$
on peut appliquer l'hypothèse de récurrence aux jugements
$$\Gamma \vdash Q : (\vec{x} : \vec{E})(I\ \vec{x}) \to s$$
et
$$\Gamma \vdash Q' : (\vec{x} : \vec{E})(I\ \vec{x}) \to s$$
et ainsi vérifier $Q = Q'$.
On remarque ensuite que
$$\Gamma :: (Y, (\vec{x} : \vec{E})(I\ \vec{x}) \to \text{Set}) \vdash (Y\ \vec{u}\ C)$$
et
$$\Gamma :: (Y, (\vec{x} : \vec{E})(I\ \vec{x}) \to \text{Set}) \vdash (Y\ \vec{u'}\ C').$$
On peut appliquer l'hypothèse de récurrence à ces jugements, et ainsi vérifier $(Y\ \vec{u}\ t) = (Y\ \vec{u'}\ t')$ ce qui implique $\vec{u} = \vec{u'}$ et $C = C'$.
Enfin $\vec{F} = \vec{F'}$ se montre de la même manière que dans les cas précédents.

■

Théorème 12 (Church-Rosser) *Soient deux jugements dérivables : $\Gamma \vdash A : B$ et $\Gamma \vdash A' : B$. Si $A =_{\beta\eta\iota} A'$, alors A et A' ont un réduit commun : il existe un terme C tel que $A\ \mathbf{B}^*\ C$ et $A'\ \mathbf{B}^*\ C$.*

PREUVE C'est une conséquence immédiate du lemme précédent : soient D et D' deux formes normales respectives de A et A'. La correction de la réduction nous assure $\Gamma \vdash D : B$ et $\Gamma \vdash D' : B$, et donc $D = D'$. Il suffit donc de choisir $C = D = D'$. ■

Corollaire 5.2 (Confluence) *Soit un jugement dérivable $\Gamma \vdash A : B$. Si A $\mathbf{B}^*$ A_1 et A $\mathbf{B}^*$ A_2, alors il existe un terme A_3 tel que A_1 $\mathbf{B}^*$ A_3 et A_2 $\mathbf{B}^*$ A_3.*

5.3 Décidabilité du Typage

Nous pouvons maintenant cueillir un autre fruit de notre étude méta-théorique : il est possible d'implémenter un algorithme complet vérifiant si un terme est bien formé dans un contexte donné. Les clés sont bien sûr les propriétés de normalisation et de Church-Rosser, qui nous permettent de vérifier si deux termes bien-formés sont convertibles et le lemme d'inversion qui relie la structure d'une dérivation à celle du terme typé.

Théorème 13 (Décidabilité) *Il existe un algorithme, qui, donné un terme U et un contexte Γ,*
- *soit échoue s'il n'existe pas de terme V tel que $\Gamma \vdash U : V$ soit dérivable*
- *soit retourne un terme V tel que $\Gamma \vdash U : V$ soit dérivable.*

PREUVE On raisonne par récurrence sur la somme des tailles de U et Γ. Dans chaque cas, on utilise la clause correspondante du lemme 2.28 d'inversion. On ne détaille pas tous les cas.

- Si $U = [x : U_1]U_2$ on vérifie s'il existe des jugements dérivables de formes suivantes :

$$\Gamma \vdash U_1 : V_1$$
$$\Gamma :: (x, U_1) \vdash U_2 : V_2.$$

  On vérifie ensuite que $V_1 \neq$ Extern. Sous ces conditions, il existe une dérivation de $\Gamma \vdash U : (x : U_1)V_2$; si en revanche l'une de ces conditions n'est pas vérifiée, alors le jugement n'est pas dérivable (inversion).
- Si $U = (U_1\ U_2)$, on vérifie s'il existe des jugements dérivables de formes suivantes :

$$\Gamma \vdash U_1 : V_1$$
$$\Gamma \vdash U_2 : V_2.$$

  On vérifie ensuite que la forme normale de V_1 est bien de la forme $(x : V_2')V_3$ et que $V_2 =_{\beta\eta\iota} V_2'$. Si c'est le cas, les règles CONV et APP permettent de construire une dérivation de $\Gamma \vdash U : V_3[x \setminus U_2]$; si en revanche l'une de ces conditions n'est pas vérifiée, alors aucun jugement $\Gamma \vdash U : V$ n'est dérivable.
- Les cas correspondants aux définitions inductives ($\mathsf{Ind}(X : A)\{\overrightarrow{C_i(X)}\}$, $\mathsf{Elim}(I, Q, \vec{u}, t)\{\overrightarrow{U_i}\}$, $\mathsf{Constr}(i, I)$) sont similaires aux deux prédedents.
- Si $U =$ Set, il suffit de prendre $V \equiv$ Type, avec comme condition nécéssaire et suffisante que Γ soit bien-formé. Pour cela, si $\Gamma = (\vec{x_i} : \vec{U_i})$ on commence par vérifier que U_1 admet bien un type dans le contexte vide, et que celui-ci se réduit vers Set (respectivement Type) si x_1 est une variable de terme (respectivement de prédicat). On réitère ensuite pour U_2 dans le contexte $[x_1, U_1]$ et ainsi de suite. L'unicité du type et la confluence, ainsi que le lemme 2.24, assurent que ces conditions sont suffisantes.

– Si $U = \text{Type}$ on prend $V = \text{Extern}$ après avoir vérifié, comme ci-dessus, que Γ est bien-formé.
– Si U est une variable, on vérifie que x est bien liée dans Γ, auquel cas on prend $V \equiv \Gamma(x)$ après avoir vérifié que Γ est bien-formé.

■

Remarque Il nous faut noter un point important : il est possible, en utilisant les divers résultats prouvés précédemment, de montrer que le système que nous avons étudié est en fait équivalent à celui obtenu en supprimant la distinction entre variables de preuves et de prédicats. Plus précisément on identifie ces deux classes syntaxiques, et on remplace les règles W-Type et W-Set par :

$$\boxed{(\text{W})\ \frac{\Gamma \vdash t : s \qquad \Gamma \vdash A : B \qquad s \in \{\text{Set}, \text{Type}\} \qquad x \notin \Gamma}{\Gamma :: (x, t) \vdash A : B}}$$

Les autres règles de typage et les définitions des réductions restent inchangés. On prouve alors aisément, par une récurrence similaire à celle ci-dessus, que le système obtenu est équivalent à celui que nous avons étudié, et hérite donc de toutes ses propriétés (normalisation, Church-Rosser, cohérence, décidabilité) et peut donc servir de base à une implémentation.

5.4 Discussion

On peut imaginer prouver la propriété $\mathbf{CON}_{\beta\eta\iota}$ avec des méthodes plus familières, inspirées par exemple de la technique de Tait–Martin-Löf. En revanche, on voit mal comment démontrer $\mathbf{CR}_{\beta\eta\iota}$ sans passer par l'existence d'une forme normale. Je voudrais indiquer ici quelques méthodes susceptibles de s'appliquer aussi au cas de systèmes non-normalisants. Remarquons que tout ce qui va être dit dans ce paragraphe s'applique aussi bien au Calcul des Constructions Inductives qu'à tout système de types pur (PTS) fonctionnel [56, 7].

5.4.1 Abstractions non-annotées

Une première idée est de contourner le contre-exemple de Nederpelt en supprimant tout simplement les annotations de types dans les λ-abstractions ; on remplace le constructeur de termes $[x : t_2]t_1$ par $\lambda x.t_1$. Les autres constructeurs de termes restent inchangés. La re-formulation des règles de réduction et de typage ne pose aucun problème, et on a immédiatement la propriété de Church-Rosser, même pour les termes mal-typés. Le point important est que ce système est équivalent à celui que nous étudions, à cause de la propriété

$$A =_{\beta\eta\iota} A' \Leftrightarrow \|A\| =_{\beta\eta\iota} \|A'\|.$$

Précisons ce point :

Définition 5.1 *On considère le système obtenu en gardant les même règles de typage et de réduction que dans la version de CCI étudiée dans les chapitres précédents, mais en utilisant les termes définis par la grammaire suivante* [3] *:*

$$t \quad := \quad x \mid (t\ t) \mid \lambda x.t \mid s \mid \mathsf{Constr}(i, t) \mid \mathsf{Ind}(x : t)\{\overrightarrow{t}\} \mid \mathsf{Elim}(t, t, \vec{t}, t)\{\overrightarrow{t}\}.$$

[3] Pour être précis, on remplace la ι-réduction par quelque chose de similaire à la ι_0-réduction utilisée dans la section 2.9.1. Enoncer dans le détail la définition de ce système ne présente pas de grand intérêt et nous semble inutile.

On note $\Gamma \vdash_p U : V$ *pour désigner les jugements dérivables dans ce système (donc* U *et* V *sont des termes avec* λ*-abstractions non-annotées). Il existe évidemment une traduction canonique des termes annotés vers les termes utilisés ici. Par léger abus de notation on utilise la même notation* $\| \ \|$ *que dans la section 2.9.1 pour la désigner (c'est-à-dire en particulier* $\|[x : T]t\| \equiv \lambda x.\|t\|$*). On étend également cette traduction aux contextes, en parlant de* $\|\Gamma\|$*. Nous appellerons ce système* CCI_p*.*

Le résultat d'équivalence s'énonce alors :

Théorème 14 *Si un jugement* $\Gamma \vdash A : B$ *est dérivable dans CCI, alors on peut aussi dériver* $\|\Gamma\| \vdash_p \|A\| : \|B\|$ *dans* CCI_p*. Inversement, pour toute dérivation* $\Delta \vdash_p U : V$ *dans* CCI_p*, il existe une dérivation d'un jugement* $\Gamma \vdash A : B$ *dans CCI, tel que* $\|\Gamma\| = \Delta$*,* $\|A\| = U$ *et* $\|B\| = V$*.*

Preuve Les deux implications se prouvent aisément par récurrence sur la structure des dérivations. Pour la seconde, dans le cas de la règle de conversion, on utilise évidemment le fait

$$t =_{\beta\eta\iota} t' \iff \|t\| =_{\beta\eta\iota} \|t'\|.$$

■

Si la première implication est attendue, la seconde est, à première vue, plus surprenante. De plus, la preuve est constructive, et permet, à partir de la dérivation de $\Delta \vdash_p U : V$ de construire effectivement les termes A, B, le contexte Γ et la dérivation de $\Gamma \vdash A : B$. Cet isomorphisme entre les dérivations des deux systèmes semble indiquer qu'il est préférable d'utiliser des abstractions non-typées. Le problème vient du fait que pour passer de $\vdash_p$ à $\vdash$, nous avons besoin de connaître *toute la dérivation*. En d'autres termes, étant donné un jugement $\Delta \vdash_p U : V$, si nous ne savons pas comment il a été dérivé, et même si nous savons qu'il est dérivable, nous ne pouvons construire le jugement correspondant dans CCI. En fait il peut très bien exister plusieurs jugements différents dans CCI, dont la traduction est $\Delta \vdash_p A : B$.

De fait, le typage du système CCI_p semble indécidable et la correction de la η-réduction est problématique, ce qui disqualifie ce système comme base d'une implémentation et anihilie à peu près l'avantage que la propriété de Church-Rosser pourrait apporter lors de la preuve de normalisation[4].

5.4.2 Conversions bien-typées

Une autre possibilité est de remarquer que si $\mathbf{CR}_{\beta\eta\iota}$ et $\mathbf{CON}_{\beta\eta\iota}$ ne sont pas *a priori* équivalents, c'est parce que, si A et A' sont bien formés du même type, et si $A =_{\beta\eta\iota} A'$, on n'est pas sûr que le chemin allant de A à A' ne passe que par des termes bien typés. C'est cela qui nous interdit d'invoquer $\mathbf{CON}_{\beta\eta\iota}$ pour prouver l'existence d'un réduit commun de A et A'. Plus important encore, c'est le fait que les conversions, peuvent *a priori* passer par des termes mal-typés, qui nous impose de définir les interprétations des prédicats même pour des termes mal-typés, et qui contribue ainsi pour beaucoup à la lourdeur de la preuve de normalisation (en particulier la nécessité de définir les prédicats invariants sur leur domaine). On peut y remédier en imposant cette condition au niveau des règles de typage : on supprime la règle de conversion :

$$(\text{Conv}) \ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_2 : S \qquad T_1 =_{\beta\eta\iota} T_2}{\Gamma \vdash t : T_2}$$

[4]Le problème avec la η-réduction est que l'effacement des annotations de type crée une nouvelle fois des η-radicaux. Il est possible que la η-réduction préserve toutefois le type, mais si cette propriété était vraie, elle dépendrait de toute façon fortement de la normalisation. Elle est par exemple fausse pour le système $\lambda - *$ étudié en appendice, si on y retire les annotations de type.

et on la remplace par les deux règles suivantes :

$$(\textsc{Red})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_2 : S \qquad T_1 \mathbf{B}\, T_2}{\Gamma \vdash t : T_2} \qquad (\textsc{Exp})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_2 : S \qquad T_2 \mathbf{B}\, T_1}{\Gamma \vdash t : T_2}$$

l'idée étant bien sûr qu'une utilisation de la règle Conv sera remplacée par une série d'utilisations des règles Red et Exp. De plus, cette formulation est plus rassurante à première vue, car il semble étrange d'autoriser d'abord le passage par des termes mal typés, pour vérifier in fine, une fois que la méta-théorie est faite, qu'en fait on ne se sert pas de cette possibilité. Il existe d'autres présentations, où l'on utilise un jugement d'égalité explicite $\Gamma \vdash t_1 = t_2 : T$ pour signifier qu'il existe un chemin de conversion bien-typé de type T allant de t_1 à t_2 dans le contexte Γ. Streicher [138] et Altenkirch [1] utilisent tous deux de tels formalismes ; il est facile de prouver que ces derniers sont équivalents à celui ci-dessus.

Malheureusement, et c'est un peu surprenant, cette variante du système se révèle beaucoup plus difficile à étudier, au moins pour les propriétés autres que la normalisation. On arrive aisément à prouver le lemme de substitution, puis le lemme d'inversion. Plus précisément on prouve un lemme d'inversion renforcé dans l'énoncé duquel toutes les occurrences de $A =_{\beta\eta\iota} B$ se voient remplacées par "on peut aller de A vers B par un chemin bien-typé". On peut également montrer l'unicité du typage sous la forme :

Lemme 5.3 *Soient $\Gamma \vdash A : B$ et $\Gamma \vdash A : B'$ deux jugements dérivables dans le système avec les règles* Red *et* Exp. *Alors il existe une séquence de termes $A_1, A_2, \ldots A_n$, avec $A = A_1$, $B = A_n$, et pour tout i, $\Gamma \vdash A_i : s$ et $A_i \mathbf{B}\, A_{i+1}$ ou $A_{i+1} \mathbf{B}\, A_i$.*

C'est ensuite que les choses se gâtent, car il ne semble pas y avoir, à l'heure actuelle, de preuve de correction de la réduction *(subject reduction)* pour un tel système[5]. C'est pour avoir négligé ce détail, que Harper, Honsell et Plotkin ont dû renoncer à la η-conversion dans la définition du système LF [132], alors qu'ils avaient inclus celle-ci dans leur première version [131].

On peut toutefois étudier ce système en se restreignant dès le départ aux réductions bien-typées (c'est-à-dire $t \mathbf{B}\, t'$ où t' est de même type que t). C'est, en substance, ce que fait Altenkirch [1], et on obtient ainsi une preuve plus agréable. Le problème est alors que pour vérifier l'équivalence entre les deux systèmes (le notre et celui avec Red et Exp), on a besoin de la métathéorie de notre système. Or pour des raisons d'efficacité, c'est évidemment notre système que l'on veut ultimement utiliser dans une implémentation.

Il y a diverses variantes de systèmes, plus ou moins équivalents à celui ci-dessus. On peut, par exemple mentionner également une autre variante de la règle de conversion, dont l'utilisation est équivalente aux règles Red et Exp :

$$(\textsc{Conv-WT})\ \frac{\Gamma \vdash t : T_1 \qquad \Gamma \vdash T_2 : S \qquad T_1 \downarrow T_2}{\Gamma \vdash t : T_2}$$

où $T_1 \downarrow T_2$ signifie que T_1 et T_2 ont un réduit commun.

On peut enfin utiliser les règles de conversion typées pour un système sans annotations de type dans les λ-abstractions, comme décrit dans le paragraphe suivant. Mais comme on dispose alors de la confluence "gratuitement" cette modification est alors sans grandes conséquences.

[5] Plus exactement, la seule méthode connue est de faire la méta-théorie pour le système avec conversion non-typée, comme nous le considérons ici, et de prouver ensuite l'équivalence entre les deux systèmes. Cette dernière preuve n'est pas excessivement difficile bien que non triviale. Elle sera donnée dans [57].

Toutes les variantes présentées ci-dessus sont équivalentes, pourvu que l'on soit dans un système fortement normalisable, c'est-à-dire en particulier dans le cas du Calcul des Constructions Inductives.

Le problème de trouver une preuve purement combinatoire de la propriété de confluence qui s'appliquerait aussi aux systèmes non normalisables reste ouvert. Il est conjecturé vrai par Geuvers. Sans prendre parti, nous montrons néanmoins en annexe que ce problème est lié à un autre problème ouvert, celui de l'existence d'un point fixe dans les systèmes non-normalisables, et que la confluence est fausse pour une extension "naturelle" du système Type :Type de Martin-Löf. Ceci indique que même si l'on obtenait une preuve de confluence combinatoire, pour les PTS non-normalisables, cette preuve ne serait de toute façon pas extensible à des extensions de ces systèmes, nous obligeant à emprunter le chemin le plus dur dans l'étude méta-théorique.

Annexe A

Sur Church-Rosser et l'Existence d'un Point-Fixe dans Type :Type

Dans cette partie, nous relions, de manière élémentaire deux problèmes ouverts :

- Le premier est la possibilité de construire un point fixe dans le premier système de Martin-Löf [96], dont Girard a mis en évidence l'incohérence [25]. Ce système est souvent aussi appellé "Type :Type", ou $\lambda - *$ dans la terminologie PTS. La possibilité de construire une preuve de la proposition fausse implique que cette preuve ne soit pas normalisable. Meyer et Reinhold ont cru pouvoir aller plus loin et utiliser la preuve du paradoxe de Girard pour construire un point fixe dans le système, mais leur preuve c'est ensuite avérée erronée. Doug Howe [72], puis Thierry Coquand et Hugo Herbelin [22], ont ensuite montré que l'on pouvait en revanche construire un "looping combinator", forme plus faible du point fixe. L'existence du point fixe reste cependant un problème ouvert.
- Nous relions ce problème avec celui de la confluence de $\lambda - *$ muni de la $\beta\eta$-réduction. Nous avons vu que la preuve de confluence de CC ou CCI nécessite la normalisation forte et que l'on ne sait pas si la confluence des PTS doit être considérée comme une propriété logique ou combinatoire.

Nous montrons maintenant qu'au moins l'une de ces caractéristiques n'est pas vérifiée dans $\lambda - *$. Remarquons que nous considérons la version extentionnelle de $\lambda - *$, c'est-à-dire que la règle de conversion utilise la $\beta\eta$-conversion ; sinon la propriété de Church-Rosser serait évidemment vérifiée.

Définition A.1 *On dit que l'on peut construire un point-fixe dans $\lambda - *$ s'il existe un terme F, tel que $[] \vdash t : (X : *)(X \to X) \to X$ soit dérivable, et que pour tous termes T et f on ait :*

$$(t\ T\ f)\ \mathbf{B}\ (f\ (t\ T\ f)).$$

Dans la suite on suppose l'existence d'un tel terme, et l'on va s'en servir pour nier la propriété de Church-Rosser.

Comme on dispose de la règle $* : *$, on peut utiliser t pour définir un type récursif. On se donne trois variables *distinctes* α, β et γ. On définit alors :

$$\begin{aligned} U &\equiv (t\ *\ [X : *](X \to (\alpha \to \alpha) \to \gamma)) \\ V &\equiv (t\ *\ [X : *](X \to (\beta \to \beta) \to \gamma)) \end{aligned}$$

on a alors :

$$\begin{aligned}
[\alpha : *; \beta : *; \gamma : *] &\vdash U : * \\
[\alpha : *; \beta : *; \gamma : *] &\vdash V : * \\
U &=_{\beta\eta} U \to (\alpha \to \alpha) \to \gamma \\
V &=_{\beta\eta} V \to (\beta \to \beta) \to \gamma
\end{aligned}$$

On peut alors construire :

$$\begin{aligned}
u &\equiv [x : U][i : \alpha \to \alpha](x\ x\ [y : \alpha]y) \\
v &\equiv [x : V][i : \beta \to \beta](x\ x\ [y : \beta]y)
\end{aligned}$$

On vérifie facilement que $u : U$ et $v : V$ dans le même contexte que ci-dessus. De plus, $u =_{\beta\eta} v$, puisque les deux termes ne diffèrent que par les types dans les λ-abstractions (lemme 2.7). On a donc aussi

$$(u\ u\ [y : \alpha]y) =_{\beta\eta} (v\ v\ [y : \beta]y)$$

et on remarque que ces deux termes sont bien-formés du même type γ dans le contexte ci-dessus. Il suffit alors de vérifier que ces deux termes n'admettent pas de réduit commun pour nier Church-Rosser. Or on vérifie aisément que tout réduit de $(u\ u\ [y : \alpha]y)$ est de la forme $(u_1\ u_2\ [y : \alpha]y)$ et de même tout réduit de $(v\ v\ [y : \beta]y)$ est de la forme $(v_1\ v_2\ [y : \beta]y)$. Or de tels termes ne peuvent pas être égaux puisque $\alpha \neq \beta$. On a donc :

Théorème 15 *Au moins l'une de ces deux propositions est fausse :*

- *il existe une construction de point-fixe dans* $\lambda - *$.
- $\lambda - *$ *admet la proprieté de confluence.*

Annexe B

Une Formalisation du Paradoxe de Russell

Nous proposons une formalisation très simple du paradoxe de Russell, à notre connaissance inédite, dans une extension incohérente du Calcul des Constructions Inductives. Celle-ci est obtenue en ajoutant à CCI l'élimination forte sur les "grands" constructeurs, ainsi que l'opérateur d'élimination de l'égalité proposé par Thomas Streicher et présenté plus tôt. En dehors de sa simplicité, notre formalisation présente l'interêt de permettre la construction d'un point fixe, et de prouver ainsi que le formalisme utilisé ne vérifie pas la propriété de Church-Rosser. Cette preuve à été vérifiée mécaniquement à l'aide de la version V5.8 du système Coq, modifiée pour la rendre incohérente.

On sait que l'élimination forte sur les grands constructeurs suffit à rendre le système incohérent [25]. L'opérateur de Streicher permet simplement de construire un paradoxe plus facilement. On ne connaît pas de formalisation du paradoxe de Russell dans CCI+élimination forte sur les grands constructeurs, ni dans le système Type : Type. Aussi les problèmes de l'existence d'un point fixe et de Church-Rosser pour ces systèmes sont toujours ouverts.

Comme dans le chapitre 1, on utilise la syntaxe Coq `Inductive Definition ...` pour décrire les définitions de types inductifs. Le paradoxe de Russell est construit sur l'idée qu'un type permet de coder tous les prédicats sur ce type. Pour ce faire, nous utilisons la définition suivante :

```
Inductive Set Ens = c : (S:Set)(S->Set)->Ens.
```

Remarquons bien sûr que le constructeur c (comme *compréhension*) est grand. Comme nous disposons tout de même de l'élimination forte, nous pouvons construire les projections :
$pi1 : Ens \rightarrow \text{Set}$
$pi2 : (e : Ens)(pi1\ e) \rightarrow \text{Set}$
avec les proprietés suivantes :
$(pi1\ (c\ S\ P))\ \mathbf{B}\ S$
$(pi2\ (c\ S\ P))\ \mathbf{B}\ P$

Nous pouvons donc extraire un prédicat sur les ensembles d'un objet de type Ens, pour peu que nous sachions que sa première projection est égale à Ens. Pour définir l'égalité sur les types, une possibilité est d'utiliser :

```
Inductive Definition encode : Set =
 enc_i : Set->encode.
```

et de définir l'égalité entre les objets $S1$ et S_2 de type Set comme l'égalité entre $(enc\_i\ S_1)$ et $(enc\_i\ S_2)$. L'élimination forte nous permet d'obtenir alors toutes les propriétés habituelles de l'égalité. On peut alors caractériser les objets de type Ens dont la première projection est Ens :

$$\forall e : Ens.(is\_Ens\ e) \Leftrightarrow (pi1e) = Ens.$$

On peut alors, pour les ensembles vérifiant is_Ens définir la relation d'appartenance :
$In : (e : Ens)(is\_Ens\ e) \rightarrow Ens \rightarrow$ Set
avec :
$(In\ (c\ Ens\ P)\ (refl\ (enc\_i\ Ens)))\ \mathbf{B}\ P$
où $refl$ est simplement l'unique constructeur de l'égalité sur *encode*.
Ceci suffit à construire l'ensemble paradoxal de Russell :
$Russell \equiv (c\ Ens\ [e : Ens](p : (is\_Ens\ (pi1\ e)))(In\ e\ p\ e) \rightarrow \bot)$
On a alors :
$(In\ Russell\ (refl\ (enc\_i\ Ens))\ Russell)\ \mathbf{B}$
$(p : (is\_Ens\ (enc\_i\ Ens)))(In\ Russell\ p\ Russell) \rightarrow \bot$
dont on déduit facilement une preuve de

$$(In\ Russell\ (refl\ (enc\_i\ Ens))\ Russell) \rightarrow \bot.$$

Il reste alors à en déduire que $Russell$ est bien élément de lui-même, ce qui, on l'a vu est convertible à :

$$(p : (is\_Ens\ (enc\_i\ Ens)))(In\ Russell\ p\ Russell) \rightarrow \bot$$

mais nous pouvons alors utiliser l'opérateur de Streicher pour prouver que toutes les preuves de $(is\_Ens\ Ens)$ sont égales, et ainsi ramener cela à

$$(In\ Russell\ (refl\ (enc\_i\ Ens))\ Russell) \rightarrow \bot.$$

On a ainsi construit une preuve de $\bot$. La règle de réduction associée à l'opérateur de Streicher, qui n'a pas été nécessaire jusqu'ici, permet alors de vérifier que

$$absurd\ \mathbf{B}^+\ absurd$$

où *absurd* est la preuve de $\bot$. On peut alors aisément modifier *absurd* pour construire un point fixe.

Cette preuve peut-etre traduite immédiatement dans le système $\lambda - *$ (ou Type : Type) étendu avec l'opérateur de Streicher, en utilisant le codage imprédicatif de la somme. On en déduit :

Théorème 16 *Le système $\lambda - *$ étendu avec l'opérateur de Streicher n'est pas confluent pour la $\beta\eta$-réduction.*

Bibliographie

[1] Th. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* Thèse de PhD, Université d'Edinburgh, 1993.

[2] Ph. Audebaud. CC+ : an extension of the Calculus of Constructions with fixpoints. In *[5]*, 1992.

[3] Ph. Audebaud. *Extension du Calcul des Constructions par Points fixes.* Thèse de Doctorat, Université Bordeaux I, 1992.

[4] L. Augustsson et Th. Johnsson. Lazy ML user's manual, preliminary draft. Chalmers University, Sweden, 1989.

[5] K. Petersson B. Nordström et G. Plotkin (rédacteurs). *Proceedings of the 1992 Workshop on Types for Proofs and Programs.* Distribution électronique, 1992.

[6] Modularity of Strong Normalization and Confluence in the Algebraic λ-Cube. Proceedings of the ninth Symposium on Logic in Computer Science, IEEE press, 1994.

[7] H. Barendregt. Lambda Calculi with Types. Technical Report 91-19, Catholic University Nijmegen, 1991. In Handbook of Logic in Computer Science, Vol II, Elsevier, 1992.

[8] H.P. Barendregt. *The Lambda Calculus its Syntax and Semantics.* North-Holland, 1981.

[9] J.L. Bates et R.L. Constable. Proofs as Programs. *ACM transactions on Programming Languages and Systems*, 7, 1985.

[10] M.J. Beeson. *Foundations of Constructive Mathematics, Metamathematical Studies.* Springer-Verlag, 1985.

[11] E. Bishop. *Foundations of Constructive Analysis.* McGraw-Hill, 1967.

[12] C. Böhm et A. Berarducci. Automatic Synthesis of Typed Λ-Programs on Term Algebras. *Theoretical Computer Science*, 39, 1985.

[13] S. Boutin. Certification d'un compilateur ML en Coq. Rapport de dea, Ecole Normale Supèrieure, Ecole Polytechnique, septembre 1992.

[14] R. S. Boyer et J. S. Moore. *A computational logic.* ACM Monograph, Academic Press, 1979.

[15] L. Cardelli. Typeful Programing. Rapport Technique 45, DEC Systems Research Center, 1989. (Voir aussi le manuel QUEST joint à la distribution.)

[16] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5 (1), pp. 56-68, 1940.

[17] A. Church. *The Calculi of Lambda-Conversion.* Princeton University Press, 1941.

[18] L. Colson. About Primitive Recursive Algorithms. In M. Dezani-Ciancaglini G.Ausiello et S. Ronchi Della Rocca (rédacteurs), *Proceedings of the sixteenth International Colloquium on Automata, Languages and Programming.* Springer-Verlag, 1989.

[19] L. Colson. *Représentation intentionnelle d'algorithmes dans les systèmes fonctionnels : une étude de cas.* Thèse de Doctorat, Université Paris 7, Janvier 1991.

[20] L. Colson. On List Primitive Recursion and the Complexity of Computing *inf. BIT*, 1992. Special Issue on Programming Logic.

[21] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[22] T. Coquand et H. Herbelin. An Application of A-translation to the existence of families of looping combinators in inconsistent Type Systems. *Journal of Functional Programming*, 4 (1), pp. 77–88, 1994.

[23] T. Coquand et J. Gallier. A proof of Strong Normalization for the Calculus of Constructions Using a Kripke-Like Interpretation. In [80], 1990.

[24] Th. Coquand. *Une Théorie des Constructions.* Thèse de Doctorat, Université Paris 7, janvier 1985.

[25] Th. Coquand. An Analysis of Girard's Paradox. In *Symposium on Logic in Computer Science*, Cambridge, MA. IEEE press, 1986.

[26] Th. Coquand. Metamathematical Investigations of a Calculus of Constructions. Private communication, 1987.

[27] Th. Coquand. Metamathematical Investigations of a Calculus of Constructions. In P. Oddifredi (rédacteur), *Logic and Computer Science.* Academic Press, 1990. Rapport de recherche INRIA 1088, also in [50].

[28] Th. Coquand. An algorithm for testing type conversion in type theory. In Huet et Plotkin [81].

[29] Th. Coquand. Pattern Matching with Dependent Types. In [109].

[30] Th. Coquand et G. Huet. Constructions : A Higher Order Proof System for Mechanizing Mathematics. In *EUROCAL'85*, Linz, 1985. Springer-Verlag. LNCS 203.

[31] Th. Coquand et G. Huet. Concepts Mathématiques et Informatiques formalisés dans le Calcul des Constructions. In The Paris Logic Group (rédacteur), *Logic Colloquium'85.* North-Holland, 1987.

[32] Th. Coquand et G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3), 1988.

[33] Th. Coquand et C. Paulin-Mohring. Inductively defined types. In P. Martin-Löf et G. Mints (rédacteurs), *Proceedings of Colog'88.* Springer-Verlag, 1990. LNCS 417.

[34] G. Cousineau et G. Huet. The CAML Primer. Rapport Technique INRIA 122, 1990.

[35] N.G. de Bruijn. Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Math.*, 34 (5), pp. 381–392, 1972.

[36] N.G. de Bruijn. A survey of the project Automath. In J.P. Seldin et J.R. Hindley (rédacteurs), *to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[37] G. Dowek. Naming and Scoping in a Mathematical Vernacular. Rapport de Recherche 1283, INRIA, 1990.

[38] G. Dowek. *Démonstration automatique dans le Calcul des Constructions.* Thèse de Doctorat, Université Paris 7, décembre 1991.

[39] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin-Mohring, et B. Werner. The Coq Proof Assistant User's Guide Version 5.6. Rapport Technique 134, INRIA, décembre 1991.

[40] G. Dowek, G. Huet, B. Werner. On the Definition of the η-long Normal Form in Type Systems of the Cube. In Informal Proceedings of the BRA Workshop on Types for Proofs and Programs, H. Geuvers Editeur, Nijmegen, 1993. Diffusion électronique.

[41] P. Dybjer. From Type Theory to LCF-A Case Study in Program Verification. In Dybjer et al. (rédacteurs), *Workshop on Programming Logic*, Marstrand, 1987. University of Göteborg and Chalmers University of Technology. Report 37, Programming Methodology Group.

[42] P. Dybjer. An inversion principle for Martin-Löf's type theory. In *Workshop on Programming Logic*. University of Göteborg and Chalmers University of Technology, 1989. To appear as a report of the Programming Methodology Group.

[43] P. Dybjer. Comparing Integrated and External Logics of Functional Programs. *Science of Computer Programming*, 14 :59–79, 1990.

[44] P. Dybjer. Inductive sets and families in Martin-Löf's Type Theory and their set-theoretic semantics. In G. Huet et G. Plotkin (rédacteurs), *Logical Frameworks*, volume 14, pp. 59–79. Cambridge University Press, 1991.

[45] P. Dybjer. Universes and a General Notion of Simultaneous Inductive-Recursive Definition in Type Theory. In [109], 1992.

[46] P. Dybjer et H. Sander. A functional programming approach to the specification and verification of concurrent systemsnd verification of concurrent systems. *Formal Aspects of Computing*, 1 :303–318, 1989.

[47] P. Weis et al. The CAML Reference Manual. Rapport Technique 121, INRIA, 1990.

[48] E. Fleury. Implantation des algorithmes de Floyd et de Dijkstra dans le Calcul des Constructions. Rapport de Stage, juillet 1990.

[49] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*,1879. Reédité en traduction anglaise dans [148].

[50] G. Huet ed. *The Calculus of Constructions, Documentation and user's guide, Version V4.10.* Rapport technique INRIA 110, 1989.

[51] J. Gallier. On Girard's "Candidats de Réductibilité". In P. Oddifredi (rédacteur), *Logic and Computer Science*, pp. 123–203. Academic Press, 1990. Version complète corrigée en rapport de recherche de l'Université de Pennsylvanie, Philadelphie.

[52] G. Gentzen. *The Collected Work of Gerhard Gentzen.*Édité par M.E. Szabo. North-Holland, 1969.

[53] H. Geuvers. Type systems for Higher Order Logic. Rapport de recherche, Faculty of Mathematics and Informatics, Catholic University Nijmegen, 1990.

[54] H. Geuvers. The Church-Rosser Property for $\beta\eta$-reduction in Typed λ-Calculi. In *Symposium on Logic in Computer Science*, pp. 453–460. IEEE Computer Society Press, 1992.

[55] H. Geuvers. *Logics and Type Systems.* Thèse de Doctorat, Catholic University Nijmegen, 1993.

[56] H. Geuvers et M.-J. Nederhof. A modular proof of strong normalization for the Calculus of Constructions. *Journal of Functional Programming*, 1 (2) :155–189, 1991. Rapport de Recherche, Faculty of Mathematics and Informatics, Catholic University Nijmegen, 1989.

[57] H. Geuvers et B. Werner. On the Church-Rosser Property for Expressive Type Systems and its Consequences for their Meta-theoretic Study. Proceedings of the ninth Symposium on Logic in Computer Science, Paris. IEEE press, 1994.

[58] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la téorie des types. In *Proceedings of the 2nd Scandinavian Logic Symposium*. North-Holland, 1970.

[59] J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, 1972.

[60] J.-Y. Girard. Lambda-Calcul Typé. Notes de cours de DEA, 1986-87.

[61] K. Gödel. Über einer bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12, 1958. Réédité avec traduction anglaise en [62].

[62] K. Gödel. *Collected Works – vol. 1 : publications 1929-1936*, édité par S. Feferman. New York, Oxford University Press, 1986.

[63] K. Gödel. *Collected Works – vol. 2 : publications 1938-1974*, édité par S. Feferman. New York, Oxford University Press, 1990.

[64] M.J. Gordon, R. Milner, et C. Wadsworth. *Edinburgh LCF*. LNCS 78, Springer-Verlag, 1979.

[65] T.G. Griffin. A formulae-as-types notion of control. In *Proceedings of POPL 1990*, pp. 47–58, San Francisco, CA, 1990.

[66] R. Harrop. Concerning Formulas of the Types $A \rightarrow B \wedge C$, $A \rightarrow \exists x.B(x)$ in Intuitionistic Formal Systems. *The Journal of Symbolic Logic*, 25(1), 1960.

[67] S. Hayashi et H. Nakano. *PX, a Computational Logic*, Foundations of Computing. Foundations of Computing. MIT Press, 1988.

[68] H. Herbelin. Le théorème de Schroeder-Bernstein dans le Calcul des Constructions. Mémoire de DEA, Université Paris 7, 1988.

[69] H. Herbelin. Extraction de programmes à partir de preuves en logique classique, étude d'un exemple simple. Mémoire de Magistère, Ecole Normale Supérieure, 1991.

[70] J. R. Hindley et J. P. Seldin. *Introduction to Combinators ans λ-Calculus*, London Mathematical Society Student Texts 1. London Mathematical Society Student Texts 1. Cambridge University Press, 1986.

[71] W.A. Howard. The Formulae-as-Types Notion of Constructions. In J.P. Seldin et J.R. Hindley (rédacteurs), *to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[72] D.J. Howe. The computational behavior of Girard's paradox. In *Proc. IEEE Symp. on Logic in Computer Science*, pp. 205–214, June 1987.

[73] G. Huet. Formal Structures for Computation and Deduction. Course notes, Carnegie-Mellon University, mai 1986.

[74] G. Huet. Induction Principles Formalized in the Calculus of Constructions. In *TAPSOFT'87*. Springer-Verlag, 1987. LNCS 249.

[75] G. Huet. Adding Type :Type to the Calculus of Constructions. non publié, 1988.

[76] G. Huet (rédacteur). *Logical Foundations of Functional Programming*, The UT Year of Programming Series. The UT Year of Programming Series. Addison-Wesley, 1989.

[77] G. Huet. A Uniform Approach to Type Theory. In G. Huet (rédacteur), *Logical Foundations of Logical Programming*, pp. 337–397. Addison–Weseley, 1990. Rapport de Recherche 795, INRIA, 1988.

[78] G. Huet. The Gallina specification language : A case study. In *Proceedings of 12th FST/TCS Conference, New Delhi*. Springer Verlag LNCS, 1992.

[79] G. Huet. An analysis of Böhm's theorem. In S. Ronchi della Rocca M. Dezani-Ciancaglini et M. Venturini Zilli (rédacteurs), *To C. Böhm : Essays on Lambda-Calculus and Functional Programming*. Cambridge University Press, 1993.

[80] G. Huet et G. Plotkin (rédacteurs). *Proceedings of the first workshop on Logical Frameworks*, Antibes, mai 1990. Diffusion électronique.

[81] G. Huet et G. Plotkin (rédacteurs). *Logical Frameworks*. Cambridge University Press, mai 1991.

[82] G. Huet et Gordon Plotkin (rédacteurs). *Logical Environments*. Cambridge University Press, 1992.

[83] Y. Lafont J.-Y. Girard et P. Taylor. *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science 7. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1989.

[84] S.C. Kleene. *Introduction to Metamathematics*, Bibliotheca Mathematica. Bibliotheca Mathematica. North-Holland, 1952.

[85] J.-L. Krivine. *Théorie Axiomatique des Ensembles*. Presses Universitaires de France, 1969.

[86] J.-L. Krivine. Programmation en Arithmétique Fonctionnelle du Second ordre. Manuscript, 1987.

[87] J.-L. Krivine. Un algorithme non typable dans le système F. Manuscript, 1987.

[88] J.-L. Krivine. *Lambda-calcul types et modèles*, études et recherche en informatique. études et recherche en informatique. Masson, 1990.

[89] J.-L. Krivine et M. Parigot. Programming with Proofs. Preprint, presented at the 6th symposium on Computation Theory, Wendish-Rietz, Germany, 1987.

[90] K. Kunen. *Set Theory – An Introduction Independence Proofs*. North Holland, 1980.

[91] F. Leclerc et C. Paulin-Mohring. Programming with Streams in Coq. A case study : The Sieve of Eratosthenes. In K. Petersson B. Nordström et G. Plotkin (rédacteurs), *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pp. 245–262, 1992.

[92] X. Leroy. *Typage Polymorphe d'un Langage Algorithmique*. Thèse de Doctorat, Université Paris VII, 1992.

[93] Z. Luo. *An Extended Calculus of Constructions*. Thèse de PhD, University of Edinburgh, 1990.

[94] Z. Luo et R. Pollack. LEGO proof development system : User's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh., 1992.

[95] L. Magnusson. The new implementation of ALF. In Nordström et al. [109].

[96] P. Martin-Löf. A Theory of Types. Rapport technique, Report 71-3, Dept. of Mathematics, Université de Stockholm, 1971.

[97] P. Martin-Löf. An Intuitionistic theory of types : predicative part. In H.E. Rose et J.C. Shepherdson (rédacteur), *Logic Colloquium '73*, pp. 73–118, 1973.

[98] P. Martin-Löf. *Intuitionistic Type Theory.* Studies in Proof Theory, Bibliopolis, 1984.

[99] M. Mauny et X. Leroy. *The Caml-Light Reference Manual*, INRIA, 1992.

[100] P.F. Mendler. Recursive Types and Type Constraints in Second Order Lambda-Calculus. In *Symposium on Logic in Computer Science*, Ithaca, NY. IEEE press, 1987. IEEE.

[101] P.F. Mendler. *Inductive Definition in Type Theory.* Thèse de PhD, Cornell University, 1988.

[102] P.F. Mendler. Predicative type universes and primitive recursion. In *Symposium on Logic in Computer Science*, 1991.

[103] R. Milner, M. Tofte, et R. Harper. *The Definition of standard ML.* The MIT press, 1990.

[104] C. Mohring. Algorithm Development in the Calculus of Constructions. In *Symposium on Logic in Computer Science*, Cambridge, MA, 1986. IEEE Computer Society Press.

[105] C. Mohring. Extraction de programmes dans le Calcul des Constructions. Rapport technique, INRIA, 1986.

[106] C. Murthy. Classical Proofs as Programs : How, When and Why. In *Proceedings of the First Annual Symposiun on Constructivity in Computer Science*, 1991.

[107] C. Murthy. An Evaluation Semantics for Classical Proofs. In *Proceedings of the sixth Symposiun on Logic in Computer Science*, pp. 96–109, 1992.

[108] B. Nordström. Terminating General Recursion. *BIT*, 28, 1988.

[109] B. Nordström, K. Petersson, et G. Plotkin (rédacteurs). *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992. Diffusion électronique.

[110] B. Nordström, K. Petersson, et J. Smith. *Programming in Martin-Löf's Type Theory*, International Series of Monographs on Computer Science. International Series of Monographs on Computer Science, Oxford Science Publications, 1990.

[111] P. Odifreddi (rédacteur). *Logic and Computer Science.* Academic Press, 1990.

[112] C. Parent. Insertion dans les arbres equilibrés. Rapport de stage, juillet 1990.

[113] C. Parent. Automatisation partielle du développement de programmes dans le système COQ. Mémoire, ENS Lyon, juillet 1992.

[114] M. Parigot. Programming with Proofs : A Second Order Type Theory. In *ESOP'88*, Nancy, 1988. Springer-Verlag. LNCS.

[115] M. Parigot. On the Representation of Data in Lambda-Calculus. In *CSL'89*, LNCS, volume 440, Kaiserslautern, 1989. Springer-Verlag.

[116] M. Parigot. Recursive Programming with Proofs. *Theoretical Computer Science*, 94(2) :335–356, 1992.

[117] M. Parigot. Strong Normalization for Second Order Classical Natural Deduction. Proceedings of LICS 1994.

[118] C. Paulin et B. Werner. Extracting and Executing Programs developed in the Inductive Constructions System. In G. Huet et G. Plotkin (rédacteurs), *Proceedings of the first workshop on Logical Frameworks*, 1990.

[119] C. Paulin-Mohring. Extracting F_ω's programs from proofs in the Calculus of Constructions. Proceedings of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, 1989.

[120] C. Paulin-Mohring. *Extraction de programmes dans le Calcul des Constructions.* Thèse de Doctorat, Université Paris 7, janvier 1989.

[121] C. Paulin-Mohring. Inductive definitions in the Calculus of Constructions. Note technique parue dans [50], 1989.

[122] C. Paulin-Mohring. Recursive programs in the Calculus of Constructions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, novembre 1989.

[123] C. Paulin-Mohring. Réalisabilité et extraction de programmes. In B. Courcelle (rédacteur), *Logique et Informatique : une introduction*, Collection Didactique, volume 8, pp. 163–180, 1991.

[124] C. Paulin-Mohring. Inductive Definitions in the System Coq - Rules and Properties. Typed Lambda Calculi and Applications, M. Bezem et J. F. Groote (éditeurs). LNCS 664, Springer-Verlag, 1993.

Également Rapport de Recherche, LIP-ENS Lyon 92-49, 1992.

[125] C. Paulin-Mohring et B. Werner. Synthesis of ML programs in the system Coq. *Journal of Symbolic Computation*, vol. 15, pp 607-640, 1993.

[126] G. Peano. *Arithmetices principia, nova methodo exposita.* Turin, 1889. Reéditeé en traduction anglaise dans [148].

[127] F. Pfenning et C. Paulin-Mohring. Inductively defined types in the Calculus of Constructions. In *Proceedings of Mathematical Foundations of Programming Semantics*, LNCS 442, LNCS 442. Springer-Verlag, 1990. also technical report CMU-CS-89-209.

[128] B. Pierce, S. Dietzen, et S. Michaylov. Programming in Higher-Order Typed Lambda-Calculi. Rapport technique, Carnegie-Mellon University, 1989.

[129] G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1, 1975.

[130] D. Prawitz. *Natural Deduction.* Almqvist and Wiskell, Stockholm, 1965.

[131] F. Honsell et G. Plotkin R Harper. A Framework for Defining Logics. In *Proceedings of Logic in Computer Science.* IEEE Computer Society Press, 1987. pp. 194–204.

[132] F. Honsell et G. Plotkin R Harper. A Framework for Defining Logics. *Journal of the ACM*, 40 (1), pp. 143–184, 1993.

[133] J. Rouyer. Développement de l'Algorithme d'Unification dans le Calcul des Constructions. To appear as a technical report, août 1992.

[134] B. Russell. Lettre à Frege, 1912. Rééditée dans [148].

[135] J. Shoenfield. *Mathematical Logic*, Series in Logic. Series in Logic. Addison Wesley, 1967.

[136] J. Smith. Propositional Functions and Families of Types, *Notre-Dame Journal of Formal Logic.*

[137] M. Stefanova. A proof of Strong Normalization for the Calculus of Constructions with Natural Numbers. Communication personnelle, 1995.

[138] T. Streicher. *Semantics of Type Theory.* Birkhäuser, 1991.

[139] C. Svenson. A normalization proof for Martin-Löf's type theory. Rapport technique, Université de Göteborg, 1990.

[140] W.W. Tait. Intensional Interpretation of Functionals of Finite Type I. *Journal of Symbolic Logic*, 32 (2), 1967.

[141] D. Terrasse. Traduction de TYPOL en COQ. Application à Mini ML. Rapport de dea, IARFA, 1992.

[142] A.S. Troelstra (rédacteur). *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. LNM 344, Springer-Verlag, 1973.

[143] A.S. Troelstra et D. van Dalen. *Constructivism in Mathematics, an introduction*, Studies in Logic and the foundations of Mathematics, volumes 121 and 123. Studies in Logic and the foundations of Mathematics, volumes 121 and 123. North-Holland, 1988.

[144] V. Unger. Implantation de quelques algorithmes sur les tableaux dans le Calcul des Constructions. Rapport de Stage, juillet 1991.

[145] A Computation Model for Executable Higher-Order Algebraic Specification Languages. Proceedings de LICS 91.

[146] L. S. van Benthem Jutting. Typing in pure type systems. *Information and Computation*, 199+. à paraître.

[147] D. T. van Daalen. *The Language Theory of Automath*. Thèse de PhD, Technische Hogeschool Eindhoven, 1980.

[148] J. van Heijenoort (rédacteur). *From Frege to Gödel*. Harvard University Press, 1967.

[149] B. Werner. Synthèse et Exécution de Programmes Certifiés à partir du Calcul des Constructions. Mémoire, Université de Paris VII, septembre 1990.

[150] B. Werner. A Normalization Proof for an Impredicative Type System with Large Elimination over Integers. In Nordström et al. [109].

Table des matières

Résumé

L'objet de cette thèse est la méta-théorie du Calcul des Constructions Inductives (CCI), c'est à dire les Calcul des Constructions étendu par des types et des prédicats inductifs. Le Calcul des Constructions a été présenté en 1985 par Thierry Coquand. Il s'agit d'un λ-calcul typé qui, à travers l'isomorphisme dit de Curry-Howard, peut-être vu comme un formalisme logique. Ce système qui étend à la fois la logique d'ordre superieur de Church et les systèmes de Martin-Löf est particulièrement expressif du point de vue algorithmique et peut facilement être mis en œuvre sur ordinateur.

Dans le Calcul des Constructions originel, les types de données (entiers, listes, sommes, etc) sont représentés dans le λ-calcul à travers un codage imprédicatif. Cette solution est élégante mais conduit à un certain nombre de difficultés pratiques et théoriques. Pour y remédier, Thierry Coquand et Christine Paulin-Mohring on proposé d'étendre le formalisme par un mécanisme génerique de définitions inductives. C'est cette extension, utilisée dans le système Coq, qui est étudiée dans cette thèse. Le résultat essentiel est que le système vérifie bien la proprieté de normalisation forte. On en déduit les proprietés de cohérence logique, de confluence et de décidabilité du typage.

L'aspect le plus spectaculaire de l'extension par des types inductifs est la possibilité de définir de nouveaux types et de nouvelles propositions par récurrence structurelle (élimination forte). Cette caractéristique, qui donne toute sa signification à la notion de types dépendants, augmente énormément le pouvoir de la règle de conversion, et par là, la difficulté de la preuve de normalisation. L'interprétation de l'élimination forte dans une preuve de normalisation par réductibilité est la nouveauté essentielle de ce travail.

De plus, nous considérons ici un système avec η-conversion. Une conséquence est que la propriété de confluence n'est plus combinatoire et doit être prouvée après la normalisation, ce qui augmente à nouveau la difficulté de la preuve de celle-ci. A ce titre, nous présentons également quelques résultats nouveaux sur des systèmes non-normalisants qui montrent que pour des λ-calculs typés, la propriété de confluence est logique et non combinatoire.

Mots-Clés

Isomorphisme de Curry-Howard, Lambda-calcul typé, Normalisation, Réductibilité, Confluence, Types Inductifs, Calcul des Constructions.