ENS Lyon
Magistère d'Informatique et Modélisation
Rapport de stage de deuxième année – Été 2003

Un Calcul des Constructions Implicite

Étude d'une variante du calcul d'Alexandre Miquel

Lionel VAUX
Stage supervisé par Thierry COQUAND

Chalmers Tekniska Högskola

Avant-propos

Ceci constitue mon rapport de stage de deuxième année du Magistère d'Informatique et Modélisation de l'ENS Lyon.

Le stage s'est déroulé dans le département Computing Science de la Chalmers University of Technology, sous la direction de Thierry Coquand. Je tiens à remercier Thierry pour l'intérêt des sujets qu'il m'a proposés et son acceuil chaleureux. Merci aussi à Pierre Hyvernat pour le soutien moral et les sorties et Catarina Coquand pour l'acceuil encore, et aussi le fromage et le beurre sans sel. Merci enfin à Alexandre Miquel, que je n'ai toujours pas rencontré, mais sans qui ce travail n'aurait pas vu le jour, et dont le rapport de thèse est un exemple de clarté et d'exhaustivité.

Introduction

Depuis les années 1970, on sait exprimer par des termes du λ-calcul les preuves de propriétés formées dans de nombreux systèmes logiques, en utilisant la correspondance de Curry-Howard.

En particulier, le Calcul des Constructions Inductives (CIC) est à la base de l'assistant de preuve Coq, développé à l'INRIA. En tant que système de types d'ordre supérieur, cependant, la richesse de CIC a pour contre-partie une certaine lourdeur dans la syntaxe : contrairement aux systèmes de types des langages de la famille ML, des annotations de type systématiques sont nécessaires pour déterminer le type des objets.

Prenons l'exemple de listes polymorphes. En OCaml par exemple :

```
[] : 'a list
```

alors qu'en Coq :

```
nil : (A:Set)(list A)
```

Ainsi il n'existe pas en Coq de liste vide polymorphe, mais un constructeur qui, une fois instancié, donne la liste vide pour chaque type dans `Set`. En particulier, `nil` n'a pas le type `(list nat)`.

Dans les implémentations existantes, ce problème a reçu des solutions d'ordre pratique : certains termes ont des arguments marqués comme implicites et le système infère ces arguments lorsque les termes sont utilisés dans les preuves. Mais il faut garder à l'esprit que ces arguments invisibles pour l'utilisateur sont bel et bien conservés en interne dans les preuves.

Le Calcul des Constructions Implicite (CCI) d'Alexandre Miquel tente de répondre à ce problème. On pourra se reporter à [Miq98] et [Miq01] pour une description complète de ce calcul et de ses propriétés. Notons simplement que le système de types de Miquel comprend des sortes qui forment une hiérarchie d'univers très proche de celle du Calcul des Consructions Inductives du système Coq. Tous les jugements sont des jugements de typage : « dans le contexte Γ, le terme M a le type A ». Miquel démontre que que tout terme typable est fortement normalisable et que CCI est cohérent. Par ailleurs la relation de typage est indécidable.

Le but de cette présentation est de démontrer l'intérêt d'un calcul implicite, en introduisant un formalisme proche de CCI, mais sans hiérarchie d'univers, et même sans imprédicativité. Le calcul que nous présentons ici (noté CCI') est une variante de CCI, présumée plus simple. Il n'y a pas de sortes : les types sont distincts des termes. Les jugements ont plusieurs formes : bonne formation d'un contexte, d'un type dans un contexte, typage d'un terme dans un contexte et sous-typage. Ceci ce traduit par un plus grand nombre de règles, mais les propriétés des modèles sont plus faciles à établir.

Malgré ces modifications, l'intérêt des arguments implicites est conservé, ainsi qu'une certaine similarité avec les systèmes de types des langages ML. Pour s'en convaicre on pourra se reporter au paragraphe 1.3.

Plan Dans la section 1, on décrit le calcul CCI'. Dans les sections 2 et 3 on s'intéresse à ses modèles et leurs propriétés, et on établit des conditions suffisantes pour la correction. Dans la section 4 on introduit la notion de saturation, qu'on utilise dans la section 5 pour démontrer la normalisation faible des termes typables. Enfin, en section 6, on présente des pistes pour des travaux futurs ou en cours.

1 Le calcul

1.1 Syntaxe

On définit la forme syntaxique des termes, des types et des contextes par la grammaire suivante :

$$\begin{array}{rl} \text{Termes :} & M, N ::= x \mid \lambda x.M \mid M\ N \\ \text{Types :} & A, B ::= \mathsf{Set} \mid \mathsf{El}\ M \mid \Pi x{:}A.B \mid \cap x{:}A.B \\ \text{Contextes :} & \Gamma ::= \diamond \mid \Gamma, x : A \end{array}$$

Les termes du calcul sont en fait les termes (éventuellement ouverts) du λ-calcul pur. El transforme un terme en un type ; on verra plus tard que pour que $\mathsf{El}\ M$ soit bien formé, il faut que M ait le type Set, *i.e.* les termes dans Set sont des codes pour des types. $\Pi x{:}A.B$ est le produit dépendant usuel ; ici on l'appelle produit explicite, par opposition à $\cap x{:}A.B$, le produit implicite, qui suit la même construction mais est soumis à des règles de typage différentes. Les termes et types sont considérés modulo α-conversion.
Remarque Il n'y a pas de notion de sorte dans le système de types : les types sont syntaxiquement différents des termes (d'où la construction avec El) et ne sont pas eux-mêmes typables.

1.2 Typage

On définit simultanément des jugements de formes $\Gamma \vdash$, $\Gamma \vdash A$, $\Gamma \vdash M : A$ et $\Gamma \vdash A \leq B$ par les règles d'inférence données en figure 1. On utilisera la notation générique $\Gamma \vdash J$ pour tout jugement de l'une de ces formes, *i.e.* avec $J ::= \epsilon \mid A \mid M : A \mid A \leq B$.

1.2.1 Signification du produit implicite

Les deux règles de typage principales concernant le produit implicite sont (Gen) et (Inst) qui correspondent à (Abs) et (App) pour le produit explicite. On peut interpréter ces deux règles comme définissant une intersection : un terme est dans un type intersection ssi il est dans chaque type de l'intersection. Il est important de remarquer que dans (Gen), la variable liée par le produit ne peut apparaître dans le terme typé. En effet, le contexte dans la conséquence ne contient plus cette variable, alors que le terme est inchangé.

1.2.2 Produit implicite et sous-typage

Ici aussi, on peut utiliser une analogie avec l'intersection :
- un type est sous-type d'une intersection s'il est sous-type de tous les types de l'intersection ;
- une intersection est un sous-type d'un type si l'un des types de l'intersection est sous-type de ce type.

Contextes :

$$\frac{}{\diamond \vdash}\ (\text{WF-E}) \quad \frac{\Gamma \vdash A \quad x \notin DV(\Gamma)}{\Gamma, x : A \vdash}\ (\text{WF-C})$$

Types :

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set}}\ (\textsc{Set}) \quad \frac{\Gamma \vdash M : \mathsf{Set}}{\Gamma \vdash \mathsf{El}\ M}\ (\textsc{El})$$

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash \Pi x{:}A.B}\ (\textsc{Exp}) \quad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \cap x{:}A.B}\ (\textsc{Imp})$$

Termes :

$$\frac{\Gamma \vdash \quad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ (\textsc{Var})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : \Pi x{:}A.B}\ (\textsc{Abs}) \quad \frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B(N/x)}\ (\textsc{App})$$

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(M)}{\Gamma \vdash M : \cap x{:}A.B}\ (\textsc{Gen}) \quad \frac{\Gamma \vdash M : \cap x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M : B(N/x)}\ (\textsc{Inst})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A \leq B}{\Gamma \vdash M : B}\ (\textsc{Cum})$$

Sous-typage :

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set} \leq \mathsf{Set}}\ (\text{S-}\textsc{Set})$$

$$\frac{\Gamma \vdash M : \mathsf{Set} \quad \Gamma \vdash N : \mathsf{Set} \quad M =_{\beta\eta} N}{\Gamma \vdash \mathsf{El}\ M \leq \mathsf{El}\ N}\ (\text{S-}\textsc{El})$$

$$\frac{\Gamma \vdash A_2 \leq A_1 \quad \Gamma, x : A_2 \vdash B_1 \leq B_2 \quad \Gamma, x : A_1 \vdash B_1}{\Gamma \vdash \Pi x{:}A_1.B_1 \leq \Pi x{:}A_2.B_2}\ (\text{S-}\textsc{Exp})$$

$$\frac{\Gamma, x : A \vdash C \leq B \quad x \notin FV(C)}{\Gamma \vdash C \leq \cap x{:}A.B}\ (\text{S-}\textsc{Imp})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B(M/x) \leq C \quad \Gamma, x : A \vdash B}{\Gamma \vdash \cap x{:}A.B \leq C}\ (\textsc{Imp}\text{-S})$$

Renforcement :

$$\frac{\Gamma, x : A \vdash B \quad x \notin FV(B)}{\Gamma \vdash B}\ (\textsc{Str-Type})$$

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(M) \cup FV(B)}{\Gamma \vdash M : B}\ (\textsc{Str-Term})$$

$$\frac{\Gamma, x : A \vdash B \leq C \quad x \notin FV(B) \cup FV(C)}{\Gamma \vdash B \leq C}\ (\textsc{Str-Sub})$$

Fig. 1 – Règles de typage

1.2.3 Produit implicite et renforcement

Les règles de la forme :

$$\frac{\Gamma, x : A \vdash J \quad x \notin FV(J)}{\Gamma \vdash J}$$

sont appelées règles de renforcement et leur apparition en tant que règles de typage est atypique. En effet, ces règles ou les règles analogues sont généralement admissibles dans le Calcul des Constructions usuel et ses variantes, mais pas ici. Notons par exemple $\bot = \cap A{:}\mathsf{Set}.A$ le faux implicite. En utilisant les règles de renforcement, on peut prouver que $\cap x{:}\bot.\bot = \bot$ (extensionnellement : $\Gamma \vdash M : \bot \iff \Gamma \vdash M : \cap x{:}\bot.\bot$), ce qui est assez surprenant pour un opérateur qu'on voudrait identifier avec l'intersection ; en effet, on conjecture que $\bot$ n'a pas de preuve dans l'environnement vide.

Via les règles de renforcement, on peut en fait prouver plus généralement :

$$(\Gamma \vdash M : B \iff \Gamma \vdash M : \cap x{:}A.B) \text{ quand } x \notin FV(B)$$

même quand on ne trouve pas de N tel que $\Gamma \vdash N : A$. On prouve facilement $(\Rightarrow)$ dans le cas général, mais pour $(\Leftarrow)$ on a besoin d'introduire une variable de type A.

1.3 Quelques exemples

Commençons par un exemple très simple : la fonction identité. Dans CIC, elle est forcément paramétrée par un type. Dans un formalisme à la Coq, on écrirait par exemple :

$$id : \Pi A{:}\mathsf{Set}.(A \to A) := \lambda A : \mathsf{Set}.\lambda x : A.x$$

Dans CIC, on écrit :

$$id : \cap A{:}\mathsf{Set}.(\mathsf{El}\ A \to \mathsf{El}\ A) := \lambda x.x$$

C'est plus concis à la déclaration, mais aussi à l'utilisation, car dans CIC, l'identité a un seul argument.

Revenons maintenant aux listes polymorphes. Dans CIC, on a usuellement :

$$list : \mathsf{Set} \to \mathsf{Set}$$

$$nil : \Pi A{:}\mathsf{Set}.(list\ A)$$

$$cons : \Pi A{:}\mathsf{Set}.(A \to list\ A \to list\ A)$$

alors que dans CCI', on peut trouver :

$$list : \mathsf{Set} \to \mathsf{Set}$$

$$nil : \cap A{:}\mathsf{Set}.(list\ (\mathsf{El}\ A))$$

$$cons : \cap A{:}\mathsf{Set}.((\mathsf{El}\ A) \to list\ (\mathsf{El}\ A) \to list\ (\mathsf{El}\ A))$$

et alors le comportement est beaucoup plus proche des systèmes de types des langages ML. En effet, dans CCI', nil a bien le type $list$ $\mathsf{El}\ A$ pour tout A de type Set. De même *cons* ne nécessite pas d'argument pour préciser le type de base des listes.

On peut toutefois se demander quels arguments peuvent disparaitre des termes, *i.e.* quels produits explicites peuvent devenir implicites tout en gardant un comportement identique. La réponse réside dans la règle (GEN) : l'argument en question ne doit pas apparaître dans le reste de la preuve.
Par exemple, on peut écrire une fonction :

$$head : \cap A{:}\mathsf{Set}.\Pi l{:}list\ (\mathsf{El}\ A).((not\ (eq\ (list\ (\mathsf{El}\ A))\ l\ nil)) \to \mathsf{El}\ A)$$

mais on ne peut pas espérer écrire une fonction équivalente :

$$head' : \cap A{:}\mathsf{Set}.\Pi l{:}list\ (\mathsf{El}\ A). \cap p{:}not\ (eq\ (list\ (\mathsf{El}\ A))\ l\ nil).\mathsf{El}\ A$$

En effet, on a besoin de la preuve de $not\ (eq\ (list\ (\mathsf{El}\ A))\ l\ nil)$ pour obtenir le premier élément de l. En particulier, on peut prouver que : $head'\ nil : list\ (\mathsf{El}\ A)$, ce qui n'est pas le comportement attendu.

Une dernière remarque : le produit implicite est plus « fin » que le produit explicite. En effet, un produit explicite a toujours au moins autant de preuves que le produit implicite correspondant : si on a $\Gamma \vdash M : \cap x{:}A.B$, $x \notin FV(M)$, alors $\Gamma \vdash \lambda x.M : \Pi x{:}A.B$.

En particulier, le faux implicite $\perp = \cap A{:}\mathsf{Set}.(\mathsf{El}\ A)$ implique le faux explicite $\perp' = \Pi A{:}\mathsf{Set}.(\mathsf{El}\ A)$. On peut toutefois aussi montrer une réciproque :

$$false : \mathsf{Set}, cont : \mathsf{El}\ false \to \perp\ \vdash \lambda p.(cont\ (p\ false)) : \perp' \to \perp$$

1.4 Bonne formation des jugements dérivables

En plus de démontrer la bonne formation des jugements de typage, ce lemme est utile pour les preuves par induction car il implique la présence de certaines sous-dérivations.

Lemme 1 (Bonne formation) *Soit $\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ un contexte, A et B des types et M un terme. Posons $\Gamma[i] = x_1 : A_1, \ldots, x_i : A_i$. Les propriétés suivantes sont vérifiées :*

(i) *si $\Gamma \vdash$ est dérivable alors pour tout i t.q. $1 \leq i \leq n$, $FV(A_i) \subset DV(\Gamma[i-1])$ et $\Gamma[i-1] \vdash A_i$ apparaît dans la dérivation ;*

(ii) *si $\Gamma \vdash J$ est dérivable alors $FV(J) \subset DV(\Gamma)$ et $\Gamma \vdash$ apparaît dans la dérivation.*

Preuve. Les deux parties sont prouvées simultanément par induction sur la structure des dérivations. □

C'est le seule lemme que nous démontrerons à propos du calcul uniquement. Dans la suite, on introduit des modèles du calcul et des résultats sur ces modèles ; ensuite on utilisera un lemme de correction pour lier le calcul et ses modèles.

2 λ-modèle

La notion de λ-modèle est la structure sur laquelle repose l'interprétation des termes dans nos modèles. En axiomatisant cette structure, on peut l'abstraire de la description des modèles du calcul et ainsi gagner en généralité. On peut ensuite obtenir des résultats à propos du calcul en instanciant le λ-modèle sous-jacent.

2.1 Définition

Notons **Var** l'ensemble des variables et Λ celui des termes. Soit un ensemble $\mathfrak{D}$ donné, notons $\mathbf{Val}(\mathfrak{D})$ l'ensemble des fonctions de $\mathbf{Var} \longrightarrow \mathfrak{D}$ (environnements).

Définition 1 *Un λ-modèle est un triplet $\mathfrak{M} = (\mathfrak{D}, \circ, [\![.]\!]_.)$ t.q. $\circ\ :\ (\mathfrak{D} \times \mathfrak{D}) \longrightarrow \mathfrak{D}$ et $[\![.]\!]_.\ :\ \Lambda \times \mathbf{Val}(\mathfrak{D}) \longrightarrow \mathfrak{D}$ obéissant aux axiomes suivants :*

$$\begin{array}{lr} [\![x]\!]_\rho = \rho(x) & (\delta) \\ [\![M\ N]\!]_\rho = [\![M]\!]_\rho \circ [\![N]\!]_\rho & \textit{(App)} \\ [\![\lambda x.M]\!]_\rho \circ a = [\![M]\!]_{\rho[x\mapsto a]} & (\beta) \end{array}$$

$\circ$ est l'application dans le modèle et $[\![.]\!]_.$ est l'interprétation des termes dans les environnements sur $\mathfrak{D}$.

Ce sont les conditions minimales que l'on peut exiger d'une « bonne » interprétation des termes, mais dans la suite il nous faudra parfois imposer des propriétés supplémentaires, en particulier d'extensionnalité.

2.2 Semi-extensionnalité

Définition 2 *Un λ-modèle semi-extensionnel est un λ-modèle doté de l'axiome supplémentaire :*

$$(\forall a \in \mathfrak{D},\ [\![M]\!]_{\rho[x\mapsto a]} = [\![N]\!]_{\rho'[x\mapsto a]}) \Longrightarrow [\![\lambda x.M]\!]_\rho = [\![\lambda x.N]\!]_{\rho'}. \quad (\xi)$$

Dans un λ-modèle semi-extensionnel, les propriétés suivantes sont vérifiées.

Lemme 2 (Indifférence) $(\forall x \in FV(M),\ \rho(x) = \rho'(x)) \Longrightarrow [\![M]\!]_\rho = [\![M]\!]_{\rho'}$.

Preuve. Par induction sur M, en utilisant (ξ) pour l'abstraction. □

Lemme 3 (Substitution) $[\![M(N/x)]\!]_\rho = [\![M]\!]_{\rho[x\mapsto[\![N]\!]_\rho]}$.

Preuve. Par induction sur M. On détaille le cas $\lambda x.M$: modulo α-conversion dans $\lambda x.M$, $x \notin FV(N)$ et $x \neq y$, et alors :

$$\begin{array}{rcll} \forall a \in \mathfrak{D},\ [\![M(N/y)]\!]_{\rho[x\mapsto a]} & = & [\![M]\!]_{\rho[x\mapsto a][y\mapsto[\![N]\!]_{\rho[x\mapsto a]}]} & \text{(hypothèse d'induction)} \\ & = & [\![M]\!]_{\rho[x\mapsto a][y\mapsto[\![N]\!]_\rho]} & (x \notin FV(N)) \\ & = & [\![M]\!]_{\rho[y\mapsto[\![N]\!]_\rho][x\mapsto a]} & (x \neq y) \end{array}$$

donc $[\![(\lambda x.M)(N/y)]\!]_\rho = [\![\lambda x.M]\!]_{\rho[y\mapsto[\![N]\!]_\rho]}$ par (ξ). □

Lemme 4 (β-congruence) *Si $M =_\beta N$ alors $[\![M]\!]_\rho = [\![N]\!]_\rho$.*

Preuve. Il est suffisant de prouver le résultat quand $M \to_\beta N$. Par induction sur M : M n'est pas une variable et on utilise le lemme 3 pour l'application et (ξ) pour l'abstraction. □

2.3 Extensionnalité

Définition 3 *Un λ-modèle extensionnel est un λ-modèle avec l'axiome supplémentaire :*

$$(\forall a \in \mathfrak{D},\ b \circ a = c \circ a) \Longrightarrow b = c \qquad (Ext)$$

Lemme 5 (Affaiblissement de (Ext)**)** *Un λ-modèle extensionnel est semi-extensionnel.*

Preuve. (ξ) est direct par (β) et (Ext). □

Dans un λ-modèle extensionnel, on a le lemme suivant.

Lemme 6 ($\beta\eta$-congruence) *Si $M =_{\beta\eta} N$ alors $[\![M]\!]_\rho = [\![N]\!]_\rho$.*

Preuve. Il est suffisant de prouver le résultat quand $M \to_\beta N$ et $M \to_\eta N$. Le premier cas est donné par le lemme 4. Le second est direct avec (Ext) (et le lemme 3). □

3 Interprétation des types par des pers

Remarque Cette partie et la suivante tentent de généraliser la méthode utilisée dans [Tak]. Il est intéressant de noter que dans la suite, on parvient à établir une notion de saturation sans préciser le λ-modèle sous-jacent.

Dans ce qui suit, on définit une structure sur les λ-modèles pour modéliser les types : un per-modèle interprète un type dans un environnement par une relation d'équivalence partielle sur un λ-modèle. L'interprétation des types est ainsi paramétré par l'interprétation des termes.[1]

3.1 Pers

Une per (*partial equivalence relation*) $\mathcal{A}$ sur l'ensemble $\mathfrak{D}$ est une relation binaire symétrique et transitive. $\mathbf{Per}(\mathfrak{D})$ est l'ensemble de toutes les pers sur $\mathfrak{D}$. On écrira $u = v : \mathcal{A}$ pour $\mathcal{A}(u, v)$ et $u : \mathcal{A}$ pour $u = u : \mathcal{A}$. Quand $u : \mathcal{A}$, $\overline{u}_\mathcal{A}$ est la classe d'équivalence de u dans $\mathcal{A}$, *i.e.* l'ensemble de tous les $v \in \mathfrak{D}$ t.q. $u = v : \mathcal{A}$. On écrit $\mathfrak{D}/\mathcal{A}$ l'ensemble de toutes les classes d'équivalence dans $\mathcal{A}$. $\mathbf{Fam}(\mathcal{A})$ est l'ensemble de toutes les fonctions $\mathcal{F} : \mathfrak{D}/\mathcal{A} \to \mathbf{Per}(\mathfrak{D})$. Si $\mathcal{F} \in \mathbf{Fam}(\mathcal{A})$ et $u : \mathcal{A}$, on pourra écrire $\mathcal{F}(u)$ au lieu $\mathcal{F}(\overline{u}_\mathcal{A})$.

Supposons qu'on a une opération binaire $\circ$ sur $\mathfrak{D}$. Si $\mathcal{A} \in \mathbf{Per}(\mathfrak{D})$ et $\mathcal{F} \in \mathbf{Fam}(\mathcal{A})$ alors on définit $\overline{\Pi}(\mathcal{A}, \mathcal{F})$ et $\overline{\sqcap}(\mathcal{A}, \mathcal{F}) \in \mathbf{Per}(\mathfrak{D})$ par :

$$u_1 = u_2 : \overline{\Pi}(\mathcal{A}, \mathcal{F}) \text{ ssi } \forall v_1 = v_2 : \mathcal{A},\ u_1 \circ v_1 = u_2 \circ v_2 : \mathcal{F}(u_1) \text{ et}$$

$$u_1 = u_2 : \overline{\sqcap}(\mathcal{A}, \mathcal{F}) \text{ ssi } \forall v_1 = v_2 : \mathcal{A},\ u_1 = u_2 : \mathcal{F}(u_1).$$

Remarque On peut identifier chaque sous-ensemble $\mathcal{A}$ de $\mathfrak{D}$ avec une per sur $\mathfrak{D}$ par : $u_1 = u_2 : \mathcal{A} \iff (u_1 = u_2 \in \mathcal{A})$, *i.e.* $\overline{u}_\mathcal{A} = \{u\}$.

[1] D'autres structures peuvent être utilisées de cette même manière ; en particulier, il est possible de donner un modèle à notre version de CCI dans lequel les types sont interprétés par des sous-ensembles du λ-modèle – ce que nous ne présentons pas dans ce rapport.

3.2 Égalité intentionnelle sur les types

Dans la suite, on suppose que $\mathfrak{M} = (\mathfrak{D}, \circ, [\![.]\!]_.)$ est un λ-modèle, $* \in \mathbf{Per}(\mathfrak{D})$ et $\mathcal{E} \in \mathbf{Fam}(*)$. On note $\mathcal{T}_\Lambda$ l'ensemble des types.

On définit simultanément une relation binaire sur les couples (A, ρ), avec $A \in \mathrm{T}_\Lambda$ et $\rho \in \mathbf{Val}(\mathfrak{D})$, et une interprétation $(A, \rho) \mapsto [\![A]\!]_\rho \in \mathbf{Per}(\mathfrak{D})$ définie quand $[A]_\rho = [A]_\rho$ par :

- $[\mathsf{Set}]_{\rho_1} = [\mathsf{Set}]_{\rho_2}$ et $[\![\mathsf{Set}]\!]_{\rho_1} = *$;
- $[\mathsf{El}\ M_1]_{\rho_1} = [\mathsf{El}\ M_2]_{\rho_2}$ ssi $[\![M_1]\!]_{\rho_1} = [\![M_2]\!]_{\rho_2} : *$, et alors $[\![\mathsf{El}\ M_1]\!]_{\rho_1} = \mathcal{E}([\![M_1]\!]_{\rho_1})$;
- $[\mathrm{R}x_1{:}A_1.B_1]_{\rho_1} = [\mathrm{R}x_2{:}A_2.B_2]_{\rho_2}$ ssi
  - $[A_1]_{\rho_1} = [A_2]_{\rho_2}$,
  - $a_1 = a_2 : [\![A_1]\!]_{\rho_1} \Longrightarrow [B_1]_{\rho_1[x_1 \mapsto a_1]} = [B_2]_{\rho_2[x_2 \mapsto a_2]}$,
  - $\mathcal{F}_i\ :\ a \mapsto [\![B_i]\!]_{\rho_i[x_i \mapsto a]} \in Fam([\![A_i]\!]_{\rho_i})$,

  et alors $[\![\mathrm{R}x_1{:}A_1.B_1]\!]_{\rho_1} = \overline{\mathrm{R}}([\![A_1]\!]_{\rho_1}, \mathcal{F}_1)$ (avec $R = \Pi$ ou $\cap$).

On écrira $[A]_\rho$ pour $[A]_\rho = [A]_\rho$.

Lemme 7 *L'égalité intentionnelle est une per sur $\mathcal{T}_\Lambda \times \mathbf{Val}_\mathfrak{D}$.*

Preuve. Inductions. □

Lemme 8 *Si $[A_1]_{\rho_1} = [A_2]_{\rho_2}$ alors $[\![A_1]\!]_{\rho_1} = [\![A_2]\!]_{\rho_2}$.*

Preuve. Induction facile sur la preuve de $[A_1]_{\rho_1} = [A_2]_{\rho_2}$. □

Ainsi on peut omettre la dernière condition dans la preuve de $[\mathrm{R}x_1{:}A_1.B_1]_{\rho_1} = [\mathrm{R}x_2{:}A_2.B_2]_{\rho_2}$ comme indiqué dans le lemme suivant.

Corollaire 9 *Si $[A_1]_{\rho_1} = [A_2]_{\rho_2}$ et $a_1 = a_2 : [\![A_1]\!]_{\rho_1} \Longrightarrow [B_1]_{\rho_1[x_1 \mapsto a_1]} = [B_2]_{\rho_2[x_2 \mapsto a_2]}$, alors $[\mathrm{R}x_1{:}A_1.B_1]_{\rho_1} = [\mathrm{R}x_2{:}A_2.B_2]_{\rho_2}$.*

Si $\mathfrak{M}$ est semi-extensionnel, on a le lemme suivant.

Lemme 10 *Supposons $\forall x_1 \in FV(A_1),\ \rho_1(x_1) = \rho'_1(x_1)$ et $\forall x_2 \in FV(A_2),\ \rho_2(x_2) = \rho'_2(x_2)$. Alors $[A_1]_{\rho_1} = [A_2]_{\rho_2} \iff [A_1]_{\rho'_1} = [A_2]_{\rho'_2}$.*

Preuve. On prouve $[A_1]_{\rho_1} = [A_2]_{\rho_2} \Longrightarrow [A_1]_{\rho'_1} = [A_2]_{\rho'_2}$ par induction sur la preuve de $[A_1]_{\rho_1} = [A_2]_{\rho_2}$, en utilisant le lemme 2 dans le cas $[\mathsf{El}\ M_1]_{\rho_1} = [\mathsf{El}\ M_2]_{\rho_2}$. □

Lemme 11 $[B_1(M_1/x_1)]_{\rho_1} = [B_2(M_2/x_2)]_{\rho_2} \iff [B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]} = [B_2]_{\rho_2[x_2 \mapsto [\![M_2]\!]_{\rho_2}]}$ *et alors* $[B_1(M_1/x_1)]_{\rho_1} = [B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]}$.

Preuve. On prouve $[B_1(M_1/x_1)]_{\rho_1} = [B_2(M_2/x_2)]_{\rho_2} \Longrightarrow$ $([B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]} = [B_2]_{\rho_2[x_2 \mapsto [\![M_2]\!]_{\rho_2}]}$ et $[B_1(M_1/x_1)]_{\rho_1} = [B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]})$ par induction sur la preuve de $[B_1(M_1/x_1)]_{\rho_1} = [B_2(M_2/x_2)]_{\rho_2}$ en utilisant le lemme 3 dans le cas $[\mathsf{El}\ N_1(M_1/x_1)]_{\rho_1} = [\mathsf{El}\ N_2(M_2/x_2)]_{\rho_2}$.
On prouve $[B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]} = [B_2]_{\rho_2[x_2 \mapsto [\![M_2]\!]_{\rho_2}]} \Longrightarrow$ $([B_1(M_1/x_1)]_{\rho_1} = [B_2(M_2/x_2)]_{\rho_2}$ et $[B_1(M_1/x_1)]_{\rho_1} = [B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]})$ par induction sur la preuve de $[B_1]_{\rho_1[x_1 \mapsto [\![M_1]\!]_{\rho_1}]} = [B_2]_{\rho_2[x_2 \mapsto [\![M_2]\!]_{\rho_2}]}$, en utilisant le lemme 3 dans le cas $[\mathsf{El}\ N_1]_{\rho_1[x_1 \mapsto M_1]} = [\mathsf{El}\ N_2]_{\rho_2[x_2 \mapsto M_2]}$. □

3.3 Validité

3.3.1 Interprétation des contextes

Soit Γ un contexte, on définit $\lceil\!\lceil \Gamma \rfloor\!\rfloor \subset \mathbf{Val}(\mathfrak{D}) \times \mathbf{Val}(\mathfrak{D})$ par les règles :

$$\frac{}{\rho_1 = \rho_2 : \lceil\!\lceil \diamond \rfloor\!\rfloor}\ \text{(E-I)}$$

$$\frac{\rho_1 = \rho_2 : \lceil\!\lceil \Gamma \rfloor\!\rfloor \quad [A]_{\rho_1} = [A]_{\rho_2} \quad \rho_1(x) = \rho_2(x) : \lceil\!\lceil A \rfloor\!\rfloor_{\rho_1}}{\rho_1 = \rho_2 : \lceil\!\lceil \Gamma, x : A \rfloor\!\rfloor}\ \text{(C-I)}$$

3.3.2 Jugements valides

On définit simultanément des jugements de la forme $\Gamma \Vdash J$ par :

$$\frac{}{\diamond \Vdash}\ \text{(E-V)} \qquad \frac{\Gamma \Vdash A \quad x \notin DV(\Gamma)}{\Gamma, x : A \Vdash}\ \text{(C-V)}$$

$$\frac{\Gamma \Vdash \quad \forall \rho_1 = \rho_2 : \lceil\!\lceil \Gamma \rfloor\!\rfloor,\ [A_1]_{\rho_1} = [A_2]_{\rho_2}}{\Gamma \Vdash A_1 = A_2}\ \text{(Type-V)}$$

$$\frac{\Gamma \Vdash A \quad \forall \rho_1 = \rho_2 : \lceil\!\lceil \Gamma \rfloor\!\rfloor,\ [\![M_1]\!]_{\rho_1} = [\![M_2]\!]_{\rho_2} : \lceil\!\lceil A \rfloor\!\rfloor_{\rho_1}}{\Gamma \Vdash M_1 = M_2 : A}\ \text{(Term-V)}$$

$$\frac{\Gamma \Vdash A_1 \quad \Gamma \Vdash A_2 \quad \forall \rho : \lceil\!\lceil \Gamma \rfloor\!\rfloor,\ \lceil\!\lceil A_1 \rfloor\!\rfloor_\rho \subset \lceil\!\lceil A_2 \rfloor\!\rfloor_\rho}{\Gamma \Vdash A_1 \leq A_2}\ \text{(Sub-V)}$$

3.4 Correction

3.4.1 Propriétés utiles

Lemme 12 *Si* $\Gamma, x : A \Vdash$ *alors* $\Gamma \Vdash$.

Preuve. Inversion sur (C-V) et (Type-V). □

Lemme 13 *(i) Si* $\Gamma \Vdash$ *alors* $(\rho_1, \rho_2 \mapsto \rho_1 = \rho_2 : \lceil\!\lceil \Gamma \rfloor\!\rfloor) \in \mathbf{Per}(\mathbf{Val}(\mathfrak{D}))$;
(ii) $(A_1, A_2 \mapsto \Gamma \Vdash A_1 = A_2) \in \mathbf{Per}(\mathrm{T}_\Lambda)$;
(iii) $(M_1, M_2 \mapsto \Gamma \Vdash M_1 = M_2 : A) \in \mathbf{Per}(\Lambda)$.

Preuve. (i) est prouvé par induction sur Γ. (ii) est direct par le lemme 7. (iii) est aisé car l'interprétation d'un type est une per. □

Lemme 14 *Si* $\Gamma \Vdash$, $\rho_1 = \rho_2 : \lceil\!\lceil \Gamma \rfloor\!\rfloor$ *et* $y \notin DV(\Gamma)$ *alors* $\forall a_1, a_2 \in \mathfrak{D}$, $\rho_1[y \mapsto a_1] = \rho_2[y \mapsto a_2] : \lceil\!\lceil \Gamma \rfloor\!\rfloor$.

Preuve. Induction sur Γ. □

Lemme 15 *Si* $\Gamma_1 \subset \Gamma_2$ *et* $\Gamma_2 \Vdash$ *alors*

- $\lceil\!\lceil \Gamma_2 \rfloor\!\rfloor \subset \lceil\!\lceil \Gamma_1 \rfloor\!\rfloor$
- $\Gamma_1 \Vdash J \Longrightarrow \Gamma_2 \Vdash J$.

Preuve. Facile. On prouve d'abord le résultat avec $\Gamma_2 = \Gamma_1, x : A$. □

Lemme 16 *Supposons que* $\mathfrak{M}$ *est extensionnel. Si* $\Gamma \Vdash M_1$ *et* $M_1 \sim M_2$ *alors* $\Gamma \Vdash M_1 = M_2 : A$.

Preuve. Direct en utilisant le lemme 6. □

3.4.2 Théorème de correction

Théorème 17 *Supposons que $\mathfrak{M}$ est extensionnel. Supposons que pour tout $\rho \in \mathbf{Val}(\mathfrak{D})$ et tout $A \in \mathrm{T}_\Lambda$, $[A]_\rho$ implique $[\![A]\!]_\rho \neq \emptyset$. Alors les implications (écrites comme des inférences) suivantes sont vérifiées :*

$$\frac{}{\diamond \Vdash}\ (\text{M-WF-E}) \qquad \frac{\Gamma \Vdash A \quad x \notin DV(\Gamma)}{\Gamma, x : A \Vdash}\ (\text{M-WF-C}) \qquad \frac{\Gamma \Vdash}{\Gamma \Vdash \mathsf{Set}}\ (\text{M-Set})$$

$$\frac{\Gamma \Vdash M = N : \mathsf{Set}}{\Gamma \Vdash \mathsf{El}\ M = \mathsf{El}\ N}\ (\text{M-El}) \qquad \frac{\Gamma \Vdash A_1 = A_2 \quad \Gamma, x : A_1 \Vdash B_1 = B_2}{\Gamma \Vdash \Pi x{:}A_1.B_1 = \Pi x{:}A_2.B_2}\ (\text{M-Exp})$$

$$\frac{\Gamma \Vdash A_1 = A_2 \quad \Gamma, x : A_1 \Vdash B_1 = B_2}{\Gamma \Vdash \cap x{:}A_1.B_1 = \cap x{:}A_2.B_2}\ (\text{M-Imp}) \qquad \frac{\Gamma \Vdash \quad (x : A) \in \Gamma}{\Gamma \Vdash x : A}\ (\text{M-Var})$$

$$\frac{\Gamma, x : A \Vdash M_1 = M_2 : B}{\Gamma \Vdash \lambda x.M_1 = \lambda x.M_2 : \Pi x{:}A.B}\ (\text{M-Abs})$$

$$\frac{\Gamma \Vdash M_1 = M_2 : \Pi x{:}A.B \quad \Gamma \Vdash N_1 = N_2 : A}{\Gamma \Vdash M_1\ N_1 = M_2\ N_2 : B(N_1/x)}\ (\text{M-App})$$

$$\frac{\Gamma, x : A \Vdash M_1 = M_2 : B \quad x \notin FV(M_1) \cup FV(M_2)}{\Gamma \Vdash M_1 = M_2 : \cap x{:}A.B}\ (\text{M-Gen})$$

$$\frac{\Gamma \Vdash M_1 = M_2 : \cap x{:}A.B \quad \Gamma \Vdash N : A}{\Gamma \Vdash M_1 = M_2 : B(N/x)}\ (\text{M-Inst})$$

$$\frac{\Gamma \Vdash M_1 = M_2 : A \quad \Gamma \Vdash A \leq B}{\Gamma \Vdash M_1 = M_2 : B}\ (\text{M-Cum}) \qquad \frac{\Gamma \Vdash}{\Gamma \Vdash \mathsf{Set} \leq \mathsf{Set}}\ (\text{M-S-Set})$$

$$\frac{\Gamma \Vdash M_1 : \mathsf{Set} \quad \Gamma \Vdash M_2 : \mathsf{Set} \quad M_1 \sim M_2}{\Gamma \Vdash \mathsf{El}\ M_1 \leq \mathsf{El}\ M_2}\ (\text{M-S-El})$$

$$\frac{\Gamma \Vdash A_2 \leq A_1 \quad \Gamma, x : A_2 \Vdash B_1 \leq B_2 \quad \Gamma \Vdash \Pi x{:}A_1.B_1}{\Gamma \Vdash \Pi x{:}A_1.B_1 \leq \Pi x{:}A_2.B_2}\ (\text{M-S-Exp})$$

$$\frac{\Gamma, x : A \Vdash C \leq B \quad x \notin FV(C)}{\Gamma \Vdash C \leq \cap x{:}A.B}\ (\text{M-S-Imp})$$

$$\frac{\Gamma \Vdash M : A \quad \Gamma \Vdash B(M/x) \leq C \quad \Gamma \Vdash \cap x{:}A.B}{\Gamma \Vdash \cap x{:}A.B \leq C}\ (\text{M-Imp-S})$$

$$\frac{\Gamma, x : A \Vdash B_1 = B_2 \quad x \notin FV(B_1) \cup FV(B_2)}{\Gamma \Vdash B_1 = B_2}\ (\text{M-Str-Type})$$

$$\frac{\Gamma, x : A \Vdash M_1 = M_2 : B \quad x \notin FV(M_1) \cup FV(M_2) \cup FV(B)}{\Gamma \Vdash M_1 = M_2 : B}\ (\text{M-Str-Term})$$

$$\frac{\Gamma, x : A \Vdash B_1 \leq B_2 \quad x \notin FV(B_1) \cup FV(B_2)}{\Gamma \Vdash B_1 \leq B_2}\ (\text{M-Str-Sub})$$

Preuve. La preuve est longue mais facile. Chaque implication est prouvée par induction. L'ordre des preuves est important : en particulier il faut prouver (M-Str-Type) avant (M-S-Imp). L'hypothèse supplémentaire est utilisée dans les preuves des implications (M-Str-*). □

Corollaire 18 *Si $\mathfrak{M}$ est extensionnel et pour tout (A, ρ), $[A]_\rho \Longrightarrow [\![A]\!]_\rho \neq \emptyset$ alors $\Gamma \vdash J \Longrightarrow \Gamma \Vdash J$.*

Ainsi, nous avons déterminé des conditions suffisantes pour que notre modèle soit correct. Dans la section suivante, nous donnons une méthode pour établir ces conditions.

4 Saturation

L'idée derrière la notion de saturation est de trouver des propriétés qui sont conservées par les opérations utilisées dans les produits ($\overline{\Pi}$ et $\overline{\cap}$). Ainsi, si une telle propriété est vérifiée par $*$ et $\mathcal{E}(a)$, $\forall a : *$, alors elle est vérifiée par tout $\llbracket A \rrbracket_\rho$.

L'intérêt principal de la propriété de saturation que nous donnons réside dans le fait qu'elle est définie selon un λ-modèle abstrait.

On suppose que $\mathfrak{M} = (\mathfrak{D}, \circ, \llbracket . \rrbracket_.)$ est un λ-modèle.

Définition 4 $(\mathbf{N}, \mathbf{S})$ *est une paire stable sur* $\mathfrak{M}$ *ssi :*
- $\mathbf{N} \neq \emptyset$ *;*
- $\mathbf{N} \subset \mathbf{S} \in \mathbf{Per}(\mathfrak{D})$ *;*
- $\forall n_1 = n_2 : \mathbf{N},\ \forall s_1 = s_2 : \mathbf{S},\ n_1 \circ s_1 = n_2 \circ s_2 : \mathbf{N}$ *;*
- $(\forall n_1 = n_2 : \mathbf{N},\ a_1 \circ n_1 = a_2 \circ n_2 : \mathbf{S}) \Longrightarrow a_1 = a_2 : \mathbf{S}$.

Remarque Il peut être intéressant de considérer la dernière condition comme une sorte d'*extensionnalité relative* de $\mathbf{S}$ sur $\mathbf{N}$.

Dans la suite, on suppose donnée une paire stable $(\mathbf{N}, \mathbf{S})$ sur $\mathfrak{D}$.

Définition 5 $\mathcal{A} \in \mathbf{Per}(\mathfrak{D})$ *est saturé ssi* $\mathbf{N} \subset \mathcal{A} \subset \mathbf{S}$.

Lemme 19 *Supposons que* $*$ *est saturé et* $\forall a : *$, $\mathcal{E}(a)$ *est saturé. Alors* $[A]_\rho \Longrightarrow \llbracket A \rrbracket_\rho$ *est saturé.*

Preuve. Par induction sur la preuve de $[A]_\rho$. Set et $\mathsf{El}\ M$ sont des cas triviaux hypothèse.
Si $[\mathrm{R}x{:}A.B]_\rho$, alors $[A]_\rho$ et $\forall a_1 = a_2 : \llbracket A \rrbracket_\rho$, $[B]_{\rho[x \mapsto a_1]} = [B]_{\rho[x \mapsto a_2]}$. Ainsi $\llbracket A \rrbracket_\rho$ est saturé et $\forall a : \llbracket A \rrbracket_\rho$, $\llbracket B \rrbracket_{\rho[x \mapsto a]}$ est saturé.
$\forall a : \llbracket A \rrbracket_\rho$, $\mathbf{N} \subset \llbracket B \rrbracket_{\rho[x \mapsto a]}$ donc $\mathbf{N} \subset \llbracket \cap\, x{:}A.B \rrbracket_\rho$. Soit $c_1 = c_2 : \llbracket \cap\, x{:}A.B \rrbracket_\rho$. $\llbracket A \rrbracket_\rho$ est saturé et donc non vide. Soit $a : \llbracket A \rrbracket_\rho$, $c_1 = c_2 : \llbracket B \rrbracket_{\rho[x \mapsto a]} \subset \mathbf{S}$. On en déduit que $\llbracket \cap\, x{:}A.B \rrbracket_\rho$ est saturé.
Soit $n_1 = n_2 : \mathbf{N}$. $\forall a_1 = a_2 : \llbracket A \rrbracket_\rho$, $a_1 = a_2 : \mathbf{S}$. Donc $n_1 \circ a_1 = n_2 \circ a_2 : \mathbf{N} \subset \llbracket B \rrbracket_{\rho[x \mapsto a_1]}$. Ainsi $\mathbf{N} \subset \llbracket \Pi x{:}A.B \rrbracket_\rho$. Soit $c_1 = c_2 : \llbracket \Pi x{:}A.B \rrbracket_\rho$. $\forall n_1 = n_2 : \mathbf{N}$, $\mathbf{N} \subset \llbracket A \rrbracket_\rho$ implique $c_1 \circ n_1 = c_2 \circ n_2 : \llbracket B \rrbracket_{\rho[x \mapsto n_1]} \subset \mathbf{S}$. Et donc $c_1 = c_2 : \mathbf{S}$. On en déduit que $\llbracket \Pi x{:}A.B \rrbracket_\rho$ est saturé. □

Corollaire 20 *Supposons que* $\mathfrak{D}$ *est un* λ*-modèle extensionnel avec une paire stable* $(\mathbf{N}, \mathbf{S})$ *t.q.* $*$ *est saturé et* $\forall a : *$, $\mathcal{E}(a)$ *est saturé. Alors le théorème 18 s'applique.*

Preuve. $[A]_\rho \Longrightarrow \llbracket A \rrbracket_\rho$ saturé $\Longrightarrow \llbracket A \rrbracket_\rho \supset \mathbf{N} \neq \emptyset$. □

Dans la section suivante, on utilise la saturation dans un modèle particulier pour prouver la normalisation (faible) des termes typés dans le calcul.

5 Normalisation des termes typables

5.1 Pers sur Λ'

On définit $\Lambda' = \Lambda/\sim$.

Dans la suite on se place dans le per-modèle $\mathfrak{M}_{\Lambda'}$ défini par :
- $\mathfrak{D} = \Lambda'$;
- $M \circ N = M\ N$;
- $\llbracket M \rrbracket_\rho = M\ \rho$.

Remarque Pour que cette définition ait un sens, il faut identifier chaque terme M avec sa classe modulo $\sim$. Alors $\circ$ et $[\![\cdot]\!]_{.}$ sont bien définies grâce au lemme suivant :

Lemme 21 *Si* $M_1 \sim M_2$*,* $N_1 \sim N_2$ *et* $\rho_1 \sim \rho_2$ *alors* $M_1\ N_1 \sim M_2 N_2$ *et* $M_1\ \rho \sim M_2\ \rho$*.*

Ainsi, l'égalité dans $\mathfrak{M}_{\Lambda'}$ est $\sim$ sur les termes.

Définition 6 *Un terme* ν *est neutre ssi c'est une variable* x*, ou de la forme* $(\nu'\ S)$ *où* ν' *est neutre et* S *est fortement normalisable.*

Dans la suite, on note $\mathbf{N}$ l'ensemble de tous les termes neutres et $\mathbf{S}$ celui de tous les termes fortement normalisables. Plus précisément :

Définition 7 $M = N : \mathbf{N} \iff (M \sim \nu,\ N \sim \nu$ *et* ν *est neutre*$)$*.*
$M = N : \mathbf{S} \iff (M \sim P,\ N \sim P$ *et* P *est fortement normalisable*$)$*.*

Lemme 22 $(\Lambda', (M,N) \mapsto (M\ N), (M,\rho) \mapsto (M\ \rho))$ *est un* λ*-modèle extensionnel et* $(\mathbf{N},\mathbf{S})$ *est une paire stable.*

Preuve. Extensionnalité : supposons $(\forall P \in \Lambda,\ M\ P \sim N\ P)$, alors soit $x \notin FV(M) \cup FV(N)$, $M\ x \sim N\ x$ donc $M \leftarrow_\eta \lambda x.M\ x \sim \lambda x.N\ x \rightarrow_\eta N$.
Paire stable : les trois premières conditions sont directes ; supposons $(\forall \nu \in \mathbf{N},\ M\ \nu \sim N\ \nu \in \mathbf{S})$, comme précédemment cela implique $M \sim N$, et $M\ P \in \mathbf{S} \Longrightarrow M \in \mathbf{S}$ est toujours vrai. □

5.2 Les termes typables sont normalisables

On définit l'environnement identité $\rho_0 \in \mathbf{Val}_{\Lambda'}\ :\ x \mapsto x$.

Lemme 23 *Si* $\Gamma \Vdash$ *alors* $\rho_0 : [\![\Gamma]\!]$ *.*

Preuve. Induction sur Γ. □

Corollaire 24 *Si* $\Gamma \vdash M : A$ *alors* M *est* $\beta\eta$*-normalisable.*

Preuve. $\Gamma \vdash M : A \Longrightarrow \Gamma \Vdash M : A \Longrightarrow ([\![M]\!]_{\rho_0} : [\![A]\!]_{\rho_0}) \Longrightarrow M : \mathbf{S}$. Donc $M \sim M'$, avec M' fortement normalisable. □

6 Travaux à venir

6.1 Implémentation

L'étude du calcul présenté ici a donné lieu à une implémentation en OCaml. Le programme développé est un vérificateur de typage ; il s'agit en fait du portage OCaml d'un prototype en Haskell écrit par Thierry Coquand.

Cette implémentation est encore en développement et demande encore du travail pour être utilisable, en particulier sur le plan théorique. Entr'autres, la vérification de type est probablement indécidable.

Plus précisément : au cours de la vérification de type, on crée les méta-variables correspondant aux termes impliqués dans les règles (Inst) et (Imp-S).

On obtient ensuite des équations sur les termes qu'il faut résoudre, tout en respectant le type imposé pour la méta-variable.

Dans l'implémentation actuelle, on peut facilement se persuader que, modulo la présence probable de bugs, lorsque le programme accepte l'entrée alors le jugement de typage est dérivable. Par contre il existe des typages dérivables sur lesquels le vérificateur de type ne sait pas répondre.

Le problème n'est donc pas de trouver un algorithme qui répond sur tout entrée, mais de déterminer sur quelles familles l'implémentation fonctionne. Nous n'avons pas de réponse satisfaisante à cette interrogation. Il semble par contre qu'on peut obtenir le résultat suivant.

Lemme 25 *L'algorithme termine toujours avec comme réponse possible :*
- *le typage est dérivable ;*
- *une liste d'équations sur des termes et des méta-variables telle que le jugement de typage est dérivable si ce système d'équations admet une solution ;*
- *échec.*

On peut alors s'intéresser à des méthodes génériques pour résoudre de telles équations. On peut aussi essayer de déterminer si les équations obtenues ont une forme particulière. Tout ceci est pour l'instant très jeune mais on peut espérer quelques résultats intéressants.

Voici enfin quelques exemples de jugements acceptés par le programme. La syntaxe des objets est proche de celle de Coq : `(x:A)B` dénote $\Pi x{:}A.B$, `{x:A}B` dénote $\cap x{:}A.B$; comme on sait différencier un terme d'un type par la seule position, on note `M` pour $\mathsf{El}\ M$. Pour chaque déclaration (`id : A`), on vérifie que le type est bien formé dans le contexte courant, puis on augmente le contexte ; pour chaque vérification de type (`id : A = M`), on vérifie la bonne formation du type, puis le typage du terme, et on ajoute la déclaration correspondante au contexte.

Un lemme sur une égalité typée :

```
A:*
x:A
y:A
eq:A->A->*
f:A->A->A
com:{x:A}{y:A}(eq (f x y) (f y x))
cong:{x:A}{y:A}{z:A}{t:A} (eq x y) -> (eq z t) -> (eq (f x z) (f y t))
ref:{x:A}(eq x x)
sym:{x:A}{y:A}(eq x y)->(eq y x)
assoc:{x:A}{y:A}{z:A} (eq (f x (f y z)) (f (f x y) z))
lem: {x:A}{y:A}{z:A} (eq x y) -> (eq (f x z) (f y z)) = [h](cong h ref)
```

Illustration du 1.2.3 :

```
A:*
B:*
id:({x:A}B)->B=[x]x
```

Illustration du 1.3 :

```
perp:*
not:*->*
notE:{a:*}(not a)->a->perp
notI:{a:*}(a->perp)->(not a)
list: (A:*)*
nil:{A:*}(list A)
eq:{a:*}a->a->*
head: {A:*} (x:(list A)) {p: (not (eq x nil))} (list A)
error: {A:*} (list A)=(head nil)
```

CCI' est plus fin que CIC (non conservativité) :

```
eq:{a:*}a->a->*
trans:{a:*}{x:a}{y:a}{z:a}(eq x z)->(eq y z)->(eq x y)
id:{a:*}(a->a)->(a->a)->*
k:{a:*}(a->a)->(a->a)->(a->a)
axk:{a:*}{x:a->a}{y:a->a}(id (k x y) x)
n:*
z:n
b:*
true:b
f:{a:*}(a->a)->n
axf:{a:*}{x:a->a}{y:a->a}(id x y)->(eq (f x) (f y))
rem1:(eq (f (k ([x]x) ([x]z))) (f ([x]x))) =
   (axf axk)
rem2:(eq (f (k ([x]x) ([x]true))) (f ([x]x))) =
   (axf axk)
lem:(eq (f (k ([x]x) ([x]z))) (f (k ([x]x) ([x]true))))=
   (trans rem1 rem2)
```

6.2 Normalisation forte

On a établi la normalisation faible des termes bien typés. Mais dans le calcul original, Alexandre Miquel établit la normalisation forte ; la preuve en est toutefois beaucoup plus complexe, de par le modèle utilisé.

Il sera intéressant de s'y consacrer et d'essayer de trouver une preuve de normalisation forte dans le cadre simplifié de notre version du calcul.

6.3 Cohérence

La cohérence du calcul s'énonce ; « le faux implicite $\cap A{:}\mathsf{Set}.A$ n'a pas de preuve dans le contexte vide ».

Miquel établit la cohérence en démontrant un résultat de cohérence relative :

Lemme 26 *Si tous les termes typables admettent une forme normale de tête faible alors le calcul est cohérent.*

Ce lemme est aisé dès qu'on a la propriété d'auto-réduction :

Conjecture 27 *Si* $\Gamma \vdash M : A$ *et* $M \rightarrow_{\beta\eta} N$ *alors* $\Gamma \vdash N : A$.

La démonstration de cette propriété pour le calcul de Miquel est assez longue et complexe, et on peut là encore en espérer une version simplifiée pour notre calcul.

6.4 Logique du premier ordre

On peut enfin s'intéresser à une modélisation de la logique du premier ordre dans CCI', *i.e.* trouver une traduction des formules du premier ordre dans CCI' telle qu'une formule est valide ssi sa traduction admet un terme de preuve.

Références

[Miq98] Alexandre Miquel. *Arguments implicites dans le Calcul des Constructions.* Rapport de stage de DEA, Université Paris 7, 1998. Disponible sur `http://pauillac.inria.fr/~miquel/index.html`.

[Miq01] Alexandre Miquel. *Le Calcul des Constructions Implicite : syntaxe et sémantique.* Rapport de thèse, Université Paris 7, dec. 2001. Disponible sur `http://pauillac.inria.fr/~miquel/index.html`.

[Tak] Coquand — Pollack — Takeyama. A logical framework with dependently typed records. Disponible sur `http://www.cs.chalmers.se/~coquand/MODULE/main.html`.