Logique et jugements génériques

Emmanuel Jeandel

28 août 2001

Table des matières

Introduction

State College (sic) , petite ville à la taille des Etats-Unis, en plein coeur de la Pennsylvanie, pays des écureuils et des lapins, accueille également une grande université, Penn State University, et un laboratoire d'informatique de renom, et la simple évocation des noms d'Haskell Curry et Thomas Jech, qui exercèrent longtemps dans ces lieux, suffit à faire comprendre au lecteur que nous sommes ici au pays de la logique, de la programmation fonctionnelle, et de la théorie des ensembles.

C'est donc dans une optique toute naturelle que, sous l'encadrement de Dale Miller, le stagiaire s'est interessé ici au calcul des séquents, et plus exactement à un essai de codage de la notion de généricité dans celui-ci. Le stage était un stage de recherche pure, aucun résultat n'était attendu, et le stagiaire s'amusa donc avec toutes formes de logiques, essayant de trouver celle qui était la plus adéquate et la plus convaincante pour formuler cette notion de généricité. Ce rapport résume ici toutes les tentatives effectuées, pourquoi elles ont souvent échouées, et constitue donc ainsi le condensé de dix semaines de recherche.

Premier chapitre

Outils

1.1 Le calcul des séquents

1.1.1 Le formalisme

Le calcul des séquents est un formalisme qui permet de représenter des formules logiques et d'établir leur véracité à partir de règles d'inférences. Les expressions que l'on considère sont de la forme :

$$\Gamma \longrightarrow C$$

où Γ est un ensemble de formules appelées prémisses et C une formule appelée conclusion. La syntaxe utilisée pour la définition des formules dépend bien entendu de la logique à considérer, et celle-ci sera décrite plus loin. La sémantique qu'il faut donner à $\Gamma \longrightarrow C$ est : En supposant que les formules de Γ sont vraies, est-ce que je peux prouver que C est vraie. Une réponse positive correspondra à un arbre de preuves dont toutes les feuilles sont des axiomes, la non prouvabilité étant l'absence de cet arbre, propriété bien entendu plus difficile à démontrer. Toutefois, pour le calcul classique des séquents (sans les définitions), il a été démontré que tout ce qui est vrai (dans un sens à définir) est prouvable et réciproquement, et la non prouvabilité d'une formule est donc en ce cas équivalente à la preuve de sa négation.

1.1.2 Elimination des coupures

La plupart des calculs séquents possèdent, sous une forme ou une autre, la règle dite *cut* qui se présente sous la forme :

$$\frac{\Gamma \longrightarrow B \quad \Gamma, B \longrightarrow C}{\Gamma \longrightarrow C}\, cut$$

Sa justesse est totalement naturelle : Pour prouver qu'une formule C est vraie il est toujours possible de passer par un lemme B, de montrer que le lemme est vrai, puis de l'utiliser dans la démonstration du théorème. Cette règle pourtant peut souvent poser problèmes puisque c'est la seule qui introduit des nouvelles formules, sorties de nulle part, et la question qui souvent se pose est naturellement : Puis-je faire un preuve sans démontrer le lemme ? C'est ce principe qui conduit au fameux principe d'élimination des coupures (le fameux Hauptsatz de Gentzen ([2]) dont l'énoncé est le suivant :

théorème 1.1 (*Élimination des coupures*)

> *De toute preuve Π d'un sequent, on peut déduire une preuve qui n'utilise pas la règle cut*

Ceci n'est en fait qu'un principe et n'est donc pas nécessairement vérifié par toutes les logiques. Toutefois, il semble que sa non-véracité soit tout de même problématique, puisque cela voudrait dire que certaines preuves, intrinséquement, ne seraient faisables qu'en passant par un lemme tordu qui n'a rien à voir avec la proposition à considérer.

La propriété de cut elimination joue également un grand rôle dans la recherche automatique de preuve. En effet, si on cherche une preuve, et qu'on est sûr qu'il en existe une sans cut, cela limite sérieusement le champ de recherches, qui autrement se retrouverait tout de suite infini.

De plus, cette propriété permet souvent de prouver aisément que le système est cohérent, ou, plus exactement, qu'il existe des choses fausses. Il suffit en effet pour cela de prendre une proposition F qui parait fausse (souvent la logique considérée contient un symbole correspondant à l'absurde, ou en tout cas à quelque chose de faux) et de montrer que le séquent $\longrightarrow F$ n'est pas prouvable, ce qui vu le point précédent est souvent plus aisé que pour une logique sans cut.

Précisons que la règle signalée ci-dessus, dans certaines logiques (comme la logique linéaire), s'écrit plutôt :

$$\frac{\Gamma \longrightarrow B \quad B \longrightarrow C}{\Gamma \longrightarrow C}\,cut$$

mais le principe évoqué ci dessus est toujours valide.

1.2 L_λ, λ-Prolog

1.2.1 Syntaxe abstraite d'ordre supérieure

Contrairement à l'optique souvent envisagée en ce moment, et qui consiste à considérer le λ comme une façon d'obtenir toutes les fonctions, la démarche choisie par Dale Miller, Frank Pfenning, et la plupart de leurs stagiaires et thésards consiste à considérer uniquement le λ comme une façon de représenter la syntaxe. L'expression Syntaxe abstraite d'ordre supérieure (HOAS) est souvent utilisée pour décrire ce principe, mais celle-ci a depuis quelques années pris un sens totalement différent, et nous préfererons ici utiliser l'expression λ-tree pour désigner ce phénomène.

On ne sera donc pas étonné de trouver dans ce document des opérations de 'matching' et d'unification sur des fonctions, mais le lecteur devra se souvenir que ces fonctions doivent être uniquement vues comme des objets syntaxiques. Il ne s'agit donc pas ici d'une unification du second ordre (indécidable), mais d'un compromis entre les deux points de vue, la théorie sous-jacente étant désignée sous le nom de L_λ([6]).

1.2.2 λ-Prolog

Pour expérimenter l'unification du second ordre et permettre une programmation logique plus puissante, Dale Miller et Gopalan Nadathur ([8]) ont créé le langage λ-Prolog, qui se veut un langage logique, tout en permettant d'utiliser des fonctionnalités de haut niveau grâce à l'utilisation du λ, d'un typage, de fonctions, et des quantificateurs. λ-Prolog décrit en fait un système plus fort que L_λ dans lequel l'unification ne produit pas nécessairement un plus grand unificateur et pour lequel celle-ci est même souvent indécidable.. Le lecteur pourra se rapporter à [4] pour plus d'informations.

1.2.3 Exemples et principe

Signalons au lecteur que la partie suivante n'est pas nécessairement d'un attrait considérable, et que sa non lecture n'est pas gênante pour la suite du document, les idées de L_λ n'étant citées que dans une faible partie de celui-ci .

Illustrons les concepts de L_λ et de λ-Prolog. L'idée de base est que le λ doit être vu uniquement comme un lieur syntaxique et rien de plus. Ainsi l'expression :

$$\lambda x \lambda y (A\ x\ y)$$

signifie qu'on a accès à une fonction de deux variables, A, et de même syntaxiquement :

$$\lambda x \lambda y (A\ x)$$

nous dit que nous avons ici affaire à une fonction de deux variables qui ne dépend en fait que de sa première variable. Et ainsi, si on démontre par exemple :

$$\lambda x \lambda y (A\ x) = \lambda x \lambda y (B\ y)$$

on en déduira qu'en fait A et B ne dépendent pas de leurs variables respectives.

L'idée de L_λ est encore plus manifeste dans λ-Prolog. Considérons le code suivant. Nous supposons avoir les types ev et fm et les fonctions :

- `trans : (name→ev)→fm`
- `and : fm→fm→fm`
- `evand : ev→ev→ev`

En fait, on symbolise ici deux univers, `fm` et `ev`, contenant chacune une opération `and` (resp `evand`), et `trans` permet alors de faire la liaison entre les deux univers.Considérons les définitions suivantes (que tout amateur de prolog aisément comprendra)

- `prove (and A B) :- prove A, prove B.`
- `prove (trans (λl (evand (A l) (B l)) :-`
  `prove (and (trans A) (trans B)).`

maintenant, donnons la requête suivante a l'interpreteur Prolog

```
prove (trans (λl (evand (A l) (evand (B l) (C l)))))
```

l'interpréteur va essayer d'unifier avec la première règle, ce qui est impossible. Ensuite il va essayer de voir si l'expression ne ressemble pas à une fonction qui renverrait un objet du type $(evand\ (X\ l)\ (Y\ l))$ pour des X et Y donnés.
L'unification est possible, en posant

- `X = λl (A l) = A`
- `Y = λl (evand (B l) (C l))`

et l'interpréteur en arrive au but

```
prove (and (trans A)  (trans λl (evand (B l) (C l))))
```

qui se scinde en les deux buts

- `prove (trans A)`
- `prove (trans λl (evand (B l) (C l)))`

et le dernier suivant le meme principe va devenir

- `prove (trans A)`
- `prove (and (trans B) (trans C))`

puis naturellement

- `prove (trans A)`
- `prove (trans B)`
- `prove (trans C)`

1.3 Avertissement

Ainsi que le lecteur l'aura deviné dans les quelques pages précédentes, l'essentiel de ce rapport est consacré au calcul des séquents. Ce formalisme, quoiqu'assez simple, pourra donc lui paraître fort ennuyeux. Qu'il n'hésite pas dans ce cas à se reporter directement à la conclusion, évitant ainsi les affres d'une pénible lecture et ne s'intéressant ainsi qu'au côté sympatique du rapport.

Deuxième chapitre

Le calcul des séquents

Nous présentons ici rapidement le calcul des séquents classique, et son extension avec les définitions, base indispensable pour bien analyser la recherche de la généricité qui sera l'objet des parties suivantes.

2.1 La syntaxe

Les différents symboles suivants seront utilisés
- $\top : formula$
- $\bot : formula$
- $\supset, \wedge, \vee : formula \to formula \to formula$
- $\forall_\sigma : (\sigma \to formula) \to formula$
- $\exists_\sigma : (\sigma \to formula) \to formula$
- Des fonctions (ou plutôt ici des constructeurs) $f_i : \sigma \to \cdots \to \sigma \to \sigma$
- Des prédicats $p_i : \sigma \to \cdots \to \sigma \to formula$
- Des éléments$c$ de$\mathcal{C} \subset \sigma$, l'alphabet des constantes
- Des éléments$v$ de$\mathcal{V} \subset \sigma$, l'alphabet des variables
- Des éléments$d$ de$\mathcal{D} \subset \sigma$, l'alphabet des variables de définitions

définition 2.1 (*Terme*)

Un terme est un élément de σ construit sur l'alphabet $f_i, \mathcal{C}, \mathcal{V}$

Il est important de noter qu'il n'existe ici qu'un 'type' σ. En fait, tout ce qui sera dit ici marche également s'il existe plusieurs types distincts. Toutefois, pour simplifier l'exposé, nous supposerons qu'il n'y en a qu'un seul.
Remarquons également que ce qui ici est appelé fonction correspond plus à la notion classique de constructeur.

2.2 Description classique

$\wedge$

$$\frac{\Gamma, B, C \longrightarrow D}{\Gamma, B \wedge C \longrightarrow D} \wedge L \qquad \frac{\Gamma \longrightarrow B \quad \Gamma \longrightarrow C}{\Gamma \longrightarrow B \wedge C} \wedge R$$

$\vee$

$$\frac{\Gamma, B \longrightarrow D \quad \Gamma, C \longrightarrow D}{\Gamma, B \vee C \longrightarrow D} \vee L$$

$$\frac{\Gamma \longrightarrow B}{\Gamma \longrightarrow B \vee C} \vee R_1 \qquad \frac{\Gamma \longrightarrow C}{\Gamma \longrightarrow B \vee C} \vee R_2$$

$\supset$

$$\frac{\Gamma \longrightarrow B \quad \Gamma, C \longrightarrow D}{\Gamma, B \supset C \longrightarrow D} \supset L \qquad \frac{\Gamma, B \longrightarrow C}{\Gamma \longrightarrow B \supset C} \supset R$$

$\perp$,$\top$

$$\frac{}{\Gamma, \perp \longrightarrow C} \perp L \qquad \frac{}{\Gamma \longrightarrow \top} \top R$$

$\forall$

$$\frac{\Gamma, (f\ t) \longrightarrow C}{\Gamma, \forall \lambda x (f\ x) \longrightarrow C} \forall L \qquad \frac{\Gamma \longrightarrow (f\ y)}{\Gamma \longrightarrow \forall \lambda x (f\ x)} \forall R$$

$\exists$

$$\frac{\Gamma, (f\ y) \longrightarrow C}{\Gamma, \exists \lambda x (f\ x) \longrightarrow C} \exists L \qquad \frac{\Gamma \longrightarrow (f\ t)}{\Gamma \longrightarrow \exists \lambda x (f\ x)} \exists R$$

y est une variable fraîche, i.e non libre dans f, et t est un terme.

Règles initiales et cut

$$\frac{}{\Gamma, A \longrightarrow A} init \qquad \frac{\Gamma \longrightarrow B \quad \Gamma, B \longrightarrow C}{\Gamma \longrightarrow C} cut$$

Règles structurelles

$$\frac{\Gamma, A, A \longrightarrow C}{\Gamma, A \longrightarrow C} contr \qquad \frac{\Gamma \longrightarrow C}{\Gamma, A \longrightarrow C} weaken$$

2.3 Ajout des définitions

La définition d'un prédicat p est un ensemble d'objets de la forme

$$\forall \overline{d}[(p\ \overline{t}) \stackrel{\triangle}{=} (f\ \overline{d})]$$

où les variables présentes dans les t_i sont précisément les $d_i \in \mathcal{D}$.

Ainsi,la définition du prédicat append est :

$$\forall M, (append\ nil\ M\ M) \stackrel{\triangle}{=} \top$$
$$\forall x \forall L \forall M \forall P (append\ (cons\ x\ L)\ M\ (cons\ x\ P)) \stackrel{\triangle}{=} (append\ L\ M\ P).$$

Certaines conditions supplémentaires sont nécessaires pour certifier que $\forall \overline{d}[(p\ \overline{t}) \stackrel{\triangle}{=} (f\ \overline{d})]$ est bien une définition. Le lecteur peut se rapporter à [3] pour plus de détails. Ainsi la ligne suivante, qui définit la satisfaction :

$$\forall x(sat\ (non\ x)) \stackrel{\triangle}{=} (sat\ x) \supset \bot$$

n'est pas valide. En effet, à droite, le prédicat sat se retrouve à gauche d'un $\supset$, et ceci est interdit, via l'introduction de niveaux. L'intérêt principal de cette limitation est de permettre l'élimination des coupures, puisqu'elle n'est pas valable sans cela. Nous ne nous étendrons pas sur ce propos, qui n'est pas l'objectif de ce rapport.

Définissons maintenant la notion de substitution

définition 2.2 (*Substitutions*)

Une substitution θ est une application de $\mathcal{V}$ dans les termes, et une substitution de définition τ est une application de $\mathcal{D}$ dans les termes. Si A est une formule, on définit assez naturellement de manière inductive $A\theta$.

On peut maintenant donner les règles correspondant aux définitions

$$\frac{\Gamma \longrightarrow B\tau}{\Gamma \longrightarrow A}\ def\ R$$

pour une définition $H \stackrel{\triangle}{=} B$ telle que $H\tau = A$

$$\frac{\left\{\Gamma\theta, B\tau \longrightarrow C\theta, \text{Pour tous } \tau, \theta, \text{ et definitions } H \stackrel{\triangle}{=} B \text{ si } H\tau = A\theta\right\}}{\Gamma, A \longrightarrow C}\ def\ L$$

Cette règle conduit apparemment à une infinité de prémisses. Mais en fait, on peut démontrer que le nombre de prémisses peut être limité, si on introduit des notions de plus grand unificateur (MGU, CSU, etc).

2.4 Exemples

Afin de bien visualiser comment marche le mecanisme des définitions, nous allons examiner une preuve dans cette logique. Nous supposerons le lecteur familier avec la logique des séquents, et nous nous intéresserons donc uniquement à la partie concernant les définitions. Nous allons donc prouver :

$$\longrightarrow \forall x \forall y \forall L \forall M (append\ (cons\ x\ nil)\ M\ L) \supset (append\ nil\ (cons\ x\ M)\ L)$$

Après quelques étapes, le 'calcul' mène à :

$$(append\ (cons\ x\ nil)\ M\ L) \longrightarrow (append\ nil\ (cons\ x\ M)\ L)$$

Le mécanisme des définitions entre alors en jeu. On va donc commencer ici par appliquer la règle $defL$. En fait cela revient à chercher *comment* on a pu avoir $append\ (cons\ x\ nil)\ M\ L)$ Aucune unification n'est possible avec la première règle :

$$(append\ nil\ D\ D) \stackrel{\triangle}{=} \top$$

puisqu'il faudrait unifier nil et $(cons\ x\ nil)$ ce qui n'est pas possible Avec la seconde cependant :

$$(append\ (cons\ d_1\ D_2)\ D_3\ (cons\ d_1\ D_4)) \stackrel{\triangle}{=} (append\ D_2\ D_3\ D_4)$$

il suffit de prendre la substitution qui à L associe $(cons\ x\ T)$ pour un T nouveau, et la substitution :

$$\begin{cases} d_1 \mapsto x \\ D_2 \mapsto nil \\ D_3 \mapsto M \\ D_4 \mapsto T \end{cases}$$

Cette solution correspond d'une certaine façon à la notion d'une 'solution maximale', au sens des unificateurs indiqués plus haut . Le séquent devient donc :

$$\frac{(append\ nil\ M\ T) \longrightarrow (append\ nil\ (cons\ x\ M)\ (cons\ x\ T))}{(append\ (cons\ x\ nil)\ M\ L) \longrightarrow (append\ nil\ (cons\ x\ M)\ L)}\ def\ L$$

Appliquant une nouvelle fois la règle $def\ L$, on va pouvoir conclure a l'égalité de M et T :

$$\frac{\top \longrightarrow (append\ nil\ (cons\ x\ T)\ (cons\ x\ T))}{(append\ nil\ M\ T) \longrightarrow (append\ nil\ (cons\ x\ M)\ (cons\ x\ T))}\ def\ L$$

et il suffit d'une application de $def\ R$ pour arriver à

$$\frac{\top \longrightarrow \top}{\top \longrightarrow (append\ nil\ (cons\ x\ T)\ (cons\ x\ T))}\ def\ R$$

Et la preuve est finie. Le lecteur pourra se reporter à l'annexe A.1 pour voir toutes les étapes détaillées.

2.5 Elimination des coupures

La preuve de l'élimination des coupures peut se trouver dans [3]. Cette preuve est basée sur une transformation de preuves : On réduit une preuve Π en une preuve Π' par une relation R, on démontre que R termine et que les formes normales sont sans cut, et la preuve est terminée .

2.6 Implantation de la logique

Pour pouvoir expérimenter la logique, Dale Miller et Jeremie Wajs ont alors construit un démonstrateur ([10]), nommé Iris, entièrement codé en λProlog, en faisant l'un des premiers systèmes du type où le mecanisme de syntaxe abstraite d'ordre supérieure (HOAS) est directement réalisé par le compilateur, et donc pour lequel il est inutile d'avoir à l'implanter. En fait, Iris couvre une logique un peu plus large, $FO\lambda^{\Delta\mathrm{N}}$ (First Order Logic with Definitions and Natural induction), mais nous ne parlerons pas dans cet exposé de l'induction sur les entiers.

Troisième chapitre

La logique avec jugements génériques

On veut maintenant introduire une façon de considérer des quantificateurs génériques. Par exemple, si on considère la proposition suivante du π-calcul :

$$congruence\ \nu n(P\ n)\ \nu n(Q\ n) \stackrel{\triangle}{=} \forall n, congruence\ (Pn)\ (Qn)$$

Celle-ci ne suppose pas que le n est générique. Ainsi, une preuve de congruence pourra par exemple procéder par cas sur n, ce qui n'est pas dans l'esprit de la règle. On veut plutôt arriver à prouver la propriété sans être obligé de savoir "combien" vaut n. Dale Miller, dans [5] évoquait une possibilité de codage de cette propriété au niveau objet, c'est à dire au niveau des définitions. Nous inspirant de cette idée, nous allons tenter ici de procéder à un codage *directement dans la logique*.

3.1 Motivations

Résumons le but recherché : Nous voulons introduire un nouveau quantificateur, qui sera noté $\bigwedge$, et qui signifiera 'pour tout x générique'. Mais que veut dire exactement : être générique ?

3.1.1 Le π-calcul, les variables fraîches

Dans le π-calcul ([7]), on note la présence d'un opérateur de restriction ν, lieur dont le but est bien ici de signifier 'pour un nom frais'. On retrouve exactement l'idée que l'on cherche à exprimer.

Considérons un encodage classique du π-calcul. Nous aurions alors :

$$\frac{P \xrightarrow{\tau} Q}{[x = x]P \xrightarrow{\tau} Q}$$

$$\frac{\forall n, (P\ n) \xrightarrow{\tau} (Q\ n)}{\nu n(P\ n) \xrightarrow{\tau} \nu n(Q\ n)}$$

Mais dans ce cas, une proposition comme : $\nu n[x = n]P \xrightarrow{\tau} P \supset false$ n'est pas prouvable. En effet, le calcul des séquents transformera, à l'aide de $\supset R$ et

$def\ L$ le séquent en :

$$(\forall n[x = n]P \xrightarrow{T} P) \longrightarrow false$$

et il faudrait pouvoir instancier n avec quelque chose de nouveau. Mais si on l'instancie par exemple avec une variable y nouvelle, l'application de $def\ L$, contrairement au but recherché, n'échouera pas, mais au contraire concluera à l'égalité de x et y. En fait, il faudrait ici réussir à instancier n par quelque chose **qui ne peut pas devenir** x. .

3.1.2 L'exemple fondamental

Nous reprenons ici l'exemple de [5] qui illustre exactement le concept recherché.

Considérons les définitions suivantes :

$$\begin{cases} (p\ X\ X\ Y) \triangleq \top \\ (p\ X\ Y\ X) \triangleq \top \\ (p\ Y\ X\ X) \triangleq \top \end{cases} \quad eq\ X\ X \triangleq \top$$

le prédicat p est donc vrai si deux de ses arguments sont identiques.

Supposons avoir dans le langage une fonction $pr : \sigma \to \sigma \to \sigma$.

Il alors assez logique de vouloir que la proposition suivante soit vraie :

$$\bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3) \supset (eq\ t_1\ t_2)$$

En effet à priori, pour que p soit vraie, la seule possibilité est d'avoir utilisé la première des trois définitions plus haut proposées. En effet, il aurait fallu sinon réussir à unifier deux variables génériques différentes, ce qui est impossible. Ici, malheureusement, la proposition :

$$\forall y_1 \forall y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3) \supset (eq\ t_1\ t_2)$$

est impossible à prouver (en tout cas par un calcul des séquents) puisqu'il faudrait instancier les variables y_1 et y_2, mais avec quoi ? Nous n'avons aucune constante pour nous aider dans cette tâche ! Ainsi, l'intérêt de ce nouveau quantificateur apparaît clairement sur cet exemple.

3.2 La syntaxe

Introduisons la syntaxe qui sera utilisée pour essayer de définir ce que doit être le quantificateur. Nous allons en fait, puisqu'il s'agit bien d'ajouter quelque chose au calcul, étendre la syntaxe, avec :

– Des éléments g de $\mathcal{G} \subset \sigma$, l'alphabet des variables génériques

définition 3.1 (*terme générique*)

Un terme sera dit non générique s'il est formé sur l'alphabet $f_i \cup \mathcal{V} \cup \mathcal{C}$, générique sinon (id est formé sur $f_i \cup \mathcal{V} \cup \mathcal{C} \cup \mathcal{G}$) .

3.3 Le calcul sous-jacent

Les règles ne changent pas pour les opérateurs de base, on ajoute toutefois aux règles sur $\exists$ et $\forall$ la précision : :

$\exists$

$$\frac{\Gamma, (f\ y) \longrightarrow C}{\Gamma, \exists \lambda x (f\ x) \longrightarrow C} \exists L \qquad \frac{\Gamma \longrightarrow (f\ t)}{\Gamma \longrightarrow \exists \lambda x (f\ x)} \exists R$$

$\forall$

$$\frac{\Gamma, (f\ t) \longrightarrow C}{\Gamma, \forall_\sigma \lambda x (f\ x) \longrightarrow C} \forall L \qquad \frac{\Gamma \longrightarrow (f\ y)}{\Gamma \longrightarrow \forall_\sigma \lambda x (f\ x)} \forall R$$

y est ici une variable fraiche de $\mathcal{V}$ et t un terme **non générique**.

3.4 Et la règle $\bigwedge$?

Assez logiquement, la règle d'introduction de $\bigwedge$ doit être :

$$\frac{\Gamma \longrightarrow (f\ y)}{\Gamma \longrightarrow \bigwedge_\sigma \lambda x (f\ x)} \bigwedge R$$

où y est une variable fraiche de $\mathcal{G}$.

Et la règle d'élimination doit être de la forme :

$$\frac{\Gamma, (f\ t) \longrightarrow C}{\Gamma, \bigwedge_\sigma \lambda x (f\ x) \longrightarrow C} \bigwedge L$$

Le problème est bien entendu quel choix prendre pour t ?

- Si on prend t un terme quelconque, on admet implicitement que si on a prouvé quelque chose pour une variable générique, on l'a prouvé pour toute variable. Ceci pourrait être plausible, mais ce n'est pas la définition de 'générique' que nous souhaitions ici définir. À priori, l'idée d'une variable générique est qu'elle n'est pas décomposable (par exemple, on ne peut pas dire qu'un entier est soit nul soit le successeur d'un entier), et les démonstrations avec les variables génériques sont à priori plus faciles. Ainsi, ce choix paraît délicat. Il a été pourtant choisi dans un premier lieu, mais sa mise en place conduit à une logique complètement différente, dans lequel prouver quelque chose sur une variable générique était plus dur que pour une variable normale.
- Si on prend t une variable générique quelconque. Ce choix paraît bien plus acceptable : Si on a prouvé quelque chose sur une variable générique, on l'a prouvé pour toutes les variables génériques. Cependant, un petit problème subsiste. En effet, on peut alors prouver :

  $$\bigwedge \lambda x \bigwedge \lambda y (P\ x\ y) \longrightarrow \bigwedge \lambda x (P\ x\ x)$$

  Il suffira d'instancier les deux quantificateurs avec la variable générique obtenue par application de la règle d'élimination droite. Mais cette règle (vraie pour le quantificateur $\forall$) parait ici infondée. En effet, à priori, quand on a démontré

  $$\longrightarrow \bigwedge \lambda x \bigwedge \lambda y (P\ x\ y)$$

on a écrit

$$\longrightarrow (P\ g_1\ g_2)$$

pour deux variables g_1 et g_2 et on s'attend à ce que on ait plus ou moins implicitement supposé g_1 et g_2 distinctes. Ceci n'est absolument pas une nécessité, mais semble assez raisonnable vis-à-vis des exemples que nous voulons couvrir. En effet, dans une tentative de codage du π-calcul, nous souhaiterions que que la satisfaction de

$$(\nu n)(P\ n) \to (\nu n)(Q\ n)$$

soit codée par

$$\bigwedge \lambda n((P\ n) \to (Q\ n))$$

Et dans ce cas, la satisfaction d'une formule comme

$$(\nu n)(\nu m)(P\ n\ m) \to \ldots$$

devra considérer les deux variables comme distinctes

– La troisième solution est d'autoriser t à être une variable générique *qui n'apparaissait pas déjà* dans f. Cette possibilité semble intéressante, et sera donc étudiée plus précisément dans le paragraphe suivant.
– Introduire une nouvelle variable, donc une variable qui n'existait pas encore. Cette solution est stable et conduit à certains résultats, mais rend cependant certaines preuves délicates, puisqu'on perd l'*atomicité* En effet, le séquent

$$\bigwedge x(Ax) \longrightarrow \bigwedge x(Ax)$$

est prouvable par la règle $init$. Mais si on applique $\bigwedge R$, on obtient

$$\bigwedge x(Ax) \longrightarrow (Ay)$$

et ici, on est bloqué : il n'y a aucun moyen de prouver ce séquent. Ce type de propriété d'atomicité est pourtant très important, puisqu'il symbolise le fait qu'il est toujours possible, dans une politique de recherche de preuves 'goal directed' de faire toutes les transformations possibles, puisque celles-ci ne changent pas la satisfaction.
Ainsi, seule la troisième possibilité semble viable.

3.5 Hélàs, trois fois hélàs

Bien que cette dernière solution apparaisse comme idéale, elle n'est toujours pas valide. En effet, le séquent ci-dessous :

$$\longrightarrow \bigwedge x(A\ x \supset (\bigwedge z(A\ z \supset B\ x)) \supset B\ x)$$

n'est pas prouvable sans cut, mais une preuve existe si on ajoute le cut, comme l'annexe A.3 le montre. En effet, l'expression

$$A\ x \longrightarrow (\bigwedge z(A\ z \supset B\ x)) \supset B\ x$$

doit être fausse, mais elle se réduit par un cut en les expressions

$$A\,x \longrightarrow \bigwedge y \bigwedge z(A\,z \supset B\,y)) \supset B\,y)$$

et

$$A\,x, \bigwedge y \bigwedge z(A\,z \supset B\,y)) \supset B\,y) \longrightarrow (\bigwedge z((A\,z) \supset B\,x)) \supset B\,x)$$

qui selon toutes apparences sont justes.Nous n'avons donc pas d'élimination des coupures, et la logique a donc de bonnes chances de ne pas être consistante.

Ce contre exemple est beaucoup plus important qu'il n'y paraît puisqu'il indique en fait *qu'il n'est pas possible* de trouver une définition acceptable de $\bigwedge$ dans le calcul des séquents. En effet, il suffit de regarder la ligne précédent le cut. Avant celle-ci nous avons juste utilisé la règle $\bigwedge R$ qui semble assez indiscutable. Et au niveau du cut, nous avons une formule que nous voulons fausse, mais les deux prémisses sont vraies (si on les examine hors contexte), et en fait nous les voulons vraies : la satisfaction de ces deux formules doit être une conséquence de la théorie. Ainsi, toute tentative semble vouée à l'échec. Pour être complet, nous signalerons que Andrew Pitts et Murdoch Gabbay, dans [1] réussirent à donner une définition de ce quantificateur en théorie des ensembles, comme signifiant : 'pour tout x, sauf un nombre fini'. On trouve dans un preprint du papier les règles proposées pour ce quantificateur dans le calcul des séquents, mais celles-ci ont curieusement été supprimées de l'article final, leur formulation ne leur semblant visiblement pas satisfaisante.

3.6 Examen de la situation

Le problème se pose de la façon suivante :

– Si nous considérons que nous pouvons alors remplacer $\bigwedge x$ par n'importe quelle variable générique, nous arrivons à une logique qui ne nous satisfait pas, et qui est incohérente.
– Si nous essayons de remplacer $\bigwedge x$ par une variable générique qui n'existe pas encore, nous avons là encore des problèmes

En fait, il semble que le principal problème ici est que les variables introduites ne sont pas *liées* les unes aux autres de façon suffisante . La liaison apparaît nécessaire à cause de la première remarque, mais la liaison promise par la deuxième méthode est encore insuffisante. Hélàs, il semblerait qu'il est difficile de faire entrer cette liaison dans le calcul des séquents. Nous avons remarqué que la seule possibilité qui codait exactement ce que nous voulions échouait justement pour des raisons de portée de variables. Nous avions défini une notion de : 'non présent dans la formule', mais cette notion disparaît hélàs complètement avec des règles comme cut, et donc cette forme de liaison apparait insuffisante. Il faudrait, par un moyen ou un autre, permettre de ressusciter cette notion, tout en préservant évidemment l'élimination des coupures.

Quatrième chapitre

Autre tentative

Les tentatives menées ayant toutes échouées, nous en revenons donc à l'idée évoquée par Dale Miller dans [5] , et qui consiste à travailler avec deux logiques : Une logique au niveau objet et une meta-logique. Toutefois, nous essaierons d'employer un codage plus simple que celui initialement proposé. Comme on pourra le voir ci-dessous, la forte liaison introduite par le quantificateur évite certains des problèmes ci-dessus rencontrés.

4.1 La syntaxe

Les différents symboles suivants seront utilisés

- Dans la meta-logique : les mêmes que précédemment
- Pour la logique objet
  - $\top : objformula$
  - $\bigwedge_\sigma : (\sigma \to objformula) \to objformula$
  - $\& : objformula \to objformula \to objformula$
  - $\triangleright : (evs \to objformula) \to formula$
- Pour utiliser evs
  - $\hat{\rho} : evs \to evs$
  - $\rho_\sigma : evs \to \sigma$

Comme on peut s'en apercevoir, le $\triangleright$ sert à faire la transition entre les deux logiques.

4.2 La logique

Ici, il s'agit juste d'etendre la logique, donc nous n'avons pas besoin de repréciser les différentes règles. Introduisons donc uniquement les nouvelles règles...

$$\frac{\Gamma, \triangleright B, \triangleright C \longrightarrow D}{\Gamma, \triangleright \lambda l (B\ l) \& (C\ l) \longrightarrow D} \& L \qquad \frac{\Gamma \longrightarrow \triangleright\ B \quad \Gamma \longrightarrow \triangleright\ C}{\Gamma \longrightarrow \triangleright\ \lambda l (B\ l) \& (C\ l)} \& R$$

$$\frac{\Gamma, \triangleright \lambda l (B\ \rho_\sigma l\ \hat{\rho} l) \longrightarrow D}{\Gamma, \triangleright \lambda l \bigwedge x (B\ x\ l) \longrightarrow D} \bigwedge L \qquad \frac{\Gamma \longrightarrow \triangleright\ \lambda l (B\ \rho_\sigma l\ \hat{\rho} l)}{\Gamma \longrightarrow \triangleright\ \lambda l \bigwedge x (B\ x\ l)} \bigwedge R$$

4.3 De l'intérêt d' evs

Comme on peut le deviner dans les règles, evs a un peu le rôle d'une pile, ou d'une liste, qui contiendrait les variables dites 'génériques'.Les deux autres solutions auxquelles il a été réfléchi sont :

- $\bigwedge$ provoque juste l'apparition d'une variable fraîche d'un ensemble à décrire. Cette solution est acceptable, mais la méthode retenue présente l'intérêt de 'lier' les variables au triangle : il ne s'agit pas de variables génériques pour toute l'expression, mais juste générique pour la sous-expression considérée. Et d'après ce qui précède, c'est bien cela que nous cherchons.
- Une autre forme de codage. La façon plus naturelle de coder le fait que B dépend de variables $x_1, \ldots x_n$ est certainement

  $$\lambda x_1 \ldots \lambda x_n (B\ x_1 \ldots x_n)$$

  Mais cette manière de procéder rend beaucoup plus difficile le 'matching' ; Lorsqu'il s'agit de déterminer quel règle doit s'appliquer, il faut en effet eliminer tous les quantificateurs avant d'avoir accès a la structure du terme.

Le meilleur choix semble donc être cette alternative, qui permet d'accéder rapidement à la structure, tout en liant de façon tout de même convenable les termes.

4.4 Définitions objets

Nous allons maintenant pouvoir introduire les définitions objets, et donc le véritable intérêt de $\triangleright$

définition 4.1 (*Définition objet*)

La définition d'un prédicat objet p est un ensemble d'objets de la forme

$$\forall \overline{d}[(p\ \overline{t}) \stackrel{\Omega}{=} (f\ \overline{d})]$$

où les variables présentes dans les t_i sont précisément les $d_i \in \mathcal{D}$, et f une fonction à valeurs dans $objformula$.

Cette définition est évidemment semblable à celle vu précédemment. La notion de substitution va elle changer :

définition 4.2 (*Substitution*)

Une substitution de définition objet φ est la donnée d'une fonction de $\mathcal{D}$ dans $evs \to \sigma$ L'application d'une substitution à un terme C est définie récursivement par :

- $(C\&D)\varphi = \lambda l(C\varphi\ l \& D\varphi\ l)$
- $(\bigwedge x, C\ x)\varphi = \lambda l(\bigwedge x, (C\ x)\varphi\ l)$
- $(\top)\varphi = \lambda l \top$
- $d\varphi = \lambda l(\varphi(d)\ l)$, $d \in \mathcal{D}$
- $(p\ x_1 \ldots x_n)\varphi = \lambda l(p\ (x_1\varphi\ l) \ldots (x_n\varphi\ l))$

Comme précédemment, on aura alors :

$$\frac{\Gamma \longrightarrow \rhd B\varphi}{\Gamma \longrightarrow \rhd A}\, def\Omega\ R$$

pour une définition $H \overset{\Omega}{=} B$ telle que $H\varphi = A$

$$\frac{\left\{\Gamma\theta, \rhd B\varphi \longrightarrow C\theta, \text{Pour tous } \varphi, \theta, \text{ et definitions } H \overset{\Omega}{=} B \text{ si } H\varphi = A\theta\right\}}{\Gamma, \rhd A \longrightarrow C}\, def\Omega\ L$$

4.5 L'exemple d'application

Reprenons l'exemple qui nous intéresse :

$$\left\{\begin{array}{l}(p\ X\ X\ Y) \overset{\Omega}{=} \top \\ (p\ X\ Y\ X) \overset{\Omega}{=} \top \\ (p\ Y\ X\ X) \overset{\Omega}{=} \top\end{array}\right. \quad eq\ X\ X \overset{\triangle}{=} \top$$

Essayons maintenant d'établir la proposition :

$$\longrightarrow \forall t_1, t_2, t_3, (\rhd \lambda l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3))) \supset (eq\ t_1\ t_2)$$

Après plusieurs applications des règles classiques, nous arrivons à la forme plus courtoise :

$$\rhd \lambda l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3)) \longrightarrow (eq\ t_1\ t_2)$$

Puis l'utilisation du quantificateur $\bigwedge$ conduit à :

$$\rhd \lambda l \bigwedge y_2 (p\ (pr\ \rho_\sigma l\ t_1)\ (pr\ \rho_\sigma l\ t_2)\ (pr\ y_2\ t_3)) \longrightarrow (eq\ t_1\ t_2)$$

puis

$$\rhd \lambda l (p\ (pr\ \rho_\sigma \hat{\rho} l\ t_1)\ (pr\ \rho_\sigma \hat{\rho} l\ t_2)\ (pr\ \rho_\sigma l\ t_3)) \longrightarrow (eq\ t_1\ t_2)$$

Utilisons maintenant $def\Omega\ L$. Cherchons donc par exemple à chercher une substitution à partir du terme

$$(p\ X\ Y\ X) \overset{\Omega}{=} \top$$

On cherche donc θ et φ tels que :

$$(p\ X\ Y\ X)\varphi = \lambda l (p\ (pr\ \rho_\sigma \hat{\rho} l\ t_1)\ (pr\ \rho_\sigma \hat{\rho} l\ t_2)\ (pr\ \rho_\sigma l\ t_3))\theta$$

Ceci se réécrit :

$$(p\ X\ Y\ X)\varphi = \lambda l (p\ (pr\ \rho_\sigma \hat{\rho} l\ t_1\theta)\ (pr\ \rho_\sigma \hat{\rho} l\ t_2\theta)\ (pr\ \rho_\sigma l\ t_3\theta))$$

(l'astuce est bien là : une substitution θ ne peut pas 'toucher' aux variables $\rho_\sigma l$) et donc ici aucune unification n'est possible. Toutefois avec la règle :

$$(p\ X\ X\ Y) \overset{\Omega}{=} \top$$

qui donne :

$$(p\ X\ X\ Y)\varphi = \lambda l(p\ (pr\ \rho_\sigma \hat{\rho} l\ t_1\theta)\ (pr\ \rho_\sigma \hat{\rho} l\ t_2\theta)\ (pr\ \rho_\sigma l\ t_3\theta))$$

il suffit de prendre :

$$\varphi : \left\{ \begin{array}{l} X \mapsto (pr\ \rho_\sigma \hat{\rho} l\ t_1) \\ Y \mapsto (pr\ \rho_\sigma l\ t_3) \end{array} \right. \quad \theta : t_2 \mapsto t_1$$

Et on obtiendra donc :

$$\triangleright \lambda l \top \longrightarrow (eq\ t_1\ t_1)$$

séquent à partir duquel il est aisé de conclure.

Comme le lecteur s'en est vite aperçu, la notation $\rho_\sigma \hat{\rho} l$ est pénible, ainsi, pour une plus grande compréhension, nous noterons dans toute la suite $\sharp 2$ pour $\rho_\sigma \hat{\rho} l$ et $\sharp 1$ pour $\rho_\sigma l$ (la 'variable' liée au λ s'appelera toujours l). Nous laissons au lecteur le soin de généraliser cette notation en introduisant $\sharp n$ pour tout $n > 0$. De même, il sera désormais noté $\triangleright_l$ plutôt que $\triangleright \lambda l$.

Le lecteur pourra se rapporter à l'appendice A.2 pour une preuve détaillée.

4.6 Intérêt et limitations

Résumons les propriétés de ce codage :

– Une forte liaison entre les variables, via la mise en place d'un opérateur $\triangleright$ et d'un codage considérant les 'variables' génériques comme totalement distinctes et 'intouchables' par les variables plus classiques
– Nous avons élimination des coupures, et sans même avoir à prouver quoi que ce soit ! En effet, tout ce que nous avons fait ci-dessus est directement codable dans la logique de départ. Il suffit d'ajouter

$$\triangleright_l((A\ l)\&(B\ l)) \stackrel{\triangle}{=} \triangleright_l(A\ l), \triangleright_l(B\ l)$$
$$\triangleright_l(\bigwedge x(A\ x\ l)) \stackrel{\triangle}{=} \triangleright_l(A\ \rho_\sigma l\ \hat{\rho} l)$$

et chaque définition objet de la forme

$$\bigwedge \overline{d}(p\ \overline{t}) \stackrel{\Omega}{=} (f\ \overline{d})$$

devient :

$$\forall \overline{d} \triangleright (p\ \overline{(t\ l)}) \stackrel{\triangle}{=} \triangleright (f\ \overline{(d\ l)})$$

(remarquer que d est ici une fonction, il est alors nécessaire, comme il a été indiqué précédemment , de considérer maintenant que nous pouvons être en présence de plusieurs types de données) Et dans ce cas, l'application des règles $\bigwedge$, $\&$ et $def\,\Omega$ correspond à l'application de $def\ L$ et $def\ R$! Donc nous sommes exactement dans le contexte de la logique étudiée précédemment et nous sommes sûr d'avoir l'elimination des coupures...
– L'atomicité est de nouveau présente, comme les quelques lignes ci-dessous le montrent :

$$\dfrac{\dfrac{\dfrac{}{\triangleright_l (A\ \sharp 1) \longrightarrow \triangleright_l\ (A\ \sharp 1)}\ init}{\triangleright_l \bigwedge x(A\ x) \longrightarrow \triangleright_l\ (A\ \sharp 1)}\ \bigwedge L}{\triangleright_l \bigwedge x(A\ x) \longrightarrow \triangleright_l \bigwedge x(A\ x)}\ \bigwedge R$$

Cependant, même si l'atomicité est conservée, la liaison permet tout de même quelques incohérences, comme par exemple le fait que les séquents :

$$\triangleright_l \bigwedge x \bigwedge y(A\ x\ y) \longrightarrow \triangleright_l \bigwedge y \bigwedge x(A\ x\ y)$$

et

$$\triangleright_l \bigwedge x \bigwedge y(A\ x) \longrightarrow \triangleright_l \bigwedge x(A\ x)$$

ne soient pas prouvables. Ils conduisent en effet naturellement à :

$$\triangleright_l (A\ \sharp 2\ \sharp 1) \longrightarrow \triangleright_l\ (A\ \sharp 1\ \sharp 2)$$

et

$$\triangleright_l (A\ \sharp 2) \longrightarrow \triangleright_l\ (A\ \sharp 1)$$

séquents à partir desquels nous ne pouvons conclure.

4.7 Vers un notion de swapping

Cette dernière difficulté trouve peut-être une réponse dans un article d'Andrew Pitts ([9]) qui suggère une logique des noms, dans lesquels il n'existe pas de substitution, mais uniquement des échanges de noms. Nous allons donc essayer de transposer cette notion dans notre calcul.

définition 4.3 (*Permutation*)

Une permutation ξ est une bijection de $\mathbf{N}^\star$ dans $\mathbf{N}^\star$. On définit alors l'application d'une permutation à un objet $\lambda l(C\ l)$ par : :

$$\begin{array}{lcr}
\lambda l((C\ l)\&(D\ l))\xi & \mapsto & \lambda l(C\xi\ l\&D\xi\ l) \\
\lambda l(\bigwedge xC\ x\ l)\xi & \mapsto & \lambda l(\bigwedge x(C\ x)\xi\ l) \\
(\lambda l\top)\xi & \mapsto & \lambda l\top \\
\lambda l(p\ (x_1\ l)\dots(x_n\ l))\xi & \mapsto & \lambda l(p\ (x_1\xi\ l)\dots(x_n\xi\ l)) \\
\lambda l(\rho_\sigma\hat{\rho}^n l)\xi & \mapsto & \lambda l(\rho_\sigma\hat{\rho}^{\xi(n)} l)
\end{array}$$

ou encore

$$\begin{array}{lcr}
(\sharp n)\xi & \mapsto & \sharp(\xi(n)) \\
X\xi & \mapsto & X\ \textit{sinon}
\end{array}$$

On peut désormais ajouter à nos règles :

$$\frac{\Gamma \longrightarrow \rhd C\xi}{\Gamma \longrightarrow \rhd C}\ \xi\ R$$

et

$$\frac{\Gamma, \rhd C\xi \longrightarrow A}{\Gamma, \rhd C \longrightarrow A}\ \xi\ L$$

C'est à dire : On peut toujours échanger des variables génériques.

Et les deux problèmes évoqués ci-dessus ont disparus ! En effet, A partir de la ligne

$$\triangleright_l (A \sharp 2 \sharp 1) \longrightarrow \triangleright_l (A \sharp 1 \sharp 2)$$

on peut désormais faire la permutation

$$\xi : \begin{cases} 1 & \mapsto 2 \\ 2 & \mapsto 1 \\ x > 2 & \mapsto x \end{cases}$$

et appliquer $\xi\ R$ pour obtenir :

$$\triangleright_l (A \sharp 2 \sharp 1) \longrightarrow \triangleright_l (A \sharp 2 \sharp 1)$$

et conclure ! La même permutation permet également de conclure dans le cas de

$$\triangleright_l (A \sharp 2) \longrightarrow \triangleright_l (A \sharp 1)$$

Il faudrait toutefois vérifier que cette nouvelle règle ne fait pas perdre la propriété d'élimination des coupures. Il suffit pour s'en convaincre de remarquer que

$$\cfrac{\cfrac{\Gamma \longrightarrow \triangleright C\xi}{\Gamma \longrightarrow \triangleright C}\xi\ R \qquad \cfrac{\Gamma, \triangleright C\Xi \longrightarrow A}{\Gamma, \triangleright C \longrightarrow A}\xi\ L}{\Gamma \longrightarrow A}\mathit{cut}$$

se transforme naturellement en

$$\cfrac{\Gamma \longrightarrow \triangleright C\xi \qquad \cfrac{\Gamma, \triangleright C\xi(\xi^{-1} \circ \Xi) \longrightarrow A}{\Gamma, \triangleright C\xi \longrightarrow A}\xi\ L}{\Gamma \longrightarrow A}\mathit{cut}$$

C'est la principale règle qui permet de comprendre pourquoi l'élimination des coupures va marcher. En fait, on peut remarquer qu'il suffit, au lieu d'introduire ces règles bien trop compliquées, d'utiliser uniquement une règle $init\xi$:

$$\cfrac{}{\Gamma, \triangleright_l A\xi \longrightarrow \triangleright_l A}init\xi$$

et de supprimer les règles ξ ci-dessus mentionnées.

On peut alors prouver que de toute preuve de $\Gamma \longrightarrow C$ qui utilise les règles ξ, on peut déduire une preuve qui utilise uniquement $init\xi$.

La preuve se fait par induction, et en fait il suffit de remarquer qu'on peut toujours changer la règle

$$\cfrac{\Gamma \longrightarrow \triangleright C\xi}{\Gamma \longrightarrow \triangleright C}\xi\ R$$

en

$$\cfrac{\Gamma \longrightarrow \triangleright C\xi \qquad \cfrac{}{\Gamma \triangleright C\xi \longrightarrow \triangleright C}init\xi}{\Gamma \longrightarrow \triangleright C}\mathit{cut}$$

et de même

$$\cfrac{\Gamma, \triangleright C\xi \longrightarrow A}{\Gamma, \triangleright C \longrightarrow A}\xi\ L$$

devient

$$\cfrac{\cfrac{}{\Gamma, \triangleright C \longrightarrow \triangleright C\xi}\,init\Omega \qquad \cfrac{\Gamma, \triangleright C\xi \longrightarrow A}{\Gamma, \triangleright C, \triangleright C\xi \longrightarrow A}\,weaken}{\Gamma, \triangleright C \longrightarrow A}\,cut$$

L'étape suivante de demonstration est le fait intéressant suivant :

Lemme 4.1

De toute preuve de $\Gamma \longrightarrow A$, on peut déduire un preuve de $\Gamma \longrightarrow A$ dont toutes les étapes de preuves dont la conclusion est $\Delta \longrightarrow C \wedge D$ sont telles que la règle utilisée est $\wedge R$.

Ce lemme signifie : Dès que la formule à prouver est de la forme $A \wedge B$, on peut appliquer $\wedge R$. La démonstration se fait par induction et n'est pas d'une grande difficulté.

Le fait important est alors que la réduction définie dans [3] ne modifie pas cette propriété.

On démontre alors ensuite de manière naturelle que :

Lemme 4.2

Sous la condition du lemme 4.1, de toute preuve Π de $\Gamma \longrightarrow \triangleright C$ on peut déduire une preuve $\Pi\xi$ du séquent $\Gamma \longrightarrow \triangleright C\xi$ qui a le même arbre de preuve.

La preuve d'élimination des coupures se fait alors immédiatement :

- On peut tout d'abord supposer que toutes nos preuves vérifient la propriété d'atomicité du lemme 4.1.
- On remarque alors que si on prend la preuve d'élimination des coupures de [3] et que l'on considére la règle d'élimination locale du cut pour *init*, à savoir la transformation de

$$\cfrac{\cfrac{\Pi}{\Gamma \longrightarrow C} \qquad \Gamma, C \longrightarrow C}{\Gamma \longrightarrow C}$$

en

$$\cfrac{\Pi}{\Gamma \longrightarrow C}$$

on peut en déduire la règle de transformation de

$$\cfrac{\cfrac{\Pi}{\Gamma \longrightarrow \triangleright C\xi} \qquad \Gamma, \triangleright C\xi \longrightarrow \triangleright C}{\Gamma \longrightarrow \triangleright C}$$

en

$$\cfrac{\Pi\xi^{-1}}{\Gamma \longrightarrow \triangleright C}$$

transformation *justifiée par le lemme 4.2*, et on est alors sûr que cette règle ne va pas contrarier la terminaison du procédé puisqu'elle agit *exactement* comme la règle *init*. Et la preuve d'elimination est ainsi terminée.

Ainsi, cet ajout dans la logique de possibilités de permutation semble bien cohérent et permet de garantir des propositions qui sans elles semblaient fausses.

Conclusion

Nous avons ici essayé de définir la notion de généricité, et la façon de la capturer dans le calcul des séquents. Un codage direct, à l'aide d'un nouveau quantificateur, n'a hélàs souvent que peu donné satisfaction, celui-ci se révélant souvent inefficace à traduire la notion souhaitée, ou hélàs source d'incohérence dans un calcul qui en est pourtant initialement dépourvu

Ensuite, grâce à un encodage à l'aide de deux logiques distinctes, nous avons réussi à traduire de façon convenable la notion de généricité que nous voulions mettre en place Toutefois, l'encodage de propriété plus forte, comme par exemple, essayer d'implanter un $\supset$ dans la logique objet pourrait hélàs dans un tel concept être difficile à manier, nous laisserons donc au lecteur interessé qui cherche un sujet de stage le soin de répondre à cette question

Travail effectué

Mis à part la première partie décrivant la logique des séquents, toutes les idées de ce document ont été apportées par son auteur. Celui-ci travailla aussi sur une notion de généricité beaucoup plus forte, mais qu'il oublia rapidement puisqu'éloigné du contexte qu'il étudiait.

Le stagiaire ajouta également quelques fonctions au système Iris, afin de le rendre plus agréable. Néanmoins, il ne lui fut pas possible d'implanter la logique qu'il s'est proposé de décrire, en raison de problèmes sous-jacents au logiciel en question, la perte des sources du parseur en étant la principale cause. Toutefois, si le temps et l'envie le permettent, l'implantation devrait être assez aisée, une fois cette difficulté contournée.

Remerciements

Je remercie les disquaires de State College pour leur grande diversité, le patron du Taco Bell pour sa jovialité, ma colocataire pour m'avoir supporté pendant deux mois, Rio pour la qualité de ses lecteurs CD/MP3, Undernet pour ses serveurs IRC, Emmanuel Beffara pour son package LaTeX dont vous venez d'apprécier la saveur, Guido Van Rossum pour avoir inventé le meilleur langage de script à l'heure actuelle, le Hammerstein Ballroom de N.Y pour la qualité de ses concerts pendant l'été, et bien entendu Dale Miller, Jeremie Wajs et toute l'équipe du Pond Lab pour m'avoir accueilli.

Je regretterai toutefois l'absence de climatisation le week end, l'absence d'une machine à café dans le laboratoire (!), l'absence de nuages sous ce climat bien trop chaud, l'absence d'horlogers pour remplacer la pile de ma montre, et l'absence de parcs dignes d'intérêt dans une si petite ville.

annexe A
Preuves complètes

FIG. A.1 – append et du bon usage des définitions (voir 2.4)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\top \longrightarrow \top}\ \top R}{\top \longrightarrow (append\ nil\ (cons\ x\ T)\ (cons\ x\ T))}\ def\ R}{(append\ nil\ M\ T) \longrightarrow (append\ nil\ (cons\ x\ M)\ (cons\ x\ T))}\ def\ L}{(append\ (cons\ x\ nil)\ M\ L) \longrightarrow (append\ nil\ (cons\ x\ M)\ L)}\ def\ L}{\longrightarrow (append\ (cons\ x\ nil)\ M\ L) \supset (append\ nil\ (cons\ x\ M)\ L)}\ \supset R}{\longrightarrow \forall M (append\ (cons\ x\ nil)\ M\ L) \supset (append\ nil\ (cons\ x\ M)\ L)}\ \forall R}{\longrightarrow \forall L \forall M (append\ (cons\ x\ nil)\ M\ L) \supset (append\ nil\ (cons\ x\ M)\ L)}\ \forall R}{\longrightarrow \forall y \forall L \forall M (append\ (cons\ x\ nil)\ M\ L) \supset (append\ nil\ (cons\ x\ M)\ L)}\ \forall R}{\longrightarrow \forall x \forall y \forall L \forall M (append\ (cons\ x\ nil)\ M\ L) \supset (append\ nil\ (cons\ x\ M)\ L)}\ \forall R
$$

FIG. A.2 – L'exemple classique utilisant le quantificateur générique (voir 4.5)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\rhd_l \top \longrightarrow \top}\ \top R}{\rhd_l \top \longrightarrow (eq\ t_1\ t_1)}\ def R}{\rhd_l (p\ (pr\ \sharp 2\ t_1)\ (pr\ \sharp 2\ t_2)\ (pr\ \sharp 1\ t_3)) \longrightarrow (eq\ t_1\ t_2)}\ def\Omega\ L}{\rhd_l \bigwedge y_2 (p\ (pr\ \sharp 1\ t_1)\ (pr\ \sharp 1\ t_2)\ (pr\ y_2\ t_3)) \longrightarrow (eq\ t_1\ t_2)}\ \bigwedge L}{\rhd_l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3)) \longrightarrow (eq\ t_1\ t_2)}\ \bigwedge L}{\longrightarrow (\rhd_l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3))) \supset (eq\ t_1\ t_2)}\ \supset R}{\longrightarrow \forall t_3, (\rhd_l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3))) \supset (eq\ t_1\ t_2)}\ \forall R}{\longrightarrow \forall t_2, t_3, (\rhd_l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3))) \supset (eq\ t_1\ t_2)}\ \forall R}{\longrightarrow \forall t_1, t_2, t_3, (\rhd_l \bigwedge y_1 \bigwedge y_2 (p\ (pr\ y_1\ t_1)\ (pr\ y_1\ t_2)\ (pr\ y_2\ t_3))) \supset (eq\ t_1\ t_2)}\ \forall R
$$

FIG. A.3 – Non-Élimination des coupures (voir 3.5)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{A\,x, A\,x \supset B\,y \longrightarrow B\,y}}{A\,x, \bigwedge z(A\,z \supset B\,y) \longrightarrow B\,y}}{A\,x \longrightarrow (\bigwedge z(A\,z \supset B\,y)) \supset B\,y}}{A\,x \longrightarrow \bigwedge y \bigwedge z(A\,z \supset B\,y)) \supset B\,y)} \qquad \dfrac{A\,x, \bigwedge z((A\,z) \supset B\,x)) \supset B\,x) \longrightarrow (\bigwedge z((A\,z) \supset B\,x)) \supset B\,x)}{A\,x, \bigwedge y \bigwedge z(A\,z \supset B\,y)) \supset B\,y) \longrightarrow (\bigwedge z((A\,z) \supset B\,x)) \supset B\,x)}}{A\,x \longrightarrow (\bigwedge z(A\,z \supset B\,x)) \supset B\,x}}{\longrightarrow (A\,x \supset (\bigwedge z(A\,z \supset B\,x)) \supset B\,x)}}{\longrightarrow \bigwedge x(A\,x \supset (\bigwedge z(A\,z \supset B\,x)) \supset B\,x)}
$$

Bibliographie

[1] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 214–224, Trento, Italy, 1999. IEEE Computer Society Press.

[2] Gerhard Gentzen. *The Collected Papers of Gerhard Gentzen*. North Holland, 1969.

[3] R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction, 2000.

[4] D. Miller. Lambdaprolog : An introduction to the language and its logic, 1997.

[5] D. Miller. Encoding generic judgments, 2001.

[6] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4) :497–536, 1991.

[7] R. Milner. Communicating and mobile systems : the pi-calculus, 1999.

[8] Gopalan Nadathur. A higher-order logic as the basis for logic programming. Technical Report DUKE–TR–1990–11, 1, 1990.

[9] A. Pitts. A first order theory of names and binding, 2001.

[10] J. Wajs. Design and implementation of a theorem prover for operational semantics, 2000.