République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed Bouguerra - Boumerdès

Faculté des Sciences
Département d'Informatique

Mémoire

*Pour l'obtention du diplôme de*

**MAGISTER EN INFORMATIQUE**

***Option* : « *INFORMATIQUE FONDAMENTALE* »**

*Présenté et soutenu publiquement par*

**HAMIMED Lyazid**

***Thème :***

# Un Système de Preuve d'Ordre Supérieur Basé sur le E_Lambda calcul

**JURY**

| | | |
|---|---|---|
| Mr AHMED-NACER M. | Pr. USTHB | Président |
| Mr ZEGOUR D.E | Pr I.N.I | Examinateur |
| Mr KHALIFAT M. | Cc. UMBB | Examinateur |
| Mr MEZEGHICHE M. | Pr. UMBB | Rapporteur |

*2006*

# Remerciements.

Mes sincères remerciements vont au Pr. Mohamed Mezghiche pour la proposition, l'encadrement de ce travail, ses précieux conseils et pour sa grande patience à la relecture de mes rapports.

Mes remerciements s'adressent à Monsieur Mohamed AHMED NACER, Professeur de L'USTHB, pour l'intérêt qu'il a accordé à ce travail en acceptant de le juger et de présider le jury.

Mes remerciements s'adressent également aux membres du jury : Monsieur Djamel Edinne ZEGOUR, Prof de l'INI et Monsieur Mohamed KHALIFAT pour avoir accepté de participer à l'évaluation de ce travail.

Mes sincères remerciements s'adressent également à ma famille et à ma future compagne pour leurs soutient durant la période de mes études.

Je tiens à remercier mes amis G. TOUIL, C. SALEMI, M. CHAABANI, A. BADAOUI, S. KERRACHE, H. AMROUCHE, Y. LALAOUI pour leurs lecture, correction et aide à la réalisation de ce travail.

Je remercie également tous mes collègues et mes amis de l'UMBB, l'INI, et l'IAP.

# Résumé

La majorité des systèmes de preuve existants sont basés sur le paradigme « type as formula », due au typage du lambda calcul et plus précisément inspiré de l'interprétation de la relation des termes avec leurs types. Cette interprétation, connue souvent l'isomorphisme de Curry Howard, consiste à considérer les types comme étant des propositions et les termes comme étant des preuves.

Le Eλ-calcul est une extension du lambda calcul pur, où une nouvelle procédure du processus de calcul est définie (la Eλβ-réduction) et deux constantes sont introduites : une de ces constantes est destinée à représenter l'implication et l'autre pour représenter la quantification universelle. Le système obtenu est assez puissant (le processus de calcul, la Eλβ-réduction, vérifie la propriété de Church-Rosser et offre un moyen pour éviter le paradoxe de Curry) et peut être utilisé pour interpréter les logiques d'ordre supérieur [MC02].

Le but de ce mémoire est d'étudier et de définir une procédure de preuve automatique dans le système E-lambda.

**Mots clés :** λ-calcul, systèmes de preuve, logique d'ordre supérieur, déduction naturelle, calcul des séquents. Paradoxe de Curry.

# Abstract.

The existing proof systems are all based on the paradigm " type-as-formula", induced from the typed lambda calculus and more precisely inspired from the interpretation of the relation between terms and their types. This interpretation, known often as the Curry Howard isomorphism, consists of considering types as formulas and terms as proofs. The E_lambda calculus is an extension to the pure λ-calculus, where a new procedure of the calculation process is defined (the Eλβ-Reduction) and two constants are introduced: one to represent the logic implication and the other for the universal quantification. The obtained system is consistent (it verifies the property of Church-Rosser and avoids the Curry paradox) and can be used to interpret higher order logics [MC 02].

The aim of this work is to study and to define an automatic proof procedure for the Eλ calculus.

**Key words:** λ-calculus, Proof assistants, Higher order logic, Natural deduction, Sequent calculus, Curry paradox.

# Table des Matières

INTRODUCTION ........ 3

1 LE LAMBDA CALCUL ........ 8

1.1 LAMBDA CALCUL PUR ........ 8
1.1.1 Syntaxe du langage ........ 8
1.1.2 Variables Libres et Variables Liées ........ 9
1.1.3 Processus de substitution ........ 10
1.1.4 Processus de réduction ........ 11
1.1.5 Terminaison et confluence ........ 11
1.2 LE LAMBDA CALCUL SIMPLEMENT TYPE ........ 11
1.2.1 Syntaxe ........ 12
1.2.2 Système de types ........ 12
1.2.3 Normalisation forte ........ 14
1.3 LOGIQUE CONSTRUCTIVE ........ 14
1.3.1 La logique minimale ........ 14
1.3.2 Le Calcul des Prédicats ........ 15
1.3.3 Sémantique de Heyting ........ 17
1.3.4 L'isomorphisme de Curry Howard ........ 18
1.4 EXTENSIONS DU LAMBDA CALCUL TYPE ........ 19
1.4.1 Système polymorphique du second ordre λ2 ........ 19
1.4.2 Calcul des constructions ........ 20
1.5 LOGIQUE ET CONCEPTS MATHEMATIQUES DANS LE LANGAGE DES TYPES ........ 21
1.5.1 Connecteurs et Quantificateurs ........ 21
1.5.2 Représentation de l'arithmétique ........ 23
1.5.3 Représentation des ensembles ........ 24

2 SYSTEMES DE PREUVE ........ 25

2.1 DEDUCTION NATURELLE ........ 25
2.1.1 Contextes et jugements : ........ 26
2.1.2 Règles de la déduction naturelle ........ 26
2.1.3 Formalisation du processus de dérivation ........ 28
2.2 LE CALCUL DES SEQUENTS ........ 29
2.2.1 Jugements du calcul des séquents ........ 29
2.2.2 Les règles du calcul des séquents ........ 30
2.3 QUELQUES SYSTEMES DE PREUVE ........ 33
2.3.1 Coq ........ 33
2.3.2 Agda et Alfa ........ 35
2.3.3 Isabelle/HOL ........ 36
2.3.4 Phox ........ 37

2.3.5 PVS (Prototype Verification System) ........ 38
2.3.6 ACL2 ........ 40
**2.4 FOLDROL ........ 41**
2.4.1 Principe de base ........ 41
2.4.2 Des preuves à la main ........ 42
2.4.3 Règles d'inférences ........ 42
2.4.4 L'inférence ........ 46
2.4.5 Limites du Système FOLDROL ........ 49
2.4.6 Tactiques et Méthodes Automatiques ........ 49

**3 LE SYSTEME E_LAMBDA ........ 51**

**3.1 LES TERMES DU SYSTEME EΛ ........ 51**
*Définition 3.1 Les termes de l'ensemble $C_0$* ........ 52
*Définition 3.2 Les termes d'un ensemble $C_i$* ........ 52
*Définition 3.3 les termes du système Eλ « $C_w$ »* ........ 52
*Définition 3.4 (niveau d'un Eλ-terme)* ........ 52
*Définition 3.5 (la congruence)* ........ 53
*Définition 3.6 (Eλβ-réduction )* ........ 53
**3.2 PROCESSUS D'INFERENCE ........ 53**
**3.3 LES CONNECTEURS & LES QUANTIFICATEURS LOGIQUES ........ 54**
*Proposition 3.1* ........ 55
*Proposition 3.2* ........ 56
**3.4 LA CONSISTANCE DU SYSTEME EΛ ........ 56**
3.4.1 Le paradoxe de Curry ........ 56

**4 LE DEMONSTRATEUR EΛ-PROVER ........ 57**

**4.1 PROCEDURE DE DEMONSTRATION ........ 57**
4.1.1 Transformation de la formule à prouver ........ 58
4.1.2 Construction des buts élémentaires ........ 60
*Définition 4.1 (le processus d'unification)* ........ 61
*Propriété 4.1* ........ 61
*Propriété 4.2* ........ 61
*Définition 4.2 (sous formule)* ........ 61
*Définition 4.3 (ensemble de variables)* ........ 61
*Définition 4.4 (paramètres d'un habitant)* ........ 61
*Définition 4.5 (les règles de substitutions σ)* ........ 62
*Définition 4.6 (ordre d'une relation)* ........ 62
*Définition 4.7 (ordre d'un habitant)* ........ 62
*Définition 4.8 (l'univers HABIT associée à un type)* ........ 62
*Propriété 4.3* ........ 63
**4.2 EXEMPLES : ........ 64**

***CONCLUSION ........ 71***

***BIBLIOGRAPHIE ........ 73***

***ANNEXE ........ 75***

# Table des Figures

FIGURE 1-1 LES REGLES DE TYPAGE ........ 13

FIGURE 1-2 L'ISOMORPHISME DE CURRY HOWARD ........ 15

FIGURE 1-3 REGLES DE LA LOGIQUE DES PREDICATS ........ 17

FIGURE 1-4 SYNTAXE FORMELLE DU SYSTEME ∧2 ........ 19

FIGURE 1-5 REGLES D'INFERENCE DU SYSTEME ∧2 ........ 19

FIGURE 1-6 SYNTAXE FORMELLE DU CALCUL DES CONSTRUCTIONS ........ 20

FIGURE 1-7 REGLES D'ASSIGNATION DES TYPES DANS LE CALCUL DES CONSTRUCTIONS ........ 20

FIGURE 2-1 REGLES DE LA DEDUCTION NATURELLE ........ 27

FIGURE 2-2 DEDUCTION NATURELLE EN FORMAT DE SEQUENTS ........ 29

FIGURE 2-3 REGLES POUR LA LOGIQUE PROPOSITIONNELLE ........ 43

FIGURE 2-4 LES REGLES QUANTIFIEES ........ 43

# Introduction

La majorité des systèmes de preuve existants (Coq, Agda et Alfa, Isabelle/HOL, PVS) sont basés sur le paradigme *type-as-formula*, due au typage du lambda calcul et plus précisément inspiré de l'interprétation de la relation des termes avec leurs types. Cette interprétation, connue souvent sous l'isomorphisme de Curry Howard, consiste à considérer les types comme des propositions et les termes comme des preuves.

Prouver un énoncé, en utilisant un système de preuve basé sur ce paradigme, revient à exprimer l'énoncé à démontrer par une formule de la logique considérée, puis le système se charge à synthétiser le terme qui correspond à la preuve. Si le système réussi à construire un terme dont le type est la proposition en question, alors cette proposition est dite prouvée et ce terme est considéré comme sa preuve. Dans la majorité des cas, la logique manipulée est une logique constructive d'ordre supérieur, une logique sous laquelle toute preuve de l'existence d'un objet offre un moyen pour le construire.

Le système E-Lambda [MC 02] est obtenu en étendant le lambda calcul pur par les deux constantes $\supset et\ \pi$ qui sont destinées à représenter respectivement l'implication et la quantification universelle. Cette extension a nécessité une nouvelle définition du processus de calcul. Il est démontré que le Eλ-Calcul peut interpréter la logique d'ordre supérieur.

Le but de notre travail est d'étudier une procédure de démonstration automatique pour les preuves dans le système Eλ.

Ce document décrit le travail que nous avons développé pour étudier ce problème.

Dans le chapitre 1 nous présenterons quelques notions préliminaires de la théorie du lambda calcul, où nous allons présenter les concepts de base utiles pour la compréhension de la suite de notre mémoire (la théorie du lambda calcul, la théorie des types simples et la représentation de la logique et des concepts de base des mathématiques dans ces théories).

Le chapitre 2 est consacré à la présentation des outils les plus utilisés dans la mécanisation d'un système de déduction. Dans cette partie nous allons présenter la déduction naturelle et le calcul des séquents, suivie par une présentation succincte de quelques systèmes de preuve et notamment le démonstrateur pédagogique FOLDROL [Law 92].

Dans le chapitre 3, nous présenterons le système E-lambda. Nous rappelons quelques propriétés de ce système, notamment sa richesse, sa consistance et la propriété d'interpréter la logique d'ordre supérieur.

Nous proposons dans le chapitre 4 une procédure de preuve automatique dans le système E-lambda. Une implémentation de cette procédure a été réalisée.

1 Le Lambda Calcu

**Chapitre I**

# Théorie du λ-Calcul

Dans la première partie de ce mémoire, nous allons présenter les concepts de base utile pour la compréhension de la suite de notre mémoire. Nous rappelons très brièvement la théorie du lambda calcul [Chu33, Bar 84], la théorie des types simples et la représentation de la logique et des concepts de base des mathématiques dans ces théories [Sel 97].

## 1.1 Lambda Calcul pur

Inventé par Church, le lambda-calcul est une formalisation de l'écriture des fonctions. Par exemple l'écriture suivante :

$$\lambda x.\lambda y.x$$

est une expression du lambda calcul qui dénote la fonction qui prend deux arguments $x$ et $y$ et qui retourne le premier argument comme résultat (la valeur de x).

### 1.1.1 Syntaxe du langage

Les expressions du lambda calcul, appelées souvent les lambda-termes, sont générées par la grammaire contexte libre suivante :

$$\begin{aligned} <Term> \;\rightarrow\;& Var \\ & \mid \lambda\, Var. <Term> \\ & \mid <Term><Term> \end{aligned}$$

### Définition 1.1

Les termes du lambda calcul sont définis inductivement, sur un ensemble dénombrable de variables, de la manière suivante :

- Une variable $x$ est un lambda terme ;
- L'application de deux termes $M$ et $N$, notée par la juxtaposition « $MN$ », signifiant l'application du terme $M$ au terme $N$ est aussi un lambda terme ;
- l'abstraction d'une variable, notée : $\lambda x.M$ , signifiant la fonction qui à toute variable $x$ associe $M$, où $x$ peut apparaître dans « $M$ », est aussi un terme.

L'application des lambda termes est associative à gauche : par conséquent le terme $M\ N\ Z$ se lit $(M\ N)\ Z$.

La définition précédente suppose l'existence d'un ensemble non vide et dénombrable de variables. Les expressions suivantes sont des lambda termes : $\lambda x.x$, $\lambda x.xx$, $(\lambda x.xz)y$ …

Un redex est un terme sous la forme $(\lambda x.M)N$ (il s'agit d'une application d'une abstraction à un terme).

Les parenthèses sont aussi utilisées pour identifier correctement l'argument d'une application.

## 1.1.2 Variables Libres et Variables Liées

### Définition 1.2 *(Variables libres)*

Les variables libres d'un terme T sont définies inductivement sur la structure de ce dernier de la manière suivante :

i. Si x est une variable alors $VL(x) = \{x\}$

ii. Si x est une variable et N est un terme alors $VL(\lambda x.N) = VL(N) - \{x\}$

iii. Si N et M sont deux termes alors $VL(MN) = VL(M) \cup VL(N)$

### Définition 1.3 *(Variables liées)*

Une variable ayant une occurrence non libre dans une expression est dite *liée* dans cette expression. Les variables liées d'un terme T sont définies inductivement de la manière suivante :

i. Si x est une variable alors $BV(x) = \Phi$

ii. Si x est une variable et N est un terme alors $BV(\lambda x.N) = BV(N) \cup \{x\}$

iii. Si N et M sont deux termes alors $BV(MN) = BV(M) \cup BV(N)$

**Définition 1.4** *(Terme Clos)*

Une expression $T$ du lambda calcul est dite close (ou le terme $T$ est clos) si elle n'a pas de variable libre (dans ce cas $T$ est appelé aussi combinateur).

## 1.1.3 Processus de substitution

**Définition 1.5**

Le processus de substitution est défini inductivement de la manière suivante :

i. Si $M = x \textit{ alors } M[x/t] = t$

ii. Si $M \textit{ est variable et } M \neq x \textit{ alors } M[x/t] = M$

iii. Si $M = (UV) \textit{ alors } M[x/t] = (U[x/t])(V[x/t])$

iv. Si $M = (\lambda y.N) \textit{ et } x \notin VL(M) \textit{ alors } M[x/t] = \lambda y.N$

v. Si $M = (\lambda y.N) \textit{ et } y \notin VL(t) \textit{ alors } M[x/t] = \lambda y.(N[x/t])$

vi. Si $M = (\lambda y.N) \textit{ et } y \in VL(t) \textit{ alors } M[x/t] = \lambda z.(N[y/z])[x/t]$

$\textit{où } z \notin VL(M) \textit{ et } z \notin VL(M)$

Le terme $M[x/t]$ est le terme obtenu en remplaçant les occurrences libres de la variable x, du terme M, par le terme t.

**Définition 1.6** *(α-Equivalence)*

On dit que le terme $u$ est α-équivalent à $v$ et on note $u =_\alpha v$, si le terme v est obtenu en renommant quelques variables liées de u. formellement la *α-équivalence* peut être définie de la manière suivante :

i. Si u est une variable alors $v =_\alpha u$ si et seulement si $v = u$

ii. Si u est une abstraction ( $u = \lambda x.M$ ) alors $v =_\alpha u$ ssi $v = \lambda y.N$ et $M[x/t] =_\alpha N[y/t]$.

iii. Si $u$ est une application ( $u = MN$ )alors $v =_\alpha u$ ssi $v = M'N'$ et $M' =_\alpha M \textit{ et } N' =_\alpha N$.

### 1.1.4 Processus de réduction

**Définition 1.7** *(Bêta-Réduction)*

La béta-Réduction est utilisée comme règle de calcul. Cette règle exprime qu'un redex $(\lambda x.M)N$ se calcule en *« M' »* en remplaçant tous les occurrences libres de la variable *x*, dans le terme *M*, par le terme *N*.

La notion de réduction sur les termes du λ-calcul peut être vue comme l'application d'une fonction mathématique à un argument.

**Définition 1.8** *(Forme Normale)*

Un terme est dit sous forme normale s'il ne contient aucun redex (autrement dit : aucune règle de réduction n'est applicable à ce terme).

Si un terme *M* se réduit à un terme *N*, et ce dernier est en forme normale, alors on dira que *N* est une forme normale de *M*.

Si la forme normale d'un terme existe elle est unique.

### 1.1.5 Terminaison et confluence

La terminaison du calcul présenté précédemment n'est pas assurée. Ce problème est dû à l'existence des termes qui n'admettent pas de forme normale. Par exemple le processus de calcul appliqué au terme « $(\lambda x.xx)(\lambda x.xx)$ » ne se termine pas.

En revanche, le calcul est confluent : si $M \rightarrow_{\beta R} N_1$ *et* $M \rightarrow_{\beta R} N_2$ alors il existe un terme $Z$ tel que : $N_1 \rightarrow_{\beta R} Z$ *et* $N_2 \rightarrow_{\beta R} Z$ .

## 1.2 Le Lambda Calcul Simplement Typé

Remarquons qu'avec la définition 1.7, de la bêta-réduction définie sur les expressions du lambda calcul pur, il est possible d'appliquer un terme à un terme quelconque, ce qui peut aboutir à des résultats inconsistants. Pour éviter quelques anomalies de cette application, le lambda calcul typé peut intervenir [Dub 01]. Typer un lambda terme revient à déterminer son type dont le but de vérifier s'il est correct et dans quel contexte l'utiliser, et aussi pour assurer la terminaison du calcul [DL 03]. Ce phénomène permet de restreindre les expressions du langage à une classe particulière des fonctions.

Par exemple l'expression du lambda calcul simplement typé suivante :

$$\lambda(f : i \rightarrow i)\lambda(x : i).(fx)$$

où « *i* » est un type arbitraire. Cette expression dénote la fonction qui prend en argument « *f* » (une valeur fonctionnelle) de type « $i \to i$ » et « *x* » de type « *i* », et retourne le résultat de l'application « *fx* ».

Le λ-calcul simplement typé est un formalisme fortement lié à la déduction naturelle[1]. Avec une telle interprétation, les termes du λ-calcul (appelé souvent λ-termes) représentent les dérivations de la déduction naturelle ; c'est-à-dire à chaque nœud, de l'arbre de dérivation, correspond un λ-terme et les types des λ-termes représentent les formules de la déduction naturelle.

### 1.2.1 Syntaxe

Considérons un langage dont les termes contiennent des informations de type. Plus précisément, le type d'un paramètre sera défini lors de la construction d'une abstraction. Il s'agit d'une représentation du lambda calcul simplement typé dite *représentation à la Church.* Il est possible de garder les mêmes termes que ceux du lambda calcul pur (aucune information de type dans les termes), cette représentation est dite *à la Curry.*

Les types considérés sont un type de base *Var* et les types fonctionnels : $\tau \to \tau'$ (types des fonctions dont le paramètre est de type $\tau$ et le résultat de type $\tau'$).

$$T ::= i \quad | \; T \to T$$

La flèche est associative à droite : ainsi l'expression de type $T_1 \to T_2 \to T_3$ se lit $T_1 \to (T_2 \to T_3)$.

La syntaxe des termes du lambda calcul simplement typé est définie par la grammaire suivante :

$$\begin{aligned} <Term> \quad \to \quad & x : \alpha \\ & | \; \lambda x : \alpha . <Term> \\ & | \; <Term><Term> \end{aligned}$$

### 1.2.2 Système de types

Un système de types est composé d'un langage de types et d'un ensemble de règles de typage. Le langage de types a déjà été donné et les informations de type relatives aux variables non liées des expressions sont maintenues dans un environnement de typage.

---

[1] Pour plus de détails, consulter la section 2.1

Un environnement de typage peut être vu comme une fonction partielle des variables vers des types qui associe à une variable son type.

Les règles de typage décrivent, pour chaque construction du langage, les conditions qui permettront d'affirmer que cette construction est bien typée : par exemple la construction de l'application *« MN »* est bien typée si l'expression *« M »* est bien typée et a un type fonctionnel de la forme « $\tau \rightarrow \tau'$ » et si *« N »* est aussi bien typée et de type « $\tau$ ». Le prédicat « *EstBienTypé* » est défini inductivement en fonction de l'environnement de typage, le lambda terme et le type de ce dernier. Les séquents[1] de typage sont indiqués dans la figure suivante :

$$(VAR) \quad \Gamma \mid\!- x : \alpha$$

$$(ABS) \quad \frac{\Gamma \cup \{x : \alpha\} \mid\!- M : \beta}{\Gamma \mid\!- \lambda x.M : \alpha \rightarrow \beta}$$

$$(APP) \quad \frac{\Gamma \mid\!- M : \alpha \rightarrow \beta \quad \Gamma \mid\!- N : \alpha}{\Gamma \mid\!- MN : \beta}$$

**Figure 1-1 Les Règles de typage**

$\Gamma \mid\text{-}\, x : \alpha$ Désigne le type associe à la variable $x$ dans l'environnement $\Gamma$.

$\Gamma \cup \{x : \alpha\}$ pour étendre un environnement avec une nouvelle hypothèse (si x existe déjà dans $\Gamma$ alors son image serait remplacée par $\alpha$.

L'écriture : $\Gamma \mid\text{-}\, M : \alpha$ indique que le lambda terme « M » a le type $\alpha$ relativement à l'environnement $\Gamma$.

Un terme M a le type $\alpha$ dans un environnement $\Gamma$ si le séquent $\Gamma \mid\text{-}\, M : \alpha$ est dérivable dans le système formel formé par les trois règles de typage.

**Exemple :**

Le terme $\lambda x : \alpha.x$ est bien typé, il a le type « $\alpha \rightarrow \alpha$ » dans l'environnement vide. En effet on peut construire l'arbre de dérivation suivant :

$$\frac{(x : \alpha) \mid\text{-}\, x : \alpha}{\mid\text{-}\, (\lambda x : \alpha.x) : \alpha \rightarrow \alpha}$$

De même le terme : « $\lambda x : i.\lambda f : i \rightarrow i \rightarrow i.fxx$ » est un terme bien typé de type :

« $i \rightarrow (i \rightarrow i \rightarrow i) \rightarrow i$ »

[1] Pour plus de détails, consulter la section : « Séquents et la déduction naturelle » au niveau du chapitre 2.

$$\dfrac{\dfrac{\dfrac{\dfrac{(x:i)(f:i\to i\to i) \mid\!- f:i\to i\to i, \quad (x:i)(f:i\to i\to i)\mid\!- x:i}{(x:i)(f:i\to i\to i)\mid\!- (fx):i\to i}, (x:i)(f:i\to i\to i)\mid\!- x:i}{(x:i)(f:i\to i\to i)\mid\!- (fxx):i}}{(x:i)\mid\!- (\lambda f:i\to i\to i.fxx):(i\to i\to i)\to i}}{(\lambda x:i.\lambda f:i\to i\to i.fxx):i\to(i\to i\to i)\to i}$$

En revanche, il est impossible de construire une dérivation permettant de montrer que le terme $\lambda x:\tau.xx$ est bien typé (quelque soit le type $\tau$ ).

### 1.2.3 Normalisation forte

La normalisation forte d'un terme est la propriété qui affirme que le processus de réduction se termine. Cette propriété est satisfaite dans les calculs de substitutions explicites pour les termes typables.

Pour réduire les termes nous conservons les mêmes règles de Bêta-réductions. Le calcul reste confluent. Si on restreint aux termes bien typés [DL 03], la terminaison du calcul est assurée.

Si un terme est bien typé alors il admet une forme normale et une seule.

## 1.3 Logique Constructive

### 1.3.1 La logique minimale

Les formules de la logique minimale sont définies inductivement de la manière suivante :

i. Si x est une variable propositionnelle, alors x est une formule.

ii. Si $F_1$ *et* $F_2$ sont des formule alors $F_1 \to F_2$ est une formule

Le principe de raisonnement utilisé pour les formules de la logique minimale est basé sur trois règles de déduction du tableau suivant :

| Déduction Naturelle | Lambda calcul simplement typé |
|---|---|

| | |
|---|---|
| $(HYP)\quad \dfrac{A \in \Delta}{\Delta \mid\text{-}\ A}$ | $(Var)\quad \dfrac{x:\alpha \in \Delta}{\Delta \mid\text{-}\ x:\alpha}$ |
| $(i \to)\quad \dfrac{\Delta,\ A \mid\text{-}\ B}{\Delta \mid\text{-}\ A \to B}$ | $(ABS)\quad \dfrac{\Delta \cup \{x:\alpha\} \mid\text{-}\ M:\beta}{\Delta \mid\text{-}\ (\lambda x:\alpha.M):\alpha \to \beta}$ |
| $(e \to)\quad \dfrac{\Delta \mid\text{-}\ A \quad \Delta \mid\text{-}\ A \to B}{\Delta \mid\text{-}\ B}$ | $(APP)\quad \dfrac{\Delta \mid\text{-}\ M:\alpha \to \beta \quad \Delta \mid\text{-}\ N:\alpha}{\Delta \mid\text{-}\ (MN):\beta}$ |

**Figure 1-2 l'isomorphisme de Curry Howard**

Une déduction en logique minimale peut être définie inductivement de la manière suivante :

Soit $\Delta$ un ensemble d'hypothèses

- Si $A \in \Delta$ alors $\Delta \vdash A$ est une déduction de conclusion $\Delta \vdash A$.
- Si $d$ est une déduction de conclusion $\Delta, A \vdash B$ alors $\dfrac{d}{\Delta \vdash A \to B}$ est une déduction de conclusion $\Delta \vdash A \to B$.
- Si $d_1$ est une déduction de conclusion $\Delta \vdash A \to B$ et si $d_2$ est une déduction de conclusion $\Delta \vdash A$ alors $\dfrac{d_1,\ d_2}{\Delta \vdash B}$ est une déduction de conclusion $\Delta \vdash B$.

Si il existe une déduction de conclusion $\Delta \vdash A$, on dit que $\Delta \vdash A$ est dérivable.

Si $\Delta$ est vide, on écrit $\vdash A$.

Si $\vdash A$ est dérivable alors la proposition A est dite théorème.

## 1.3.2 Le Calcul des Prédicats

Le calcul des prédicats est défini sur deux types différents d'objets : le premier type dénote les termes, qui sont des constructions sur un ensemble de variables et un ensemble de

symboles de fonctions, et le deuxième dénote les formules qui sont des constructions sur les termes.

Le premier type d'objet peut être vu comme la représentation d'une entité et le deuxième comme l'une des caractéristiques de l'entité représentée.

Par exemple le premier pour les preuves et l'autre pour les propositions [Dow 91], [Lal 90] et [Kri 90].

## Les Termes

- Les variables sont des termes ;
- Si $t_1,\ldots,t_n$ sont des termes et si $f$ est un symbole de fonction d'arité n, alors : $f(t_1,\ldots,t_n)$ est un terme.

## Formules du calcul des prédicats

- Si P est un prédicat et $t_1,\ldots,t_n$ sont des termes, alors $P(t_1,\ldots,t_n)$ est une formule ;
- $\perp$ est une formule ;
- Si A et B sont deux formules, alors $\neg A$, $A \wedge B$, $A \vee B$ et $A \rightarrow B$ sont des formules ;
- Si A une formule, alors $\forall x.A$ et $\exists x.A$ sont des formules.

Dans les formules quantifiées définies précédemment, si on considère le calcul des prédicats du premier ordre les occurrences liées sont des variables de termes et ils peuvent être des variables des fonctions ou des prédicats dans le cas où on considère le calcul des prédicats d'ordre supérieur [Cat 95]. La preuve d'une formule constitue un arbre de dérivation dont les nœuds sont des jugements (correspondant à une règle) et les feuilles sont les axiomes [Cat 95].

| | |
|---|---|
| $(Axiom) : \dfrac{A \in \Gamma}{\Gamma \mid\!- A}$ | |
| $(\neg(i)) : \dfrac{\Gamma[A] \mid\!- \perp}{\Gamma \mid\!- \neg A}$ | $(\neg(e)) : \dfrac{\Gamma \mid\!- \neg A \quad \Gamma \mid\!- A}{\Gamma \mid\!- \perp}$ $(\perp(e)) : \dfrac{\Gamma \mid\!- \perp}{\Gamma \mid\!- A}$ |
| $(\rightarrow(i)) : \dfrac{\Gamma[A] \mid\!- B}{\Gamma \mid\!- A \rightarrow B}$ | $(\rightarrow(e)) : \dfrac{\Gamma \mid\!- A \rightarrow B \quad \Delta \mid\!- A}{\Gamma, \Delta \mid\!- B}$ |
| $(\wedge(i)) : \dfrac{\Gamma \mid\!- A \quad \Gamma \mid\!- B}{\Gamma \mid\!- A \wedge B}$ | $(\wedge(e_1)) : \dfrac{\Gamma \mid\!- A \wedge B}{\Gamma \mid\!- A}$ $(\wedge(e_2)) : \dfrac{\Gamma \mid\!- A \wedge B}{\Gamma \mid\!- B}$ |

| | |
|---|---|
| $(\vee(i_1))$ : $\dfrac{\Gamma \mid - A}{\Gamma \mid - A \vee B}$ <br> $(\vee(i_2))$ : $\dfrac{\Gamma \mid - B}{\Gamma \mid - A \vee B}$ | $(\vee(e))\dfrac{\Gamma \mid -A \vee B, \Gamma \mid -A \rightarrow C , \Gamma \mid -V \rightarrow C}{\Gamma \mid -C}$ |
| $(\forall(i))$ : $\dfrac{\Gamma \mid - A}{\Gamma \mid - \forall x.A}$ | $(\forall(e))$ : $\dfrac{\Gamma \mid - \forall x.A}{\Gamma \mid - A[x \leftarrow t]}$ |
| $(\exists(i))$ : $\dfrac{\Gamma \mid - A[x \leftarrow t]}{\Gamma \mid - \exists x.A}$ | $(\exists(e))$ : $\dfrac{\Gamma \mid - \exists x.A \quad \Gamma \mid - \forall x.(A \rightarrow B)}{\Gamma \mid - B}$ |

**Figure 1-3 règles de la logique des predicats**

## 1.3.3 Sémantique de Heyting

Si on considère la logique intuitionniste (constructive) où la règle du tiers exclu n'est plus valide et la sémantique donnée à une formule n'est plus une valeur de vérité [Bar 84], on associe à chaque formule l'espace éventuellement peut être vide de ses preuves (ou démonstrations). Cette interprétation est appelée sémantique de Heyting[1] .

La sémantique de Heyting associée aux formules du calcul intuitionniste s'exprime de la façon suivante :

- Une preuve de l'implication « $A \rightarrow B$ » est une fonction qui à toute preuve de « A » associe une preuve de « B ».
- Une preuve de la conjonction « $A \wedge B$ » est un couple composé d'une preuve de « A » et d'une preuve de « B ».
- L'ensemble des preuves de la proposition absurde *False* est vide.
- Une preuve de la disjonction « $A \vee B$ » est un couple de la forme $(i, p)$ où si i vaut 1 alors *p* est la preuve de « A » et si i vaut 2 alors *p* est une preuve de « B ».
- Une preuve de « $\forall x$.A » est une fonction qui à tout objet x associe une preuve de A.
- Une preuve de « $\exists x$.A » est un couple $(t, p)$ où t est un objet (appelé *témoin*) et « *p* » est une preuve de la formule « A[x/t] ». Montrer l'existence en logique constructive revient à exhiber un témoin.

Par exemple, une preuve de « $A \rightarrow A$ » est une fonction qui à toute preuve de « A » associe une preuve de « A ». Donc la fonction identité est une preuve de cette proposition.

---

[1] Dite aussi de Brower-Heyting-Kolmoro.

On remarque aussi que si on dispose d'une preuve « *f* » de « $A \rightarrow B$ » et d'une preuve « *a* » de « A », alors il suffit d'appliquer la fonction « *f* » à l'argument « *a* » pour obtenir une preuve de « B ».

Si « $f_1$ » est une preuve de $A \rightarrow B \rightarrow C$, si « $f_2$ » est une preuve de $A \rightarrow B$ et si « *a* » est une preuve de « *A* » alors « $f_1\ a$ » est une preuve de « $B \rightarrow C$ », « $f_2\ a$ » est une preuve de « B » et donc « $(f_1\ a)(f_2\ a)$ » est une preuve de « C ».

La fonction aux trois arguments $f_1$, $f_2$ *et* $a$ qui calcule $(f_1\ a)(f_2\ a)$ est une preuve de la formule :

$$(A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow B .$$

En notation lambda-calcul, cette fonction s'écrit :

$$\lambda f_1.\lambda f_2 \lambda a.(f_1 a)(f_2 a)$$

Si on considère uniquement les formules de la logique minimale, les preuves des formules sont des termes du lambda-calcul simplement typé. Cette identification s'appuie sur la correspondance de Curry Howard[1]. Pour associer un type à une preuve on procède de la manière suivante :

- Si « A » est une formule atomique, alors le « $Type(A) = A$ » et on identifie les formules atomiques et les types de base.
- Si « A » et « B » sont deux formules quelconques, alors le $Type(A \rightarrow B)$ est un type fonctionnel (la preuve d'une implication est une fonction) $Type(A \rightarrow B) = Type(A) \supset Type(B)$

## 1.3.4 L'isomorphisme de Curry Howard

L'isomorphisme de Curry Howard est une interprétation de la relation des termes avec leurs types. Cette interprétation, connue souvent sous le paradigme *type-as-formula* [How 80], consiste à considérer les types comme des propositions logiques et la preuve d'une proposition logique comme le lambda terme correspondant.

Par exemple, l'interprétation de l'expression :

$$M : \alpha$$

[1] Cette correspondance est bien détaillée dans le chapitre 2 (section 2.1.4)

consiste à considérer $\alpha$ comme étant une proposition, et $M$ comme étant la preuve de la proposition $\alpha$.

## 1.4 Extensions du lambda calcul typé

L'utilisation du paradigme type-as-formula, défini précédemment, pour interpréter la logique est limitée à la logique propositionnelle. Dans le but d'interpréter les logiques d'ordre supérieur des extensions pour le lambda calcul typé sont définies. Parmi ces extensions nous présenterons dans ce qui suit le système polymorphique du second ordre λ2 et le calcul des constructions.

### 1.4.1 Système polymorphique du second ordre λ2

Il s'agit d'une extension du système de Curry introduite par Girard et Reynolds [RP 90]. L'alphabet de ce système est définie par deux ensembles : un ensemble de variables propositionnelles et un ensemble de termes.

La syntaxe formelle des expressions de cette extension est définie par la grammaire suivante :

$$
\begin{aligned}
&< Formule > \to \langle Term\rangle : \langle \Pr op\rangle \\
&\langle \Pr op\rangle \to \langle Vprop\rangle \mid (\langle \Pr op\rangle -> \langle \Pr op\rangle) \mid (\forall \langle Vprop\rangle)\langle \Pr op\rangle \\
&\langle Term\rangle \to \langle Vterm\rangle \mid (\langle Term\rangle\langle Term\rangle) \mid \langle Term\rangle\{\langle \Pr op\rangle\} \\
&\qquad \mid (\lambda\langle Vterm\rangle : \langle Vprop\rangle.\langle Term\rangle) \mid \Lambda\langle Vprop\rangle.\langle Term\rangle
\end{aligned}
$$

**Figure 1-4 Syntaxe formelle du système λ2**

Les règles d'assignation de type dans le système λ2 sont données par le tableau suivant :

| | **Introduction** | **Elimination** |
|---|---|---|
| $\to$ | $\dfrac{\begin{matrix}[x:\alpha] \\ \vdots \\ M:\beta\end{matrix}}{\lambda x:\alpha.M : \alpha\to\beta}$ | $\dfrac{N:\alpha \qquad M:\alpha\to\beta}{MN:\beta}$ |
| $\forall$ | $\dfrac{M:A}{\Lambda p.M : (\forall p)A}$ | $\dfrac{M:(\forall p)A(p)}{M\{\beta\}:[\beta/p]A}$ |

**Figure 1-5 Règles d'inférence du système λ2**

## 1.4.2 Calcul des constructions

D'une manière similaire de la définition du système précèdent, la syntaxe des expression du calcul des constructions [Sel 97] est définie par la grammaire suivante :

$$
\begin{array}{l}
< Formule > \rightarrow \langle Term \rangle : \langle Term \rangle | \ \langle Term \rangle : Type \\
\langle Term \rangle \rightarrow \langle Var \rangle | \mathrm{Prop} \ | \ \langle Term \rangle \langle Term \rangle \ | \ \lambda \langle Var \rangle : \langle Term \rangle . \langle Term \rangle \\
\quad\quad | \ (\forall \langle Var \rangle : \langle Term \rangle \langle Term \rangle)
\end{array}
$$

**Figure 1-6 Syntaxe formelle du calcul des constructions**

Les règles de construction des types pour cette extension sont données par le tableau suivant:

$$
\frac{\begin{array}{c}[x:\alpha] \\ \vdots \\ \beta : \mathrm{Prop}\end{array} \quad \alpha : \mathrm{Prop}}{(\forall x:\alpha)\beta : \mathrm{Prop}} (P\forall_1)
\qquad
\frac{\begin{array}{c}[x:\alpha] \\ \vdots \\ \beta : Type\end{array} \quad \alpha : Type}{(\forall x:\alpha)\beta : \mathrm{Prop}} (P\forall_2)
$$

$$
\frac{\begin{array}{c}[x:\alpha] \\ \vdots \\ \beta : Type\end{array} \quad \alpha : \mathrm{Prop}}{(\forall x:\alpha)\beta : Type} (TA_1)
\qquad
\frac{\begin{array}{c}[x:\alpha] \\ \vdots \\ \beta : Type\end{array} \quad \alpha : Type}{(\forall x:\alpha)\beta : Type} (TA_2)
$$

Condition : x n'est pas libre dans α et dans aucune hypothèse non déchargée

Les règles d'assignation des types pour cette extension sont les suivantes :

| | Introduction | Elimination |
|---|---|---|
| $\forall$ | $\frac{\begin{array}{c}[x:\alpha] \\ \vdots \\ M : \beta\end{array} \quad \alpha : \mathrm{Prop}}{\lambda x:\alpha.M : (\forall x:\alpha)\beta} (\forall P)$ <br> $\frac{\begin{array}{c}[x:\alpha] \\ \vdots \\ M : \beta\end{array} \quad \alpha : Type}{\lambda x:\alpha.M : (\forall x:\alpha)\beta} (\forall T)$ | $\frac{N : \alpha \quad M : (\forall x:\alpha)\beta}{MN : [N/x]\beta}$ |

**Condition :** x n'est pas libre dans A et dans aucune hypothèse non déchargée

**Figure 1-7 règles d'assignation des types dans le calcul des constructions**

## 1.5 Logique et Concepts Mathématiques dans le Langage des Types

Nous présentons, dans ce qui suit, la représentation de la logique et concepts mathématique [Lak 76] de base dans le système des types, formalisation des briques de base permettant le codage de l'arithmétique. Ce codage est nécessaire pour montrer la puissance de la théorie du lambda calcul [Sel 97].

### 1.5.1 Connecteurs et Quantificateurs

#### a) Implication

Soit $x$ une variable et $B$ une expression du calcul des construction ; si $x$ est non libre dans $B$ alors $(\forall x : A)B$ est généralement considérée comme $A \to B$, et si A et B sont des propositions (de type Prop), on écrit $A \supset B$. L'interprétation de l'application et l'abstraction, les cas particuliers, sont donnée par les règles suivantes :

$$\frac{\Gamma \;|\!- M : A \to B \quad \Gamma \;|\!- N : A}{\Gamma \;|\!- MN : B}$$

et

$$\frac{\Gamma, x : A \;|\!- M : B \quad \Gamma \;|\!- A \to B}{\Gamma \;|\!- \lambda x : A.M : A \to B}$$

#### b) La conjonction

La conjonction de deux proposition A et B est définie par :

$$A \wedge B \equiv (\forall p : \mathrm{Pr}\,op)((A \to B \to p) \to p)$$

Les termes de type $A \wedge B$ forment l'ensemble des couples ; dont le premier élément est de type A et le seconde est de type B. L'opérateur de couplement et ses projections nous permet de confirmer les propriétés suivantes de la conjonction [Sel 97]

$$A \to B \to A \wedge B$$
$$A \wedge B \to A$$
$$A \wedge B \to B$$

#### c) La disjonction

La disjonction de deux proposition A et B est définie [Sel 97] par :

$$A \vee B \equiv (\forall p : Prop)((A \to p) \to ((B \to p) \to p))$$

L'ensemble des termes de type $A \vee B$ est l'union des termes de type A et les termes de type B. Ces unions avec leurs injections ainsi que l'opérateur case montrent les propriétés de la disjonction englobant les résultats suivants :

$$A \to A \vee B$$
$$B \to A \vee B$$
$$A \vee B \to (\forall p : Prop)((A \to p) \to ((B \to p) \to p))$$

## ***d)*** **Le type « *void* »**

Le type *void,* noté par « $\perp$ » et qui représente la proposition absurde [Sel 97]. Ce type peut être exprimé par la proposition suivante :

$$void \quad \equiv \quad \perp \quad \equiv (\forall x : Prop)x$$

## e) La négation d'une proposition

La négation de la proposition A est définie en fonction de la proposition absurde de la manière suivante :

$$\neg A \equiv A \supset \perp$$

## f) La quantification existentielle

La quantification existentielle : $(\exists x : A)B$ où A et B sont de type *Prop* et $x$ n'apparaît pas libre dans A, est définie de la manière suivante [Sel 97]:

$$(\exists x : A)B \equiv (\forall p : Prop)((\forall x : A)(B \supset p) \supset p)$$

Les termes de type $(\exists x : A)B$ sont des couples dont le premier élément est de type A et le seconde est de type B. Ces couples sont définis d'une manière différente que celle utilisée dans la conjonction. Les résultats suivants représentent quelques propriétés de la quantification existentielle :

$$[M / x]B \supset (\exists x : A)B \qquad (avec\ M : A)$$
$$(\exists x : A)B \supset (\forall p : Prop)((\forall x : A)(B \supset p) \supset p)$$

## g) Egalité de Leibniz

Soient M et N deux expressions de type A, l'égalité de Leibniz est définie de la manière suivante :

$$M =_A N \equiv (\forall z : A \rightarrow \mathrm{Pr}\,op)(zM \supset zN)$$

Les résultats suivants représentent quelques propriétés de l'égalité (M et N sont de type A) :

$$M =_A M$$
$$M =_A N \supset (\forall z : (A \rightarrow \mathrm{Pr}\,op) \supset (zM \supset zN))$$

**h) Représentation des valeurs de vérité**

Avec les définitions précédentes, on a défini une logique intuitionniste d'ordre supérieur. Le type booléen et les valeurs de vérités sont représentés de la manière suivante :

$$\begin{aligned} Bool &\equiv (\forall u : \mathrm{Pr}\,op)(u \rightarrow u \rightarrow u), \\ T &\equiv \lambda u : \mathrm{Pr}\,op.\lambda x : u.\lambda y : u.x \\ F &\equiv \lambda u : \mathrm{Pr}\,op.\lambda x : u.\lambda y : u.y \end{aligned}$$

On peut prouver facilement que $Bool : \mathrm{Pr}\,op$ , $T : Bool$ et $F : Bool$

## 1.5.2 Représentation de l'arithmétique

Pour représenter l'arithmétique, les définitions suivantes sont nécessaires :

$$\begin{aligned} \mathrm{N} &\equiv (\forall A : \mathrm{Pr}\,op)((A \rightarrow A) \rightarrow (A \rightarrow A)) \\ 0 &\equiv \lambda A : \mathrm{Pr}\,op.\ \lambda x : A \rightarrow A.\lambda y : A.y \\ \sigma &\equiv \lambda u : \mathrm{N}.\ \lambda A : \mathrm{Pr}\,op.\ \lambda x : A \rightarrow A.\lambda y : A.x(uAxy) \end{aligned}$$

Où **N** est le type des entiers naturels, **0** est le nombre zéro et **σ** est la fonction successeur. Avec les définitions précédentes, l'entier naturel n est représenté de la manière suivante :

$$n = \lambda A : \mathrm{Pr}\,op.\lambda x : A \rightarrow A.\lambda y : A.x(x(\ldots(xy)\ldots)) \quad n\ fois$$

Il est possible de définir la constante **π** tel que :

$$\pi 0 = 0$$
$$\pi(\sigma\, n) = n$$

En utilisant la constante **π**, il est possible de définir le précurseur **R** défini de la manière suivante :

$$\left.\begin{aligned} &A : \mathrm{Pr}\,op \\ &M : A \\ &N : \mathrm{N} \rightarrow A \rightarrow A \end{aligned}\right\} \Rightarrow \begin{cases} RMN0 = M \\ RMN(\sigma\, n) = Nn(RMNn) \end{cases}$$

D'une manière similaire, on peut représenter les listes par des termes de type *A* [Sel 97]. pour cela l'utilisation d'un type *List* est nécessaire. L'application de ce type au type *A* engendre un type *ListA* de listes d'objets de type *A*. Pour assurer une manipulation parfait des listes, il faut définir :

- le type *nilA,* qui représente la liste vide.
- la fonction *consA* de type $A \to ListA \to ListA$ permettant de construire une liste d'objets de type A par l'insertion d'un objet de type A dans une liste d'objets de type A.
- récursivement des fonction sur les listes et les objectes de type A.

Par exemple la fonction *append* permettant la concaténation de deux listes est définie de la manière suivante :

Soient L1 et L2 deux listes de type *ListA* et M est un objet de type A.

$$AppendA\,(nilA)L_2 = L_2$$
$$AppendA\,(consA\,M\,L_1)L_2 = consA\,M\,(AppendA\,L_1L_2)$$

Un autre exemple de fonction définie sur les listes est la fonction « *reverse* » permettant d'inverser l'ordre des éléments d'une liste. Cette fonction est définie de la manière suivante :

$$reverseA\,L = flipA\,L(nilA)$$

Où *flip* est définie de la manière suivante :

$$flipA(nilA)L_2 = L_2$$
$$flipA(consA\,M\,L_1)L_2 = flipA\,L_1(consA\,M\,L_2)$$

## 1.5.3 Représentation des ensembles

Dans le chapitre 5 de [Hue 86], Huet a proposé de représenter la théorie des ensembles élémentaires par des formules du calcul des prédicats. L'idée est de considérer un type $U : \mathrm{Pr}\,op$ comme univers et $Set_U$ est le type $U \to \mathrm{Pr}\,op$, et si $A : Set_U$ alors la relation $x \in A$ est représentée par l'application $Ax$. De plus si $P : \mathrm{Pr}\,op$ alors l'ensemble $\{x : U \text{ tel que } P\}$ est représenté par l'expression « $\lambda x : U.P$ » et d'une manière similaire l'inclusion de l'ensemble A dans l'ensemble B est représentée par l'expression suivante :

$$A \subseteq B \equiv (\forall x : U)(x \in A \supset x \in B)$$
$$A = B \equiv (A \subseteq B) \wedge (B \subseteq A)$$
$$\Phi \equiv \{x : U \text{ tel que } \bot\}$$
$$A \cap B \equiv \{x : U \text{ tel que } x \in A \wedge x \in B\}$$
$$A \cup B \equiv \{x : U \text{ tel que } x \in A \vee x \in B\}$$
$$\sim \mathrm{A} \equiv \{x : U \text{ tel que } \neg x \in A\}$$
$$PA \equiv \lambda B : Set_U.B \subseteq A$$
$$Class_U \equiv Set_U \to \mathrm{Pr}\,op$$

On peut représenter la collection des fonctions définies de A vers *B* [Sel 97], où *A* et *B* sont de type $Set_U$, de la manière suivante :

$$\lambda f : U \to U.(\forall x : U)(x \in A \supset fx \in B)$$

2 Systèmes de preuve

# Chapitre 2

# Systèmes de Preuve

Le principal défaut des systèmes de Hilbert[1] est le fait que nous ne pouvons pas définir une procédure précise pour mécaniser le processus de démonstration. Dans un système de Hilbert, les difficultés, rencontrés pour prouver une formule, commencent dès le premier pas, où il y a plusieurs choix pour choisir la formule de départ. Le choix de cette formule est obtenu en substituant une ou plusieurs sous formules d'un axiome du système utilisé. Le même problème se pose pour passer à l'étape suivante : ajouter une hypothèse, appliquer une règle d'inférence aux formules des étapes précédentes…etc.

Au niveau de cette partie, nous allons présenter les concepts et les mécanismes de base utilisés dans la mécanisation d'un processus de déduction. Nous nous intéressons particulièrement à la déduction naturelle, le calcul des séquents et le principe de quelques systèmes de preuve, notamment le démonstrateur pédagogique FOLDROL.

## 2.1 Déduction Naturelle

Les systèmes de déduction naturelle interviennent pour remédier aux défauts des systèmes de Hilbert et offrent une stratégie, à travers les règles de déduction, permettant la manipulation de l'ensemble d'hypothèses. Le mécanisme de base de la déduction naturelle peut être présenté sous des formats différents mais tous sont équivalents.

[1] Sont les systèmes permettant la recherche des théorèmes. Une preuve en format de Hilbert est une suite de propositions, qui sont des axiomes ou des conséquences de propositions précédentes, déduites par des règles de preuve bien définies.

### 2.1.1 Contextes et jugements :

Un contexte est une liste de formules.
Un jugement est une expression de la forme :

$$\Gamma \quad |- \quad A$$

- $\Gamma$ est le contexte d'hypothèses.
- |- est le symbole de déduction.
- A est une formule qui correspond à la conclusion du jugement.

### 2.1.2 Règles de la déduction naturelle

Le processus utilisé pour démonter une formule à partir d'autres formules est appelé système de dérivation. Un système de dérivation est un système contenant des règles d'inférence sur des termes d'un langage, représentées par des déductions d'un ou plusieurs jugements vers un autre. Les règles, pour la logique classique propositionnelle, sont classées selon la symétrie introduction/élimination (figure 2.1).

L'application de ces règles en série nous permet de dériver des formules et donc faire des preuves [Cat 95]

Par exemple, si ($\pi$) est une dérivation qui prouve la proposition A, et ($\pi'$) est une dérivation qui prouve la proposition B, alors la dérivation suivante prouve la conjonction $A \wedge B$ :

$$\frac{\begin{matrix}\vdots & & \vdots \\ A & & B\end{matrix}}{A \wedge B}(\wedge I)$$

La règle d'introduction du connecteur de conjonction ($\wedge I$) permet de construire, à partir des deux preuves des formules A et B, la preuve de la conjonction de ces derniers. Dans l'autre sens les règles ($\wedge E_1$) et ($\wedge E_2$) permet de prouver une des sous formules d'une conjonction déjà prouvée.

| **Introduction** | **Elimination** |
| --- | --- |
| $\dfrac{\overset{\vdots}{A} \quad \overset{\vdots}{B}}{A \wedge B}(\wedge I)$ | $\dfrac{\overset{\vdots}{A \wedge B}}{A}(\wedge E_1)$ $\quad$ $\dfrac{\overset{\vdots}{A \wedge B}}{B}(\wedge E_2)$ |
| $\dfrac{\overset{[A]_i}{\overset{\vdots}{B}}}{A \rightarrow B}(\rightarrow I)$ | $\dfrac{\overset{\vdots}{A} \quad \overset{\vdots}{A \rightarrow B}}{B}(\rightarrow E)$ |
| $\dfrac{\overset{\vdots}{A}}{A \vee B}(\vee I_1)$ $\quad$ $\dfrac{\overset{\vdots}{B}}{A \vee B}(\vee I_2)$ | $\dfrac{\overset{\vdots}{A \vee B} \quad \overset{[A]_i}{\overset{\vdots}{\Phi}} \quad \overset{[B]_i}{\overset{\vdots}{\Phi}}}{\Phi}(\vee E)$ |
| $\dfrac{\overset{[A]_i}{\overset{\vdots}{B \wedge \neg B}}}{\neg A}(\neg I)$ | $\dfrac{\overset{\vdots}{A} \quad \overset{\vdots}{\neg A}}{B}(\neg E)$ |
| $\dfrac{\overset{\vdots}{A}}{\neg\neg A}(\neg\neg I)$ | $\dfrac{\overset{\vdots}{A}}{\neg\neg A}(\neg\neg E)$ |

**Figure 2-1 Règles de la déduction naturelle**

La règle d'élimination de la négation a comme conclusion une formule arbitraire : à partir d'une contradiction, où on a les preuves d'une formule et de sa négation, nous pouvons déduire n'importe quelle proposition. Les autres règles s'expliquent d'elles mêmes, sauf celles qui appellent la notation $[\ ]_i$ : ($\Rightarrow I$), ($\neg I$) et ($\vee E$). Les crochets représentent une hypothèse qui a été déchargée. Par exemple la règle d'introduction de l'implication ($\Rightarrow I$) qui peut être interprétée par :

Si ($\pi$) est une preuve de B utilisant A comme hypothèse, soit si :

$$\begin{matrix} A \\ \vdots \\ (\Pi) \\ \vdots \\ B \end{matrix}$$

Alors nous pouvons déduire de cette preuve une preuve de $A \rightarrow B$ sans utiliser A comme hypothèse auxiliaire. La notation utilisée pour décharger A est $[A]_i$, où $i$ est une étiquette

unique (disons un entier) dont nous nous servons pour associer l'inférence de $A \to B$ avec les occurrences de A que nous déchargeons. Déchargeons A ci-dessus :

$$\begin{array}{c} [\mathrm{A}]_1 \\ \vdots \\ (\Pi) \\ \vdots \\ \dfrac{\mathrm{B}}{\mathrm{A} \to \mathrm{B}} 1(\to I) \end{array}$$

## 2.1.3 Formalisation du processus de dérivation

Une présentation plus formelle de la déduction naturelle, qui montre à la fois les formules et l'ensemble des hypothèses auxiliaires, est la déduction naturelle en format des séquents. Avec cette représentation, au lieu de dériver juste des formules, nous dérivons des séquents de la forme suivante :

$$\Gamma \ \vdash \mathrm{A}$$

Où $\Gamma$ est un ensemble de formules, appelé le *contexte* du séquent[1] (l'ensemble des hypothèses auxiliaires courantes) et le but est de dériver la formule A.

**Définition 2.1**

Une dérivation en déduction naturelle est un arbre inversé dont les noeuds sont des séquents connectés par des règles de déduction de la figure 2-2. Les feuilles sont des instances de l'axiome (Ax).

- ✓ Une *preuve* P d'un jugement : $\Gamma \ \vdash \ A$ est une dérivation en déduction naturelle dont la racine est décorée par $\Gamma \ \vdash \ A$ .
- ✓ Nous disons aussi que P est une preuve de A *à partir de* $\Gamma$.
- ✓ S'il existe une preuve de $\Gamma \ \vdash \ A$ ., nous disons que $\Gamma \ \vdash \ A$ .est *prouvable*, ou bien que la proposition A est prouvable à partir de l'environnement $\Gamma$, et nous écrivons :

$$\Gamma \ \vdash^{ND} \mathrm{A}$$

[1] La définition est disponible dans la section 2.2

✓ Un *théorème* est une formule prouvable à partir d'un ensemble vide d'hypothèses.

| $\dfrac{}{\Gamma, A \ \vert\text{-}\ A}(Ax)$ | |
|---|---|
| **Introduction** | **Elimination** |
| $\dfrac{\Gamma \ \vert\text{-}\ A \quad \Gamma \ \vert\text{-}\ B}{\Gamma \ \vert\text{-}\ A \wedge B}(\wedge I)$ | $\dfrac{\Gamma \ \vert\text{-}\ A_1 \wedge A_2}{\Gamma \ \vert\text{-}\ A_i \quad i=1.2}(\wedge E)$ |
| $\dfrac{\Gamma, A \ \vert\text{-}\ B}{\Gamma \ \vert\text{-}\ A \rightarrow B}(\rightarrow I)$ | $\dfrac{\Gamma, A \ \vert\text{-}\ B \quad \Gamma \ \vert\text{-}\ A}{\Gamma \ \vert\text{-}\ B}(\rightarrow E)$ |
| $\dfrac{\Gamma \ \vert\text{-}\ A}{\Gamma \ \vert\text{-}\ A \vee B}(\vee I) \quad \dfrac{\Gamma \ \vert\text{-}\ B}{\Gamma \ \vert\text{-}\ A \vee B}(\vee I)$ | $\dfrac{\Gamma \ \vert\text{-}\ A \vee B \quad \Gamma, A \ \vert\text{-}\ \Phi \quad \Gamma, B \ \vert\text{-}\ \Phi}{\Gamma \ \vert\text{-}\ \Phi}(\vee E)$ |
| $\dfrac{\Gamma, A \ \vert\text{-}\ B}{\Gamma \ \vert\text{-}\ \neg A}(\neg I)$ | $\dfrac{\Gamma \ \vert\text{-}\ A \quad \Gamma \ \vert\text{-}\ \neg A}{\Gamma \vert\text{-} B}(\neg E)$ |
| $\dfrac{\Gamma \ \vert\text{-}\ A}{\Gamma \ \vert\text{-}\ \neg\neg A}(\neg\neg I)$ | $\dfrac{\Gamma \ \vert\text{-}\ \neg\neg A}{\Gamma \ \vert\text{-}\ A}(\neg\neg E)$ |

**Figure 2-2 Déduction naturelle en format de séquents**

## 2.2 Le calcul des séquents

### 2.2.1 Jugements du calcul des séquents

Dans le calcul des séquents [Tak 87], les jugements sont de la forme suivante :

$$\Gamma \ \vert- \Delta$$

où $\Gamma, \Delta$ sont des ensembles finis de propositions.

Le cas intuitionniste est le cas particulier où $\Delta$ est constitué d'une et une seule proposition. Le séquent $\Gamma \ \vert- A$ exprime que dans un contexte donné $\Gamma$, d'un système de preuve, la formule A est dérivable à partir de l'ensemble des hypothèses (les formules de l'ensemble $\Gamma$), ou aussi la formule A dépend de l'ensemble des hypothèses $\Gamma$.

## 2.2.2 Les règles du calcul des séquents

Les règles du calcul des séquents respectent une symétrie gauche/droite. Ces règles sont classés en quatre types : structurelles, logiques (il n'y a que des règles d'introduction), axiome et les règles de coupures.

### a) L'axiome

La forme générale de l'axiome est la suivante :

$$\Gamma, \varphi \ \vdash \ \Delta, \varphi$$

### b) Les règles structurelles :

i. l'affaiblissement :

$$\frac{\Gamma \ \vdash \ \Delta}{\Gamma, \varphi \ \vdash \ \Delta} \qquad \frac{\Gamma \ \vdash \ \Delta}{\Gamma \ \vdash \ \Delta, \varphi}$$

ii. La contraction

$$\frac{\Gamma, \varphi, \varphi \ \vdash \ \Delta}{\Gamma, \varphi \ \vdash \Delta} \qquad \frac{\Gamma \ \vdash \ \Delta, \varphi, \varphi}{\Gamma \ \vdash \Delta, \varphi}$$

### c) Les règles logiques :

i. L'introduction de la conjonction :

(à gauche)
$$\frac{\Gamma, \varphi \ \vdash \ \Delta}{\Gamma, \varphi \wedge \psi \ \vdash \ \Delta} \qquad \frac{\Gamma, \psi \ \vdash \ \Delta}{\Gamma, \varphi \wedge \psi \ \vdash \ \Delta}$$

(à droite)
$$\frac{\Gamma \ \vdash \ \Delta, \varphi \qquad \Gamma \ \vdash \ \Delta, \psi}{\Gamma \vdash \ \Delta, \varphi \wedge \psi}$$

ii. L'introduction de la disjonction :

(à gauche)
$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}$$

(à droite)
$$\frac{\Gamma \vdash \Delta, \varphi}{\Gamma \vdash \Delta, \varphi \vee \psi} \qquad \frac{\Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \vee \psi}$$

iii. L'introduction d'une implication :

(à gauche)
$$\frac{\Gamma, \psi \vdash \Delta \qquad \Gamma \vdash \Delta, \varphi}{\Gamma, \varphi \to \psi \vdash \Delta}$$

(à droite)
$$\frac{\Gamma, \varphi \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \to \psi}$$

iv. L'introduction de la négation:

(à gauche)
$$\frac{\Gamma \vdash \Delta, \varphi}{\Gamma, \neg\varphi \vdash \Delta}$$

(à droite)
$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta, \neg\varphi}$$

v. L'introduction d'une quantification universelle:

(à gauche)
$$\frac{\Gamma, \varphi[x/t] \vdash \Delta}{\Gamma, \forall x \varphi \vdash \Delta}$$

(à droite)
$$\frac{\Gamma \vdash \Delta, \varphi}{\Gamma \vdash \Delta, \forall x \varphi}$$

vi. L'introduction d'une quantification existentielle:

(à gauche) $$\frac{\Gamma,\varphi \quad \vdash \Delta}{\Gamma,\exists x\varphi \quad \vdash \Delta}$$

(à droite) $$\frac{\Gamma \quad \vdash \Delta,\varphi[x/t]}{\Gamma \quad \vdash \Delta,\exists x\varphi}$$

## La preuve de la formule de Pierce :

$$\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi \qquad \dfrac{\varphi \vdash \psi,\varphi}{\vdash \varphi \to \psi,\varphi}\ \text{int}\,r\,(\to droite)}{((\varphi \to \psi) \to \varphi) \vdash \varphi,\varphi}\ \text{int}\,r\,(\to gauche)}{((\varphi \to \psi) \to \varphi) \vdash \varphi}\ \textit{contraction à droite}}{\vdash ((\varphi \to \psi) \to \varphi) \to \varphi}\ \text{int}\,r\,(\to droite)$$

Le calcul des séquents défini, uniquement, avec les règles précédentes est appelé le calcul des séquents sans coupure.

## d) La règle de coupure

Cette règle permet de faire une démonstration plus courte et plus intelligente. Elle n'augmente pas la puissance du calcul des séquents.

$$\frac{\Gamma \ \vdash \Delta,\varphi \qquad \Gamma',\varphi \ \vdash \Delta'}{\Gamma,\Gamma' \ \vdash \Delta,\Delta'}$$

Gentzen a démontré le théorème d'élimination des coupures. Si une démonstration comporte des coupures, on peut les transformer en une démonstration équivalente qui ne comporte pas de coupures. La démonstration du théorème d'élimination des coupures utilise la symétrie du calcul des séquents.

## 2.3 Quelques Systèmes de Preuve

Depuis le système Automath de DeBruijn, un grand nombre de systèmes d'aide à la preuve ont été développés par des équipes différentes [Gal 86]. Certains sont orientés vers l'informatique et d'autre vers les mathématiques.

Nous rappelons par la suite quelques systèmes sélectionnés de manière à explorer le domaine manipulé. Les systèmes choisis ont une caractéristique commune : où les preuves sont vues comme des arbres dont la racine est la proposition à prouver et dont les nœuds sont étiquetés par la règle ou la tactique utilisée.

### 2.3.1 Coq

Le système Coq [Coq 91] est l'un des grands projets développés par l'INRIA en 1986 par T. Coquand et G. Huet, basé sur le calcul des constructions inductives [Wer 94] (une extension du lambda-calcul typé) comprenant les types récursifs, le polymorphisme de type dépendant et le produit ou la somme dépendants où le deuxième argument peut dépendre du premier. Un exemple des types dépendant d'un entier est le type « être une liste de longueur n ». L'utilisation du calcul des constructions dans ce système nécessite une richesse dans le mécanisme de typage où on assimile types et formules (type as formula [Cur 34, Hoa 69] ) et la preuve d'un énoncé est un terme, dont le type est l'énoncé démontré. La logique utilisée est une logique d'ordre supérieur intuitionniste (n'utilise pas le tiers-exclu). Coq permet l'extraction de programmes à partir d'une preuve. Par exemple, la preuve de la proposition : *«pour tout x de type T, il existe un y de type T' tel que la proposition P(x,y) est vraie »* est une fonction prenant en argument un x de type T et retournant un témoin *y* de type T' et une preuve D que *y* vérifie bien cette propriété. Dans le cadre de l'extraction de programme, on ne s'intéressera qu'à la valeur de y. On distinguera deux types de propositions : ayant un contenu algorithmique (dans la preuve permet de construire y) qui sera de type « *set* » et des propositions sans contenu algorithmique (permettant de construire la preuve D) et qui porteront le type « *prop*».

L'extraction néglige tout ce qui est de type « *prop* » et conserve l'autre catégorie des propositions pour définir la fonction qui à « x » associée « y ».

L'avantage majeur du calcul des constructions est son uniformité et il a pour inconvénient de n'être pas naturel pour l'utilisateur.

Pour faciliter l'utilisation du système, un langage étendu est utilisé pour traiter les définitions, déclarations et qui permet la construction des objets en intégrant quelques lignes de code, ces derniers sont traduits, automatiquement par la suite, vers le calcul des constructions.

Sous le système Coq, prouver une proposition revient à construire le terme de preuve à l'aide de l'utilisation des tactiques et des règles supérieur du calcul des constructions. La validation[1] d'une preuve consiste à vérifier le type du terme obtenu.

**Exemples :**

1) Les entiers sont définis par un type récursif ( 0 est un entier et S prend un entier et construit son successeur) par la construction suivante :

```
Inductive nat     : set :=
          0   : nat
          |S  : nat --> nat
```

2) D'une manière similaire, les booléens sont définis par la construction suivante :

```
Inductive bool    : set  :=
           True   : bool
          |false  : bool
```

3) Pour définir les fonctions on utilise un point-fixe et une construction « par cas ». Par exemple pour la fonction « *insere* » chargé à l'insertion d'un entier « x » dans une liste d'entiers « l ». la fonction « *insere* » est définie récursivement « par-cas » par le fragment suivant :

[1] Consulter l'utilisation des systèmes de preuve.

```
Fixepoint insere [x:nat; l:liste]: liste :=
   Cases l of
        Nil =>(cons x nil)
        |(cons n l')=>
             if (inferieur x n)
             then (cons x l)
             else (cons n (insere x l'))
   end.
```

4) D'une manière similaire, on peut définir les prédicats. Par exemple prenant un prédicat exprimant qu'un entier minore une liste. Ce prédicat est défini par la construction suivante :

```
Fixepoint minore [x:nat; l:liste]: prop :=
   Cases l of
        nil =>true
        |(cons n l')=>(inf_eq x n)=true/\(minore x l')
   end.
```

Comme on peut utiliser la définition précédente pour définir un nouveau prédicat exprimant qu'une liste donnée est ordonnée.

## 2.3.2 Agda et Alfa

Agda est un système de preuve basé sur la théorie des types de Martin- Löf, développé par Catarina Coquand dans le langage Haskell. C'est un vérificateur de preuve, prenant en entrée une preuve (un lambda-terme typé) et une proposition ( le type du lambda terme), et le système se charge à la correction de la preuve et son adéquation avec la proposition, en adaptant l'approche type as formula. Agda peut être vu soit comme un système de vérification de preuve, soit comme un langage de programmation. Le lambda calcul est étendu avec les définitions récursives, les types algébriques, une construction par cas et les types enregistrements.

Les preuves manipulées sont de bas niveau et il n'y a pas d'automatisation. On peut l'utiliser avec Alfa, un éditeur interactif et structuré de preuve, où l'utilisateur remplis les trous pour compléter et finir la preuve (les entités insérées sont de type étendu). Le processus de démonstration est constitué de deux phases :

- La preuve est construite par l'utilisateur.

- En suite le système se charge à la vérification.

L'approche de ALFA est différente de celle de la majorité des autres systèmes : où on manipule explicitement la structure du terme de preuve [Hal, 01].

En AGDA la terminaison des programmes n'est pas garantie par constructions, mais par un critère externe : il est possible d'écrire une fonction qui ne termine pas mais cette heuristique est capable de détecter ce problème. Comme on remarque l'absence des opérateurs de récursion) et utilisation des définitions récursives et d'analyse par cas.

Le processus de preuve dans ce système fonctionne sous deux modes :

- Ecrire la spécification[1], puis la fonction d'implémentation et en fin montrer la validité de cette dernière par rapport à la spécification.
- La deuxième, semble la plus naturelle dans Agda, consiste à écrire un programme prenant en entrées les données et retournant le résultat et la preuve que ce résultat est conforme à la spécification. Dans l'exemple d'une fonction de tri sur les listes d'entiers la fonction de tri prendra une liste et retournera une liste triée et une preuve que celle–ci l'est, ou encore une valeur du type « liste_triée » garantissant l'ordre par construction des valeurs du type.

## 2.3.3 Isabelle/HOL

Développé par L. Paulson en 1986, Isabelle est l'un des descendants de LCF de Milner. Il repose fortement sur le langage d'implémentation (standard ML) qui peut être utilisé comme macro-langage pour combiner les tactiques et automatiser les procédures. Une des particularités d'Isabelle est qu'il implémente une méta-logique, une logique permettant de définir d'autres logiques. Parmi les logiques développées par ce système on trouve :

- Une logique d'ordre supérieur, appelée HOL
- Une logique du premier ordre
- La théorie des ensembles de Zermelo- Fränkel.

Cette méta-logique est une logique d'ordre supérieur basé sur la quantification universelle, l'implication et l'égalité, qui sont suffisant pour définir les règles pour les logiques cibles que l'on souhait utiliser.

---

Par exemple la conjonction est définie par :

$$\forall P,Q,\ trueprop(P \wedge Q) \Rightarrow trueprop(P)$$
$$\forall P,Q,\ trueprop(P \wedge Q) \Rightarrow trueprop(Q)$$
$$\forall P,Q,\ trueprop(P) \Rightarrow (trueprop(Q) \Rightarrow trueprop(P \wedge Q))$$

Avec *truprop* permet d'interpréter une valeur de vérité de la logique que l'on définit vers les valeurs de vérités de la méta-logique.

Isabelle/HOL c'est Isabelle avec une logique d'ordre supérieur qui permet de manipuler des termes dans un langage s'accordant à un lambda-calcul typé avec type algébrique et analyse par cas. Le langage utilisé pour manipuler les termes est utilisé pour les formules. Les valeurs booléennes du langage de programmation sont aussi les valeurs de vérité du langage et l'égalité sur les termes est la même chose que le « si et seulement si ».

Isabelle ne permet pas de construire et d'exporter des objets preuves, mais il utilise les types de donnes abstraits pour ne permettre de construire les preuves qu'avec un certain nombre de fonctions primitives, cela permet de ne faire reposer la confiance que sur ce jeu de fonctions (car toute fonction construisant une preuve ou une partie de preuve est obligée de passer par ces fonctions primitives).

### 2.3.4 Phox

Développé par Christophe Raffaëlli, basé sur une logique d'ordre supérieur, inspirée de l'arithmétique fonctionnelle d'ordre 2 (AF2) de Jean-Louis Krivine ( lambda-calcul égalité et définitions des types inductifs ).

Phox étend AF2 à l'ordre supérieur et ajoute des constructions pour simplifier les définitions inductives.

Le processus chargé à la partie preuve est basé sur le calcul des séquents et les règles de la déduction naturelle. Une particularité de Phox est la classification de ses règles en deux catégories : d'introduction et d'élimination pour chaque connecteur ou constante et règles de réécriture. Il généralise en permettant de crée de nouvelles règles d'introduction et d'élimination ou de réécriture à partir des théorèmes montrés par l'utilisateur, en fournissant une tactique automatique capable d'utiliser la récurrence. Phox construit des termes de preuve mais ne l'exporte pas, et sa logique est simple à manipuler (que par exemple le calcul des constructions).

## 2.3.5 PVS (Prototype Verification System)

C'est un assistant de preuve très utilisé notamment dans l'industrie, développé par N. Shanker et S. Owre [OSRS 01], basé sur une logique d'ordre supérieur typé avec du sous typage (un exemple de sous typage est le type attribué aux entiers paires comme un sous type des entiers naturelles) qui ressemble un peut à la théorie des ensembles. Le typage n'est donc pas décidable. Durant l'exécution, du système PVS, des obligations de preuves sont engendrées et affectées à la charge de l'utilisateur (notamment pour l'utilisation des fonctions récursives). Et si le sous-typage est utilisé, alors dans la majorité des cas les obligations de preuves peuvent être résolus automatiquement. Les données manipulées par ce système sont organisées en théories, peuvent être paramétrées. Par exemple dans le cas de la théorie sur les listes, le paramètre peut être le type des éléments de la liste.

Le langage de termes est assez riche, c'est le lambda calcul typé avec sous typage , analyse par cas et des extensions comme la conditionnelle, les enregistrements ainsi que les fonctions sont redéfinissables.

Le langage de spécification du système PVS est une logique d'ordre supérieur avec type dépendant. Ce système n'autorise pas fonctions totales, mais le sous typage nous permet de restreindre les types par un prédicat, il suffit donc d'exprimer exactement l'ensemble sur lequel est définie une fonction. Par exemple on définira la division euclidienne comme fonction totale sur l'ensemble des entiers non nuls.

Pour définir une fonction récursive, il faut préciser une mesure sur l'ensemble des arguments et vérifier que celle-ci décroît à chaque appel récursif pour justifier et assurer la terminaison.

PVS utilise la notion d'obligation de preuve, il essaie systématiquement de vérifier le typage de ce qui lui proposé et en cas d'échec, signaler à l'utilisateur qu'il est nécessaire de prouver la conjecture sur la quelle il a échoué.

Les procédures de décisions et de simplification automatiques sont assez efficaces. Aucun objet preuve n'est engendré, tout repose donc sur la conception du code du système (les sources ne sont pas disponibles). PVS est un système assez complet comprenant une interface dédiée sous X-Emacs.

### Scripte d'une preuve PVS

Une preuve dans ce système est basée sur la notion de théorie. Une théorie est une signature avec les déclarations de types et de constantes ainsi que des axiomes, des définitions

et des théorèmes. Les théories peuvent être paramétrées par des types et des valeurs. Le type des entiers est prédéfini.

```
Eq_nat_dec   : THEORY
BEGIN
  N,m: VAR nat
  Eq_nat_dec: THEOREM (n=m) OR (NOT(n=m))
END eq_nat_dec
```

La déclaration de n et m en variable de type nat permet de réaliser une quantification universelle implicite sur tous les axiomes et les théorèmes.

Le prouveur interactif (M-x prove) peut être déclenché une fois la théorie en question est présentée en entrée du système. La preuve d'un théorème, en mode interactif, ressemble à un dialogue avec PVS. Les commande ou les tactiques, appelées règles en PVS, nécessaires pour compléter la preuve sont les suivantes :

```
1   (induct "m" )
2   (induct "n")
3   (flatten-disjunct)
4   (skolem !) (flatten-disjunct)
5   (skolem !) (flatten-disjunct) (induct  "n")
6   (flatten-disjunct)
7   (skolem !) (case "j  !2=j!1 ")
8   (skolem !) (flatten-disjunct)
9   (skolem !) (flatten-disjunct)
```

A la ligne 1, on fait une récurrence sur le premier argument de type nat. En ligne 2, on est dans le cas m=0, et on fait la récurrence sur n. A la ligne 3, on doit alors prouver que : $0 = 0 \vee \neg(0 = 0)$, ce qui est fait directement par (flatten-disjunct). (flatten-disjunct) s'occupe au traitement de la disjonction. En ligne 4, on traite le cas récurent où l'on doit montrer que $\forall j : \mathrm{N}. j = 0 \vee \neg(j = 0) \rightarrow (j+1 = 0) \vee (j+1 = 0)$ .on introduit la variable de récurrence (renommée ici de n vers j par induct). On suite on introduit l'hypothèse de

récurrence avec (flatten-disjunct). En ligne 5, c'est le cas qui correspond à la partie récurrente de la première récurrence sur m, où on doit montrer :

$$\forall j : N(\forall n : N.n = j \vee \neg(n = j)) \rightarrow (\forall n : N.n = j+1 \vee \neg(n = j+1)) \ .$$

On introduit j par (skolem !) puis l'hypothèse de récurrence par (induct "n"). En ligne 6, on traite le cas de base toujours directement avec (flatten_disjunct). En fin, en ligne 7, dans le cas récurrent, on introduit la variable de récurrence j avec (skolem !) puis on raisonne par cas sur l'égalité des variables de récurrence avec (case "j !2=j!1 "). Pour plus de détails de la preuve précédente.

### 2.3.6 ACL2

Développé par Matt Kaufman et J. Strehler Moore [KM 97], ACL2 est un héritier de Nqthun de Boyer et Moore, écrit en LISP où les termes et les formules manipulés sont aussi exprimés en LISP, d'où l'absence des types, et tout est construit à partir des listes et des paires. Il permet de prouver des programmes LISP, où toutes les variables libres sont quantifiées universellement, malgré que utilisation explicite de quantificateurs est assez difficile dû principalement à l'héritage de NqThun où Nq signifie « note quantified » et qui passe par l'utilisation des lemmes engendrés par le système.

La première particularité de ce système est l'approche utilisée, où on peut travailler directement sur des programmes LISP, et comme dans PVS les fonctions sont définies d'une manière classique, mais il est nécessaire de donner ou préciser un argument justifiant la terminaison (c'est-à-dire indiquer ce qui décroît au cours des appels récursifs, qui assure la terminaison automatique du système).

La deuxième particularité est son approche de preuves, que l'on pourrait quantifier de « complètement autorisée et particulièrement bavardé ». Pour prouver une proposition dans ce système il suffit de l'énoncer, et si tout ce passe bien de patienter en lisant la description de la preuve en cours jusqu'à ce que le système termine. Lorsque le système échoue ou que l'utilisateur suspecte que la preuve se dirige vers mauvais comportement et presse Ctrl-C, pour arrêter le moteur de preuve, il est nécessaire de lire le texte de la preuve incomplète, produit par le moteur de preuve, pour repérer ce qui a pu mal se passer. Et puis soit on remédié en ajoutant avant la proposition un nouveau lemme (et en le prouvant), soit en fournissant des suggestions au système, comme par exemple « au nœud 1.5.121 de la preuve, il conviendrait de ne pas utiliser l'induction ».

Quelque soit la solution choisi, il est nécessaire de relancer la recherche de la preuve en espérant que cette fois sera la bonne, ce qui nécessite l'amélioration de l'entité heuristique, chargé à la recherche de la preuve, pour avoir une utilisation efficace de ce système et il faut aussi comprendre ce qui risque de poser comme problèmes et être capable de détecter le plus tôt possible dans le texte qui défile un échec éventuel.

Par exemple, pour montrer que la fonction de tri par insertion satisfait les deux propriétés suivantes :

- Pour toute liste d'entier « l » *( tri l )* est une liste triée.
- Pour toute liste d'entier « l » *( tri l )* est une permutation de « l ».

L'égalité est un statut particulier, elle peut être utilisée comme une règle de réécriture tel que la partie gauche contient l'expression la plus complexe, et la plus simple dans la partie droite.

La preuve, avec ce système, se fait automatiquement sans difficultés englobant l'automatisation des preuves par induction. Il est nécessaire de faire confiance aux programmeurs des tactiques de ACL2. Un des inconvénients de ce système est le fait qu'il est basé sur LISP où toutes les structures de données doivent être exprimé sous forme de paire pointé ou de liste.

## 2.4 FOLDROL

FOLDROL [Law 92] est un démonstrateur de théorèmes, conçu par Pollsson principalement pour aider à améliorer la puissance des systèmes de preuve. Le mécanisme de preuve dans ce système est basé sur Le calcul des séquents LK de Gentzen [Tak 87] qui supporte les démonstrations Backwards[1].

Le jugement à prouver est analysé selon une méthode ascendante en appliquant les règles du système aux formules du jugement en question. En fin appeler le même processus pour valider le reste des jugements (ceux qui ne sont pas trivialement valides).

### 2.4.1 Principe de base

Un jugement[2] dans le système FOLDROL a la forme suivante : $\Gamma \succ \Delta$ où $\Gamma, \Delta$ sont deux ensembles de formules.

[1] Une démonstration est dite BackWard si le processus utilisé a le but à prouver comme point de départ.

[2] Un jugement tel qu'il est généralisé dans le calcul des séquents.

L'écriture précédente signifie que *« si toutes les formules de* $\Gamma$ *sont vrais, Alors quelques formules de* $\Delta$ *sont aussi vrais »*, dans ce cas le jugement est valide.

Dans l'autre cas, s'il existe une évaluation où toutes les formules de la partie gauche sont vraies et aucune formule de la partie droite n'est vraie le jugement est faut.

Le jugement A ├A est trivialement valide appelé aussi jugement de base. Ces derniers forment les feuilles d'un arbre de démonstration.

## 2.4.2 Des preuves à la main

Pour prouver le jugement on peut suivre deux chemins, selon les priorités affectées aux règles :

1) Le premier choix nous permet d'avoir le chemin suivant :

$$\dfrac{\dfrac{A,B \mid\text{-}\ B \quad A,B \mid\text{-}\ A}{A,B \mid\text{-}\ B \wedge A}\wedge : right}{A \wedge B \mid\text{-}\ B \wedge A}\wedge : left$$

2) Le deuxième chemin nous donne :

$$\dfrac{\dfrac{A,B \mid\text{-}\ B}{A \wedge B \mid\text{-}\ B}\wedge : left \qquad \dfrac{A,B \mid\text{-}\ A}{A \wedge B \mid\text{-}\ A}\wedge : left}{A \wedge B \mid\text{-}\ B \wedge A}\wedge : right$$

Avant de penser à l'automatisation de ce processus de démonstration, il faut trouver les structures de données et les algorithmes adéquats, selon les objectifs pour représenter les entités manipulables et les processus chargés à ces manipulations.

## 2.4.3 Règles d'inférences

Parmi Les règles utilisées on trouve les règles structurelles, les règles d'éclaircissement et celles de contraction. Ces règles sont utilisées pour éclairer les relations de conséquence entre les jugements obtenus durant l'exécution du démonstrateur. Elles sont aussi utilisées pour donner plus de détails et de commentaires justifiant le chemin sélectionné durant l'extraction de la preuve.

## Les Règles Pour La Logique Propositionnelle

Il s'agit du comportement du démonstrateur avec l'environnement propositionnel, ou les formules sont construites en utilisant les connecteurs classiques ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$). Les règles d'introduction et d'élimination, associés aux connecteurs, sont indiquées dans figure suivante :

| | **Contexte Gauche** | **Contexte Droit** |
|---|---|---|
| $\wedge$ | $\dfrac{A, B \vdash}{A \wedge B \vdash}$ | $\dfrac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$ |
| $\vee$ | $\dfrac{A \vdash \qquad B \vdash}{A \vee B \vdash}$ | $\dfrac{\vdash A, B}{\vdash A \vee B}$ |
| $\rightarrow$ | $\dfrac{\vdash A \qquad B \vdash}{A \rightarrow B \vdash}$ | $\dfrac{A \vdash B}{\vdash A \rightarrow B}$ |
| $\leftrightarrow$ | $\dfrac{A, B \vdash \qquad \vdash A, B}{A \leftrightarrow B \vdash}$ | $\dfrac{A \vdash B \qquad B \vdash A}{\vdash A \leftrightarrow B}$ |
| $\neg$ | $\dfrac{\vdash A}{\neg A \vdash}$ | $\dfrac{A \vdash}{\vdash \neg A}$ |

**Figure 2-3 Règles pour la logique propositionnelle**

## Les Règles Quantifiés

| | **Contexte Gauche** | **Contexte Droit** |
|---|---|---|
| $\forall$ | $\dfrac{\forall x.A,\ A[t/x] \vdash}{\forall x.A \vdash}$ | $\dfrac{\vdash A[a/x]}{\vdash \forall x.A}$ |
| $\exists$ | $\dfrac{A[a/x] \vdash}{\exists x.A \vdash}$ | $\dfrac{\vdash \exists x.A,\ A[t/x]}{\vdash \exists x.A}$ |

**Figure 2-4 Les Règles Quantifiées**

Une condition sur l'utilisation de $\exists : Gauche$ et $\forall : Droite$ est que le paramètre « a » utilisé n'apparaît plus dans les conclusions.

## Les Paramètres Dans Les Règles Quantifies

Eviter les erreurs dû à la manipulation des variables capturés[1], est l'une des conditions principales, qu'il faut vérifier, avant de déclencher le processus de substitution. Autrement dit que ce processus ne transforme jamais une variable libre à une variable lié. Ce qui permet d'aboutir à des déductions erronées, comme dans l'exemple suivant :

$$\frac{\vdash \forall x.\exists y.y \neq x}{\vdash \exists y.y \neq y}\ \forall : Droit$$

Durant la substitution de x par y dans la sous formule « $\forall x.\exists y.y \neq x$ » la variable libre « y » devenu lié par le quantificateur « $\exists y$ ».

Pour résoudre ce type de problème, la solution la plus utilisée, consiste à renommer les variables liés, l'application de cette méthode à l'exemple précédant, consiste à renommer la variable « y » par « z » et on aura une déduction correcte : « $\exists z.z \neq y$ ».

Cette solution permet d'éviter les erreurs capturées, mais l'algorithme qui l'implémente est trop compliqué et peut aboutir à des résultats erronés.

## Les paramètres et les variables :

Une autre manière pour traiter les problèmes des variables capturés, consiste à introduire une distinction entre les paramètres et les variables. Une occurrence d'un paramètre ne doit être jamais liée. Les paramètres sont notés par a, b, c,…

Par contre, une variable doit être liée. On note les variables par x, y, z, …

Remarquons, Sous cette distinction, que quelques sous formules peuvent être des formules non légales.

**Exemple :**

L'instance « $\forall x.\exists y.y \neq x$ » a la formule « $\exists y.y \neq x$ » comme sous formule, où « x » apparaît libre, donc il faut remplacer cette occurrence par un paramètre « a ».

L'application de « $\forall : Droit$ » à l'instance « $\forall x.\exists y.y \neq x$ » produira la conclusion « $\vdash \exists y.y \neq a$ »

[1] Le problème des variables capturées : après une substitution, des variables libres deviennent liées.

## Application et traitements des règles quantifiés

Le paramètre « a » introduit par les règles « $\forall : Droit\ ou\ \exists : Gauche$ » ne doit pas apparaître dans la conclusion. Autrement dit que « a » peut contenir une valeur arbitraire et que le choix du paramètre « a » n'influe pas sur les résultats.

Par contre l'instanciation des méta-variables peut influer sur les conclusions à l'ajout des nouveaux paramètres.

**Exemple :**

Le jugement « $\forall x.R(x,x) \mid - \exists y.\forall x.R(x,y)$ » n'est pas valide. Pour le montrer, affectant l'égalité (x=y) comme interprétation de R(x,y).

$$\dfrac{\dfrac{\dfrac{R(?c, ?c) \mid\text{-} R(b, ?a)}{ALLx.R(x,x) \mid\text{-} R(b, ?a)}\ \forall :gauche}{ALLx.R(x,x) \mid\text{-} ALLx.R(x, ?a)}\ \forall :droite}{\forall x.R(x,x) \mid\text{-} \exists y.\forall x.R(x,y)}\ \exists :droite$$

Il est claire que le remplacement de « $?c\ et\ ?a\ \ par\ b$ » finir la preuve avec le jugement : $R(b,b) \mid - R(b,b)$.

Si un paramètre « b » dépend d'une méta variable « ?a », alors le paramètre « b » doit être différent des paramètres de chaque terme substitue par « ?a ».

L'application de la règle « $\forall : Droit\ ou\ \exists : Gauche$ » introduire un paramètre « b ». Une liste de meta-variables : « $?a_1$, $?a_2$ ,…, $?a_n$ », dont les éléments figurent dans la conclusion, est associée à ce paramètre pour signaler sa dépendance de l'ensemble des variables de la conclusion.

On écrit b[$?a_1$, $?a_2$ ,…, $?a_n$] pour signaler cette dépendance.

Avec cette notion, une nouvelle condition sur l'unification d'un couple ( ?a, t) est que la méta-variable « ?a » n'apparaît pas dans la liste de dépendance du paramètre « t », La détection des occurrences permet de préjuger des assignations non valides à des méta-variables.

FOLDROL, associe à chaque paramètre, une liste des méta-variables, appelée liste des dépendances.

***Pas de dépendance d'ordre 2 :***

Chaque nouveau paramètre introduit dépend de toutes les méta-variables du jugement en cours. Ce dernier peut contenir d'autres paramètres de type b[ ?a] mais il n'est pas nécessaire que ce paramètre contient la variable ?a.

Soit la preuve suivante :

$$\dfrac{\dfrac{|- P(?a, b[?a]) \qquad \dfrac{|- Q(b[?a], c)}{|- \forall y Q(b[?a], y)} \forall : Droit}{|- P(?a, b[?a]) \wedge \forall y Q(b[?a], y)} \wedge : Droit}{|- \forall x. P(?a, x) \wedge \forall y Q(x, y)} \forall : Droit$$

Le paramètre « b » dépend de la variable « ?a ». Le jugement « $|- \forall y. Q(b[?a], y)$ » contient le paramètre « b[ ?a] » et non pas la variable « ?a ». et le paramètre « c » ne dépend pas de « ?a ».

Dans l'exemple précédent, le champ d'exécution du processus de démonstration est limité à la partie droite du jugement (la conclusion). C'est pourquoi FOLDROL ignore la dépendance entre le paramètre « c » et la méta-variable « ?a ».

## 2.4.4 L'inférence

La construction des preuves sous FOLDROL est basée sur une méthode upward, avec laquelle le but à atteindre est considéré comme point de départ et l'application à chaque étape de la règle appropriée permet d'assurer la transition du système d'un état à un autre. Selon le champ d'action de la règle appliquée, des changements sont apportés sur l'ensemble de jugements du système à un instant donné. Soit il est augmenté par la décomposition du jugement sélectionné en un ensemble de sous jugements, soit il est réduit par la validation des jugements qui sont trivialement valides.

Le processus de preuve s'achève et la preuve est complète s'il n'y a pas des jugements à prouver.

Durant l'exécution du démonstrateur, la transition du système d'un état vers un autre est assurée par l'exécution séquentielle des tâches suivantes :

- La sélection d'un jugement.
- La sélection de la règle d'inférence à appliquer.
- Construction et validation des sous jugements.

## Représentation d'un jugement

Un jugement est représenté par une liste de triplets. Chaque élément de cette liste contient trois champs : le poids et le coté (gauche ou droite) de la formule et la formule.

L'état de la preuve, appelé *goaltable,* est constitué d'une liste de jugements à prouver et un ensemble d'informations décrivant des contraintes sur les jugements.

## La preuve d'un jugement

Un jugement sous FOLDROL est représenté par deux listes, chaque liste contient un ensemble de formules, noté par :

$$A_1, A_2, ..., A_n \mid\!\!- B_1, B_2, ..., B_m \, .$$

La première liste (formée par les $A_i$ $_{i=1...n}$) correspond à l'ensemble d'hypothèses et l'autre (formée par les Bi i=1…m) correspond à l'ensemble des conclusions.

La vérification et la validation du jugement précédant consiste à trouver un couple $(A_i,B_j)$ de formules, il n*m possibilités, pour lequel processus d'unification est exécuté avec succès. Si ce couple existe alors le jugement est validé.

Pour minimiser le temps de calcul[1], FOLDROL considère uniquement les formules atomiques.

### a) Sélection d'une Règle

Une fois le jugement à prouver est sélectionné[2], l'étape suivante consiste à sélectionner la règle à appliquer. Cette sélection est l'une des étapes les plus importantes du fonctionnement d'un démonstrateur de théorèmes. Selon les objectifs à atteindre : des politiques et des stratégies sont définies pour minimiser le temps d'exécution du système, jouer sur la taille de la démonstration et/ou minimiser la prolifération des sous jugement etc.

Les critères de sélection sous FOLDROL sont basés sur le degré de connectivité de chaque formule ainsi que la règle qui engendre un minimum des jugements est prioritaire que les autres. Les règles associées aux quantificateurs, particulièrement $\forall$ : *Gauche et* $\exists$ : *Droit* ont la priorité la plus faible.

Pour gérer le reste des conflits une politique d'arbitrage est introduite. Cette stratégie est définie autour d'une fonction appelée « *cost ».* La valeur de cette fonction représente le poids de la formule présentée comme argument.

[1] Le temps de calcul nécessaire pour trouver une instance commune des formules en questions.

[2] Initialement le but sélectionné est le but globale à prouver.

### b) Construction des sous Jugement

Le calcul de séquent LK est utilisé pour performer les preuves backward, dans laquelle chaque formule de la preuve est une sous formule du jugement de départ, sauf le traitement particulier pour les règles quantifiées ( $\forall$ : *Gauche et* $\exists$ : *Droit* ).

Comme nous l'avons déjà signalé, un jugement est représenté par une liste des triples (formule, côté,cost). La règle sélectionnée est appliquée à la tête de cette liste. L'application de cette règle engendre un ensemble des sous jugements.

### c) Sélection d'un Jugement

Le processus de démonstration échoue à prouver l'entité en question, si quelques jugements engendrés sont improuvables. D'où pour améliorer l'efficacité d'un démonstrateur, il est préférable de commencer par les jugements qui semblent très probablement improuvables.

La sélection du jugement, sous FOLDROL, à traiter est basé sur une des politiques les plus simples, la politique LIFO. Selon cette politique le démonstrateur favorise le traitement des derniers jugements construits.

Au niveau de chaque étape, FOLDROL remplace le jugement apparaît en tête de liste par les sous jugements engendrés par l'application de la règle sélectionnée.

Les étapes suivantes illustre le fonctionnement général du système :

- L'appel de la fonction *proof_steep*, nous permet de repérer le jugement à traiter.
- Le jugement localisé est présenté en entré de la fonction *reduce_goal, pour* appliquer la règle appropriée.
- En fin, l'appel de la fonction *insert_goals* nous permet la transition du comportement du système vers un nouvel état.

La fonction *proof_steep* avec un paramètre « n>0 », nous permet de visualiser le comportement du système après l'exécution de « n » étape et produire un journal, contient l'ensemble d'opérations exécuté durant chaque étape $E_{i\ (i=1\ldots n)}$.

FOLDROL manipule deux type d'objets : les séquents et les formules. Le premier type pour vérifier la validité de la déduction où la partie gauche contient un ensemble d'hypothèses et la partie droite contient les conclusions. Le deuxième type est réservé à la vérification de la validité d'une formule du système (les théorèmes).

## 2.4.5 Limites du Système FOLDROL

La présentation de la session conçue pour l'utilisation du système FOLDROL a deux objectifs principaux :

- La taille du dialogue utilisateur/machine doit être le plus compact possible.
- Signaler, avec des indicateurs, les sections définies par l'utilisateur (elles sont fréquents pour les systèmes assistés) et celles obtenu par le système. Pour FOLDROL les lignes qui commencent par (« > » ou « # ») sont introduites pas l'utilisateur et les autres sont des résultats obtenus par le système

***Traitement des entités propositionnelles***

Le fonctionnement de FOLDROL dans un environnement propositionnel est testé sur l'ensemble publié par Pelletier [Pel 86], un ensemble particulier des problèmes de la logique classique du premier ordre conçu spécialement pour tester les démonstrateurs de théorèmes. L'exécution de FOLDROL sur une machine Sun-3, ne nécessite que 0,1 seconde pour prouver un théorème de la logique propositionnelle.

***Traitement des entités quantifiées***

La plupart des structures de données définies, durant la conception du système, ont été conçues autour des quantificateurs pour assurer un bon fonctionnement, particulièrement les restrictions dues aux manipulations des paramètres et des méta-variables. FOLDROL peut prouver l'un des théorèmes les plus compliqués, comme il peut échouer avec l'un des plus simple.

## 2.4.6 Tactiques et Méthodes Automatiques

Le but du système présenté est d'illustrer les techniques de codage utilisées dans la représentation des entités manipulées par la majorité des démonstrateurs de théorème, sa force principale est sa simplicité.

Il est possible de performer les méthodes pour traiter les taches critiques de l'implémentation du FOLDROL. Parmi les quelles on peut citer :

- La fonction *solve_goal* prend toujours la première solution trouvée, ce qui peut bloquer le processus de preuve avec d'autres jugements. Quelques fois le démonstrateur est obligé de choisir un autre chemin pour réussir. Le système ignore toutes les solutions en laissant la preuve ouverte, autrement dit : parcourant l'espace des possibilités selon la politique « Backtracking » où Prolog peut intervenir efficacement pour implémenter cette politique.
- Pour traiter les jugements contenant des formules quantifiées, où les règles $\forall : Gauche$ et $\exists : Droit$ sont applicables, on utilise un type particulier des prédicats : $\Pr oof * (As, Bs, N, P)$ avec N représente le nombre maximal d'application des deux règles précédentes, P représente l'arbre de la preuve. Le nombre N limite l'espace recherche où un parcours avec retour arrière « Backtracking » peut l'explorer facilement.

3 Le Système E_Lam

# Chapitre 3

# Le Eλ-Calcul

Décrit dans [MC 02], le système Eλ est une extension du λ_calcul pur, où deux constantes sont introduites, une pour représenter l'implication et l'autre pour la quantification universel. Le système obtenu est assez riche, dans le sens où les deux constantes introduites sont suffisantes pour exprimer ou définir le reste des connecteurs.

La consistance du système Eλ est garantie grâce à l'affection d'un nouvel attribut, appelé niveau d'un terme, utilisé pour introduire la nouvelle définition du processus de substitution ; où le terme *[N/x]M* n'est défini que si le niveau du terme *« N »* est inférieur ou égal à celui de *« x »*. Cette restriction nécessite une définition propre du mécanisme de réduction, appelé Eλβ-reduction, qui vérifie la propriété de Church-Rosser et offre un moyen pour éviter le paradoxe de Curry. Le Eλ_calcul est aussi un système avec lequel on peut interpréter les logiques d'ordre supérieur.

Dans cette partie, nous allons présenter les fondations de cette extension ainsi que les grandes lignes permettant d'affirmer la richesse, la puissance et la consistance de cette extension.

## 3.1 Les Termes du Système Eλ

L'ensemble des termes du système Eλ est l'union des ensembles $C_0$,$C_1$, $C_2$… Chaque ensemble $C_i$ est constitué par les éléments de l'ensemble $C_{i-1}$ et quelques termes construits par l'introduction des deux constantes du Eλ_calcul.

L'ensemble $C_0$, défini trivialement, est l'ensemble des termes du λ_calcul classique dont les variables sont notées par : $x^0$, $y^0$, $z^0$…

## **Définition 3.1** *Les termes de l'ensemble $C_0$*

L'ensemble $C_0$ est défini inductivement sur l'ensemble infini dénombrable $V_0 = \{x^0, y^0, z^0, ...\}$ par l'utilisation de l'application et λ-abstraction de la manière suivante :

- Si $x \in V_0$ Alors $x \in C_0$ ;
- Si $M, N \in C_0$ Alors $(MN) \in C_0$ ;
- Si $M \in C_0, x \in V_0$ Alors $\lambda x.M \in C_0$ ;

## **Définition 3.2** *Les termes d'un ensemble $C_i$*

Un ensemble $C_i$ est défini par induction, sur l'ensemble infini dénombrable $V_i = V_{i-1} \cup \{x^i, y^i, z^i, ...\} (i \geq 1)$ en utilisant les constantes P (qui représente l'implication) et Π (qui représente la quantification universelle), de la manière suivante :

- ✓ Si $x \in V_i$ Alors $x \in C_i$ ;
- ✓ Si $M \in C_i, N \in C_j$ Alors $(MN) \in C_{\max(i,j)}$;
- ✓ Si $M \in C_i, x \in V_j$ Alors $\lambda x.M \in C_{\max(i,j)}$ ;
- ✓ Si $X \in C_i$ Alors $\Pi X \in C_{\max(1,i)}$;
- ✓ Si $X \in C_i, Y \in C_j$ Alors $PXY \in C_{\max(i,j)+1}$.

## **Définition 3.3** *les termes du système Eλ « $C_w$ »*

$C_w$ est défini par l'union des ensembles $C_i$ $_{i \geq 0}$ $C_w = \bigcup_{i<\infty} C_i$.

## **Définition 3.4** *(niveau d'un Eλ-terme)*

Le niveau d'un Eλ-terme est calculé par les règles suivantes :

$$
\begin{aligned}
&Niveau(x^i) = i \\
&Niveau(\lambda x.M) = Max(Niveau(x), Niveau(M)). \\
&Niveau(PXY) = Max(Niveau(X), Niveau(Y)) + 1. \\
&Niveau(\Pi X) = Max(1, Niveau(X)). \\
&Niveau(XY) = Max(Niveau(X), Niveau(Y)).
\end{aligned}
$$

## Définition 3.5 *(la congruence)*

On dit que P est congruent à Q et on note $(P \equiv_\alpha Q)$ si et seulement si Q peut être obtenu, à partir de P, par une suite finie de changements de variables liées de même niveau.

Dans le système Eλ, le changement de variables n'est possible que si les deux variables ont le même niveau, d'où les termes $\lambda x.(xx \supset Z)$ $et$ $\lambda y.(yy \supset Z)$ ne sont pas congruent si x et y n'ont pas le même niveau.

## Définition 3.6 *(Eλβ-réduction )*

Le processus de réduction dans le Eλ-Calcul est défini par les règles suivantes :

$$
\begin{array}{l}
(\rho) : M \rightarrow_{E\lambda\beta} M. \\
(\mu) : M \rightarrow_{E\lambda\beta} N \Rightarrow ZM \rightarrow_{E\lambda\beta} ZN. \\
(\mu') : M \rightarrow_{E\lambda\beta} N \Rightarrow X \supset M \rightarrow_{E\lambda\beta} X \supset N. \\
(\mu'') : M \rightarrow_{E\lambda\beta} N \Rightarrow \Pi M \rightarrow_{E\lambda\beta} \Pi N. \\
(\nu) : M \rightarrow_{E\lambda\beta} N \Rightarrow MZ \rightarrow_{E\lambda\beta} NZ. \\
(\nu') : M \rightarrow_{E\lambda\beta} N \Rightarrow M \supset X \rightarrow_{E\lambda\beta} N \supset X. \\
(\tau) : M \rightarrow_{E\lambda\beta} N, N \rightarrow_{E\lambda\beta} T \Rightarrow M \rightarrow_{E\lambda\beta} T. \\
(E\alpha) : \lambda x.M \rightarrow_{E\lambda\beta} \lambda y.[y/x]M \ si \ Niveau(x) = Niveau(y) \ et \ y \ est \ non \ libre \ dans \ M. \\
(E\beta) : (\lambda x.M)N \rightarrow_{E\lambda\beta} [N/x]M \ si \ Niveau(N) \leq Niveau(x) \ or \ Niveau(\lambda x.M) = 0. \\
(E\xi) : M \rightarrow_{E\lambda\beta} N \Rightarrow \lambda x.M \rightarrow_{E\lambda\beta} \lambda x.N.
\end{array}
$$

## 3.2 Processus d'inférence

- Une formule dans le Eλ-Calcul est un Eλ-terme.
- Un contexte est un ensemble fini de formules. et les hypothèses sont les formules d'un ensemble fini.
- Si Γ est un contexte et X est une formule, on dit que X est dérivable à partir de Γ, et on écrit $\Gamma \vdash X$ , si on peut dériver la formule X par le système de déduction.
- Le système de déduction du système Eλ est décrit par les règles suivantes :

$(ax)$: $X \in \Gamma \Rightarrow \Gamma \mid\!- X.$

$(Eq_{E\beta})$: $\Gamma \mid\!- X,\ X =_{E\lambda\beta} Y, Niveau(Y) \leq Niveau(X) \Rightarrow \Gamma \mid - Y.$

.$(P_e)$: $\Gamma \mid\!- X \supset Y, \Gamma \mid\!- X \Rightarrow \Gamma \mid\!- Y.$

$(P_i)$: $\Gamma, X \mid\!- Y \Rightarrow \Gamma \mid\!- X \supset Y.$

$(\Pi_e)$: $\Gamma \mid\!- \Pi(\lambda x^i.X) \Rightarrow \Gamma \mid\!- (\lambda x^i.X)Y^k$ *avec* $k \leq i.$

$(\Pi_i)$: $\Gamma \mid\!- M \Rightarrow \Gamma \mid\!- \Pi(\lambda x.M)$ *avec* $x$ *non libre dans* $\Gamma.$

## 3.3 Les connecteurs & les quantificateurs logiques

$$
\begin{array}{lll}
(\forall x^i)X & \equiv & \Pi(\lambda x^i.X) \\
\bot^i & \equiv & \forall x^i(x^i) \\
\neg^{i+1} & \equiv & \lambda x^i.(x^i \supset \bot) \\
\Xi^{i+1} & \equiv & \lambda x^i x^i.\forall z^i(x^i z^i \supset y^i z^i) \\
F^{i+1} & \equiv & \lambda x^i y^i z^i.\forall u^i(x^i u^i \supset y^i(z^i u^i)). \\
G^{i+1} & \equiv & \lambda x^i y^i z^i.\forall u^i(x^i u^i \supset y^i u^i(z^i u^i)). \\
\wedge^{i+3} & \equiv & \lambda x^i y^i.\forall z^i((x^i \supset (y^i \supset z^i)) \supset z^i). \\
\vee^{i+3} & \equiv & \lambda x^i y^i.\forall z^i((x^i \supset z^i) \supset (y^i \supset z^i)) \supset z^i. \\
EQ^{i+1} & \equiv & \lambda x^i y^i.\forall z^i((x^i x^i \supset z^i y^i))^i.
\end{array}
$$

Si on interprète le terme XY par « $Y \in X$ » ou « Y satisfait le prédicat X, alors on interprète le terme $\Xi XY$ par $X \subseteq Y$ *ou* $\forall x(Xx \supset Yx)$.

On peut exprimer la constante F en fonction de $\Xi$, avec laquelle on peut représenter les fonctions et les types. Par exemple, le terme « FabX » est interprété par « *la fonction X définie sur a vers b* ».

Remarquons que les connecteurs et les quantificateurs sont des Eλ-termes avec des niveaux différents : pour un connecteur nous avons une infinité des Eλ-termes. Par exemple pour la négation nous avons : $\neg^1, \neg^2, ..., \neg^i, \neg^{i+1}, ...$ et pour définir le terme qui représente la négation du terme X de niveau k, il faut appliquer le connecteur $\neg^{i+1} \equiv \lambda x^i.x^i \supset \bot$ *avec* $k \leq i$ et on aura : $(\lambda x^i.x^i \supset \bot\ X) \rightarrow_{E\lambda\beta} X \supset \bot$.

Le système Eλ est un système des logiques typées, et la restriction de l'ensemble des termes aux termes d'un ensemble $C_i$ (de niveau inférieur ou égale à i) dénote le sous système $E\lambda_i$ appelé souvent « système logique de type i ».

Le concept de la logique typée est utilisé aussi par A. Markov[Mar 71] et A. Chauvin [Cha 89].

Comme nous l'avons signalé ci-dessus, les deux constantes du système, correspondent au quantificateur universel et à l'implication, peuvent être vus comme un système complet dans le sens où on peut exprimer les autres connecteurs ($F, \Xi, G, \vee, ...$) en fonction de ces deux constantes, comme on va le voir par la suite.

## Proposition 3.1

Si a, b, X et Y sont des éléments de l'ensemble $C_i$, alors :

i. $F^{i+1}abX, aY \mid\text{-}\ b(XY);$

ii. $aY \mid\text{-}\ b(XY) \Rightarrow F^{i+1}abX;$

iii. $G^{i+1}XYZ, Xw \mid\text{-}\ Yw(Zw);$

iv. $Xw \mid\text{-}\ G^{i+1}XYZY;$

**Interprétations :**

1) Si X est la preuve, un habitant, de l'expression $F\alpha\beta$ du Eλ-Calcul (qui représente la proposition $\alpha \rightarrow \beta$) et Y est une preuve de α, alors l'application de la preuve X à la preuve Y est une preuve de l'expression β.
2) Si à partir d'une hypothèse α dont la preuve des Y on peut dériver la formule β dont la preuve est une application d'une preuve X à une preuve Y (noté XY), alors la preuve X est un habitant de l'expression $F\alpha\beta$ du Eλ-Calcul (qui correspond à la proposition $\alpha \rightarrow \beta$).
3) Si une expression Y du Eλ-Calcul est quantifiée universellement, par une variable d'un domaine X, a une preuve Z (qu'on note $GXYZ$) et w est une preuve de X, alors la preuve de l'application de cette expression à w (Yw) a l'application (Zw) comme preuve.

## Proposition 3.2

Si X et Y deux termes de même niveau « i », alors :

i. $EQ^{i+1}XY \mid\text{-}\ EQ^{i+1}YX;$

ii. $EQ^{i+1}XY, EQ^{i+1}YZ \mid\text{-}\ EQ^{i+1}XZ;$

iii. $EQ^{i+1}XY \mid\text{-}\ EQ^{i+1}(ZX)(ZY).$

iv. $EQ^{i+1}XY \mid\text{-}\ EQ^{i+1}(XZ)(YZ).$

## 3.4 La Consistance du Système Eλ

### 3.4.1 Le paradoxe de Curry

Soient X et Y deux termes avec Y est défini par :

$$Y \equiv (\lambda x.(xx) \supset X)(\lambda x.(xx) \supset X) \text{ Alors } Y =_{\lambda\beta} (Y \supset X)$$

Donc on peut dériver $\mid\text{-}\ X$.

La suite des dérivations suivante montre comment un processus de réduction peut aboutir au paradoxe de Curry.

| | | |
|---|---|---|
| 1. | $Y \vdash Y$ | |
| 2. | $Y \vdash Y \supset X$ | $TQ \quad Y =_{\lambda\beta} (Y \supset X)$ |
| 3. | $Y \vdash X$ | $\supset -\text{élimination}$ |
| 4. | $\vdash Y \supset X$ | $\supset -\text{introduction}$ |
| 5. | $\vdash Y$ | $TQ \quad Y =_{\lambda\beta} (Y \supset X)$ |
| 6. | $\vdash X$ | $\supset -\text{élimination}$ |

Dans le système Eλ, grâce aux restrictions portées sur le processus de réduction, le terme Y défini précédemment $Y =_{E\lambda\beta} (Y \supset X)$ ( ce terme est utile pour démontrer le paradoxe de Curry), ne peut pas être construit :

Dans le Eλ-terme « $Y \equiv (\lambda x.(xx) \supset X)(\lambda x.(xx) \supset X)$ » : si « i » est le niveau de « x » et « j » le niveau de « $(\lambda x.(xx) \supset X)$ » par construction de le niveau : « j » est supérieur au niveau i.

4 Le démonstrateur

## *Chapitre 4*

# Le démonstrateur Eλ-Prover

Nous présentons dans ce chapitre les détails de la procédure de démonstration utilisée dans l'implémentation de notre système, un démonstrateur de théorèmes d'ordre supérieur basé sur le E_lambda calcul. D'abord, nous allons décrire brièvement la procédure utilisée notamment le comportement dans un environnement trivial et la généralisation de ce mécanisme.

La procédure de démonstration est définie trivialement sur les deux constantes du E_lambda calcul. Cette procédure reçoit la formule à prouver en entrée et consiste à assigner une preuve pour chaque sous-formule de cette formule, et par la suite à construire le but principal et les sous buts (en se basant sur le calcul des séquents). La validation des buts élémentaires nécessite la résolution du système d'équations correspondant. Pour résoudre ce système, nous allons proposer et présenter une nouvelle stratégie à travers la définition de l'univers HABIT associé à la formule en question.

## 4.1 Procédure de démonstration

Le principe de base de la procédure de démonstration utilisée est inspiré principalement de la puissance des deux constantes du E_lambda calcul, présentée dans la section 3.3. Cette procédure peut être vue comme l'exécution séquentielle des taches suivantes :

i. Transformation de la formule à prouver.
ii. Construction du but principal.
iii. Construction des sous buts (buts élémentaires)
iv. Construction et résolution du problème d'unification.

### 4.1.1 Transformation de la formule à prouver

Soit α la formule à démontrer. Prouver α signifie construire le terme X du Eλ tel que l'expression (1) est dérivable.

$$\vdash \alpha X \cdots (1)$$

Pour prouver (1) nous allons la réécrire en termes d'implications en utilisant seulement le connecteur $\supset$ .

Pour cela on procède par induction sur la structure de la formule (1) comme suit, en se basant sur les définitions des constantes F et G (§3.3).

**i. L'implication et la constante F**

Si la formule en question contient la formule $A \to B$ comme sous formule et si X est la preuve associée à cette dernière, et on écrit $(A \to B)X$ . Pour définir les preuves des sous formules A et B on utilise la définition de la constante F de la section 3.3.

$$F \equiv \lambda xyz.\forall u(xu \supset y(zu))$$

$$\begin{aligned}(A \to B)X &\equiv (F\ A\ B\ )X \\ &\equiv \lambda xyz.\forall u(xu \supset y(zu))A\ B\ X \\ &\equiv \forall u(Au \supset B(Xu)) \\ &\equiv (Au \supset B(Xu))\end{aligned}$$

La définition des preuves pour les sous formules d'une implication est donnée par :

$$\frac{\begin{matrix}\vdots & & \vdots \\ (A \to B) : X & & A : u\end{matrix}}{A : u \supset B : (Xu)}$$

Cette définition est interprétée de la manière suivante :

Si X est un habitant[1] d'une formule $A \to B$ , alors un habitant de la proposition « B » est défini par l'application de l'habitant « X » (qui peut être vu comme une preuve) à celui de la proposition « A » .

[1] Un habitant d'une proposition est un terme qui peut être vu comme la preuve de cette dernière.

**ii. La Quantification universelle et la constante G**

Si la formule en question contient la formule $(\forall x \in \mathrm{N})A$ comme sous formule et si X est la preuve associée à cette dernière, et on écrit $(\forall x \in \mathrm{N})A\,X$ . Pour définir la preuve de formule A on utilise la définition de la constante G de la section 3.3.

$$G \equiv \lambda xyz.\forall u(xu \supset yu(zu))$$

$$\begin{aligned}(\forall x \in \mathrm{N})A\,X &\equiv G\mathrm{N}(\lambda x.A)X \\ &\equiv \lambda xyz.\forall u(xu \supset yu(zu))\mathrm{N}(\lambda x.A)X \\ &\equiv \forall u(\mathrm{N}u \supset (\lambda x.A)u(Xu)) \\ &\equiv \mathrm{N}u \supset (\lambda x.A)u(Xu)\end{aligned}$$

L'interprétation du traitement de la quantification universelle est la suivante :

Si X est un habitant d'une formule $(\forall x \in \mathrm{N}(A)) : X$ , alors l'habitant de la proposition « A » est défini par l'application de l'habitant « X » à celui d'un élément « t » du domaine « N ».

**iii. Connecteurs**

Le traitement des connecteurs consiste à exprimer chaque connecteur en fonction de l'implication et la quantification universelle. Ce traitement est déjà présenté dans la section 3.2 du chapitre 3.

$$\begin{aligned}(\neg A) &\equiv (A \rightarrow \bot) \\ (A \wedge B) &\equiv \forall P.(A \rightarrow (B \rightarrow P) \rightarrow P) \\ (A \vee B) &\equiv \forall P.(A \rightarrow P)((B \rightarrow P) \rightarrow P)\end{aligned}$$

**iv. La Quantification Existentielle.**

Le traitement de cette quantification est inspiré de la relation classique entre les deux quantifications et de la définition précédente de la négation:

$$\begin{aligned}(\exists x \in \mathrm{N}.P(x)) &\equiv (\neg\neg(\exists x \in \mathrm{N}.P(x))) \\ &\equiv (\neg(\forall x \in \mathrm{N}.\neg P(x))) \\ &\equiv (\neg(\forall x \in \mathrm{N}.(P(x) \rightarrow u))) \\ &\equiv ((\forall x \in \mathrm{N}.(P(x) \rightarrow u)) \rightarrow u)\end{aligned}$$

$$(\exists x \in \mathrm{N}(A(x))) \equiv (\forall x \in \mathrm{N}((A(x) \rightarrow u) \rightarrow u))$$

## 4.1.2 Construction des buts élémentaires

La formule F ainsi obtenue, par l'application du processus de transformation décrit précédemment, est appelée but principal. L'étape suivante consiste à transformer F, en appliquant les règles, du calcul des séquents, suivantes, en un ensemble des buts :

$$\frac{\Gamma, A:u \;\; |\!- B:(vu)}{\Gamma \;\; |\!- A:u \supset B:(vu)} \qquad\qquad \frac{\Gamma \;\; |\!- A:u \qquad \Gamma,\; B:(vu) \;\; |\!- C:X}{\Gamma,\; (A:u \supset B:(vu)) \;\; |\!- C:X}$$

A l'issue de l'étape précédente nous obtenons un ensemble des buts élémentaires à valider de la forme suivante :

$$\begin{cases} \alpha_1^1 u_1^1,\, \alpha_1^2 u_1^2 ,\cdots, \alpha_1^{n1} u_1^{n1} \;\; |\!- \beta_1 v_1 \\ \quad\vdots \\ \alpha_i^2 u_i^2,\, \alpha_i^2 u_i^2 ,\cdots, \alpha_i^{ni} u_i^{ni} \;\; |\!- \beta_i v_i \\ \quad\vdots \\ \alpha_m^2 u_m^2,\, \alpha_m^2 u_m^2 ,\cdots, \alpha_m^{nm} u_m^{nm} \;\; |\!- \beta_m X v_m \end{cases} \qquad \cdots\cdots(2)$$

La validation de l'ensemble des buts du système (2) nécessite la définition et la résolution de l'un de ses problèmes d'unifications. Un problème d'unifications [MS 05] est défini par l'ensemble des couples à unifier pour valider tous les buts du système (2). La validation de chaque but nécessite l'unification de l'une de ses hypothèses avec sa conclusion ce qui permet de définir un système de la forme suivante :

$$\begin{cases} \left( \alpha_1^{k_1} u_1^{k_1} \;, \beta_1 v_1 \right) \\ \vdots \\ \left( \alpha_i^{k_i} u_i^{k_i} \;, \beta_i v_i \right) \\ \vdots \\ \left( \alpha_m^{k_m} u_m^{k_m} \;, \beta_m v_m \right) \end{cases} \Rightarrow \begin{cases} \left( u_1^{k_1} = v_1 \right)_{\alpha_1^{k_1} = \beta_1} \\ \vdots \\ \left( u_i^{k_i} = v_i \right)_{\alpha_i^{k_i} = \beta_i} \\ \vdots \\ \left( u_m^{k_m} = X v_m \right)_{\alpha_m^{k_m} = \beta_m} \end{cases} \qquad \cdots\cdots\left(R^{(0)}\right)$$

Pour résoudre le système d'équations $R^{(0)}$ nous allons définir l'univers HABIT associé à la proposition en question. Pour cela les définitions suivantes sont nécessaires pour comprendre le principe de cette stratégie.

## Définition 4.1 (le processus d'unification)

Soient deux termes $t_1$ et $t_2$. On dit que $t_1$ et $t_2$ sont unifiables si et seulement si il existe une substitution $\partial$ telle que $\partial t_1 = \partial t_2$

$\partial$ s'appelle un unifieur de $t_1$ et $t_2$ ou une solution du problème d'unification de $t_1$ et $t_2$.

## Propriété 4.1

Le problème d'unification des termes du premier ordre est décidable. S'il existe des solutions, il en existe une plus générale que toutes les autres [Rob 65].

## Propriété 4.2

Le problème d'unification des lambda-termes typés d'ordre supérieur est semi-décidable. S'il y a des solutions, il n'existe pas forcément l'unifieur le plus général [Hue 75].

Une précision sur la semi-décidabilité de l'unification des termes d'ordre supérieur est que : si le problème possède des solutions il est possible de les trouver et par contre il n'est pas possible de décider qu'il n'en existe pas.

## Définition 4.2 (sous formule)

L'ensemble des sous formules d'un type est défini inductivement de la manière suivante :

i. $Sous_Formules(T) = \{T\}$ si T est une formule atomique.

ii. $Sous_Formules(A \to B) = \{A \to B\} \cup Sous_Formules(A) \cup Sous_Formules(B)$

## Définition 4.3 (ensemble de variables)

L'ensemble de variables *V* de l'univers *HABIT (T, V, σ, R)* est l'ensemble des termes utilisés pour définir les habitants des sous formules du type T. (un éléments de cet ensemble peut être vu comme un terme rigide [Hue 75]).

## Définition 4.4 (paramètres d'un habitant)

Une variable v est dite paramètre d'un habitant du type T, si et seulement si cette variable apparaît comme argument pour le terme X.

## Définition 4.5 (les règles de substitutions σ)

L'ensemble des règles de substitution σ de l'univers HABIT *(T, σ, R, X)* est défini de la manière suivante :

Si $H_1$ et $H_2$ sont deux habitant d'une sous formule atomique α et $H_2$ contient au moins un paramètre de l'habitant X alors : $(H_1 \leftarrow H_2)$ est une règle de substitution de l'ensemble σ.

Le but principale de cette stratégie est d'exprimer les résultats des fonctions en fonction des arguments et qui permet définir uniquement les termes clos.

## Définition 4.6 *(ordre d'une relation)*

L'ordre d'une relation est un entier qui représente le nombre d'applications successives des substitutions σ. $R^{(i)}$ dénote l'ensemble des relations d'ordre i. Cet ensemble est défini par induction de la manière suivante :

- $R^{(0)}$ est le système initial à résoudre.
- $R^{(i+1)} = \sigma\left(R^{(i)}\right)$.

$R^{(i+1)}$ est l'ensemble des relations obtenu en appliquant les règles de σ à l'ensemble des relations $R^{(i)}$ .

## Définition 4.7 *(ordre d'un habitant)*

Un habitant est dit d'ordre i, s'il est défini par un élément de $R^{(i)}$ de la forme :

$$(u = Xv)\,\mathrm{X}$$

Où X n'apparaît pas dans u et v.

Les habitants d'ordre i sont notés par $X^{(i)}$.

## Définition 4.8 (l'univers HABIT associée à un type)

L'univers *HABIT (T, V, σ, R)* associe à un type T est défini de la manière suivante :

i. L'ensemble des variables V est formé par l'ensemble des habitants assignés aux sous formules de T.
ii. L'ensemble des substitutions σ est défini à partir du sous ensemble $R^{(0)}$ des relations R [voir la définition 4.6.]

iii. L'ensemble des objets X est formé par l'ensemble des habitants du type T. avec $X = \bigcup_{i=0}^{\infty} X^{(i)}$ et $X^{(i)}$ représente l'ensemble des habitants de T d'ordre i.

## Propriété 4.3

Soit T un type. S'il existe un entier $n_0$ tel que $R^{(n0)}$ est vide, alors :

i. $\forall i \geq n_{0,} X^{(i)} = \Phi$

ii. l'ensemble $X$ des habitants de T est fini.

**Preuve (i):**

Pour démontrer la propriété i, il faut démontrer que $\forall i \geq n_{0,} R^{(i)} = \Phi$ Par induction sur i :

*Base de récurrence* $i = n_0$ *:* l'ensemble des relation d'ordre $n_0$ est vide (c'est une hypothèse de la propriété).

*Hypothèse de récurrence :* on suppose que $R^{(i)}{}_{n0 \leq i \leq k} = \Phi$ et on démontre que $R^{(k+1)} = \Phi$ :

Par définition $R^{(k+1)} = \sigma\left(R^{(k)}\right)$ donc $R^{(k+1)} = \sigma\left(R^{(k)}\right) = \Phi$

**Preuve (ii):**

Par définition on a :

$$X = \bigcup_{i=0}^{\infty} X^{(i)} = \bigcup_{i=0}^{n0} X^{(i)} \cup \bigcup_{i=n0}^{\infty} X^{(i)},$$

On peut démontrer que $\bigcup_{i=n0}^{\infty} X^{(i)} = \Phi$

Et par la suite démontrer, facilement que l'ensemble $\bigcup_{i=0}^{n0} X^{(i)}$ est fini.

## 4.2 Exemples :

**Exemple 01 :**

Considérons le type T $=(a \to a) \to (a \to a)$

$$TX \equiv (a \to a) \to (a \to a)X$$
$$((a \to a)u \supset (a \to a)(Xu))X$$
$$((av \supset a(uv))u \supset (aw \to a(Xuw)(Xu))X$$

L'ensemble des sous formules avec leurs habitants est le suivant :

$$\begin{cases} (a \to a) \to (a \to a) \ \{X\} \\ (a \to a) \ \{u, (Xu)\} \\ a \quad \{v, \ w, (uv), (Xuw)\} \end{cases} \Rightarrow AtomicFormula = \{a\}$$

L'ensemble des variables est $V = \{u, v, w\} \approx \{u, v\}_{u=w}$

L'ensemble des relations $R^{(0)} = \{v = uv, \quad v = Xuv, \quad uv = Xuv\}$

L'ensemble σ est formé par les substitutions suivantes :

$$\sigma = \begin{cases} \sigma_1 : v \leftarrow uv \\ \sigma_2 : v \leftarrow Xuv \\ \sigma_3 : uv \leftarrow Xuv \\ \sigma_4 : uv \leftarrow v \\ \sigma_5 : Xuv \leftarrow v \\ \sigma_6 : Xuv \leftarrow uv \end{cases}$$

L'ensemble X qui représente les habitants de T est défini par :

- $X^{(0)}$ défini à partir de $R^{(0)}$ est : $X^{(0)} = \{ \ \lambda uv.v \ , \lambda uv.uv\}$
- Par la suite, pour définir les habitants d'ordre i >0, il faut définir les relations d'ordre i.

$$R^{(1)} = \sigma\left\{\begin{array}{l} v = uv \\ v = Xuv \\ uv = Xuv \end{array}\right\} = \left\{\begin{array}{l} \xrightarrow{\sigma_1}\left\{\begin{array}{l} uv = uv, v = uuv, \\ uv = Xuv, \\ (u(uv) = Xuv)^* \end{array}\right. \\ \xrightarrow{\sigma_2}\left\{\begin{array}{l} (Xuv = uv)^*, v = uXuv, \\ Xuv = Xuv, \\ uXuv = Xuv \end{array}\right. \\ \xrightarrow{\sigma_3}\left\{\begin{array}{l} uv = uv, v = uuv, \\ (uv = Xuv)^{**} \\ (uuv = Xuv)^{**} \end{array}\right. \\ \xrightarrow{\sigma_4}\{\ v = v, (v = Xuv)^* \\ \xrightarrow{\sigma_5}\{\ v = v, uv = v \\ \xrightarrow{\sigma_6}\{\ v = uv, uv = uv \end{array}\right.$$

$X^{(1)}$ obtenu en utilisant une seul fois les substitutions de σ est :

$$X^{(1)} = \{\ \lambda uv.\, u(uv),\ \lambda uv.v\ ,\ \lambda uv.uv\ \}$$

**Exemple 02 :**

Considérons le type T $= ((a \to a) \to a) \to a$

$$\begin{array}{l} TX \equiv (((a \to a) \to a) \to a)X \\ \quad ((a \to a) \to a)u \supset a(Xu) \\ \quad ((a \to a)v \supset a(uv))u \to a(Xu) \\ \quad ((aw \to a(vw))v \supset a(uv))u \to a(Xu) \end{array}$$

L'ensemble des sous formules atomiques est :

$$AtomicFormula = \{a\}$$

L'ensemble des variables est $V = \{u, v, w\}$

L'ensemble des relations $R^{(0)} = \left\{\begin{array}{rrr} w = vw, & w = uv, & w = Xu, \\ & vw = uv, & vw = Xu \\ & & uv = Xu \end{array}\right\}$

L'ensemble σ est formé par les substitutions suivantes :

$$\sigma = \begin{cases} \sigma_1 : w \leftarrow uv \\ \sigma_2 : w \leftarrow Xu \\ \sigma_3 : vw \leftarrow uv \qquad \sigma_4 : uv \leftarrow uw \\ \sigma_5 : vw \leftarrow Xu \\ \sigma_6 : uv \leftarrow Xu \qquad \sigma_7 : uv \leftarrow Xu \end{cases}$$

L'ensemble X qui représente les habitants de T est défini par :
$X^{(0)}$ défini à partir de $R^{(0)}$ est :

$$X^{(0)} = \{ \ \lambda u.w \,,\ \lambda u.vw \,,\ \lambda u.uv \ \}$$

$$R^{(1)} = \sigma \begin{Bmatrix} w = vw, & w = uv \ , & w = Xu, \\ & vw = uv, & vw = Xu \\ & & uv = Xu \end{Bmatrix} = \begin{cases} \xrightarrow{\sigma_1} \begin{cases} uv = vw, uv = v(uv) \ , w = v(uv) \\ uv = Xu, \\ v(uv) = Xu \end{cases} \\ \xrightarrow{\sigma 2} \begin{cases} Xu = vw, uv = v(Xu) \ , w = vXu \\ v(Xu) = uv, \\ v(Xu) = Xu \end{cases} \\ \xrightarrow{\sigma_2} \{ \ w = uv, uv = Xu \\ \xrightarrow{\sigma 4} \{ \ w = uw, uw = Xu \\ \xrightarrow{\sigma_3} \begin{cases} uv = uv, v = uuv, \\ (uv = Xuv) \\ (uuv = Xuv) \end{cases} \\ \xrightarrow{\sigma_4} \{ \ v = v, (v = Xuv) \\ \xrightarrow{\sigma_5} \{ \ v = v, uv = v \\ \xrightarrow{\sigma_6} \{ \ v = uv, uv = uv \end{cases}$$

$X^{(1)}$ obtenu en utilisant une seul fois les substitutions de σ.

$$X^{(1)} = \{ \ \lambda u.uv, \lambda u.v(uv), \ \lambda u.uw \quad \cdots \ \}$$

**Exemple 03 :**

Pour vérifier la validité de la formule :

$$A \rightarrow \neg\neg A$$

La première étape consiste à exprimer cette formule en fonction des constantes du E_lambda calcul. Cette transformation nous permet d'obtenir la formule suivante :

$$A \rightarrow \neg\neg A \equiv \left(A \rightarrow \left((A \rightarrow \perp) \rightarrow \perp\right)\right)$$

Et par la suite, pour vérifier la validité de cette formule, on commence par l'attribution des termes à l'entité manipulée. Les étapes d'exécution de ce processus sont décrites par le fragment suivant:

$$\begin{aligned}
&\left(A \rightarrow \left((A \rightarrow \perp) \rightarrow \perp\right)\right) X \\
&A : u_1 \supset \left((A \rightarrow \perp) \rightarrow \perp\right)(X u_1) \\
&A : u_1 \supset (A \rightarrow \perp) u_2 \supset \perp (X u_1 u_2) \\
&A : u_1, (A \rightarrow \perp) u_2 \succ \perp (X u_1 u_2) \\
&A : u_1, A : u_3 \supset \perp : u_2 u_3 \succ \perp : (X u_1 u_2)
\end{aligned}$$

L'ensemble des formules atomiques avec leurs habitants est le suivant :

$$\Rightarrow AtomicFormula = \{A(u_1, u_3), \perp (u_2 u_3, X u_1 u_2)\}$$

L'ensemble des variables est :

$$V = \{u_1 \approx u_3, u_2\}_{u=w}$$

L'ensemble des relations d'ordre zéro est :

$$R^{(0)} = \{\ u_2 u_3 = X u_1 u_2\}_{u1=u3} = \{\ u_2 u_1 = X u_1 u_2\}$$

L'ensemble σ est formé par les substitutions suivantes :

$$\sigma = \begin{cases} \sigma_1 : u_2 u_1 \leftarrow X u_1 u_2 \\ \sigma_2 : X u_1 u_2 \leftarrow u_2 u_1 \end{cases}$$

L'ensemble X qui représente les habitants de T est défini par :

- $X^{(0)}$ défini à partir de $R^{(0)}$ est : $X^{(0)} = \{\ \lambda u_1 u_2 . u_2 u_1\}$
- $R^{(1)} = \sigma\left(R^{(0)}\right) = \Phi$ donc l'ensemble des habitants $X^{(1)}$ est vide.

**Exemple 04**

Soit le type suivant :

$$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$$

Les deux états, initial et finale du processus d'attribution des preuves sont les suivantes :

$$|-(A \to (B \to C)) \to ((A \to B) \to (A \to C))X$$

$$|-(Au_2 \supset (Bu_3 \supset Cu_1u_2u_3)) \supset ((Au_5 \supset Bu_4u_5) \supset (Au_6 \supset C(Xu_1u_4u_6)))$$

L'application de la première règle du calcul des séquents permet de définir le but suivant :

$$(Au_2 \supset (Bu_3 \supset Cu_1u_2u_3)), (Au_5 \supset Bu_4u_5), Au_6 \ |-C(Xu_1u_4u_6)$$

L'application de la deuxième règle du calcul des séquents permet de définir l'ensemble des buts élémentaires.

$$\begin{cases} (Au_5 \supset Bu_4u_5), Au_6 \ |-Au_2 \Rightarrow \begin{cases} Au_6 \ |-Au_5 & \cdots\cdots(but_1) \\ (Bu_4u_5), Au_6 \ |-Au_2 & \cdots\cdots(but_2) \end{cases} \\ (Bu_3 \supset Cu_1u_2u_3), (Au_5 \supset Bu_4u_5), Au_6 \ |-C(Xu_1u_4u_6) \quad \cdots(*) \end{cases}$$

$$(*) \Rightarrow \begin{cases} (Au_5 \supset Bu_4u_5), Au_6 \ |-Bu_3 \Rightarrow \begin{cases} (Bu_4u_5), Au_6 \ |-Au_5 & \cdots\cdots(but_3) \\ (Bu_4u_5), Au_6 \ |-Bu_3 & \cdots\cdots(but_4) \end{cases} \\ Cu_1u_2u_3, Au_5 \supset Bu_4u_5, Au_6 \ |-C(Xu_1u_4u_6) \Rightarrow \begin{cases} Cu_1u_2u_3, \ Au_6 \ |-Au_5 & \cdots\cdots(but_5) \\ Cu_1u_2u_3, Bu_4u_5, Au_6 \ |-C(Xu_1u_4u_6) \cdots (but_6) \end{cases} \end{cases}$$

L'ensemble des formules atomiques avec leurs habitants est le suivant :

$$\Rightarrow AtomicFormula = \{A(u_2 = u_5 = u_6, u_6), B(u_3, u_4u_2), C(u_1u_2u_3, Xu_1u_4u_2)\}$$

L'ensemble des variables est $V = \{u_1, u_2 \approx u_5 \approx u_6, u_3, u_4\}$

L'ensemble des relations $R^{(0)} = \{ (u_3 = u_4 u_2), (u_1 u_2 u_3 = X u_1 u_4 u_2) \}$

L'ensemble σ est formé par les substitutions suivantes :

$$\sigma = \begin{cases} \sigma_1 : u_3 \leftarrow u_4 u_2 \\ \sigma_2 : u_1 u_2 u_3 \leftarrow X u_1 u_4 u_2 \\ \sigma_4 : X u_1 u_4 u_2 \leftarrow u_1 u_2 u_3 \end{cases}$$

L'ensemble X qui représente les habitants de T est défini par :

$X^{(0)}$ défini à partir de $R^{(0)}$ est : $X^{(0)} = \{ \lambda X u_1 u_4 u_2 . u_1 u_2 u_3 \}$

L'ensemble des relations d'ordre 1 est :

$$R^{(1)} = \sigma\left(R^{(0)}\right) = \{ (u_1 u_2 (u_4 u_2) = X u_1 u_4 u_2) \}$$

Donc l'ensemble des habitants d'ordre 1 est :

$$X^{(1)} = \{ \lambda X u_1 u_4 u_2 . u_1 u_2 (u_4 u_2) \}.$$

L'ensemble des relations $R^{(2)}$ est :

- $R^{(2)} = \sigma\left(R^{(1)}\right) = \Phi$ donc l'ensemble des habitants $X^{(2)}$ est vide.

**Exemple 05**

Considérons l'entité quantifiée suivante :

$$\forall P \left( P_0 \rightarrow \left( \left( \forall n \in \mathrm{N} \left( P_n \rightarrow P_{n+1} \right) \rightarrow \forall n \in \mathrm{N} P_n \right) \right) \right)$$

Le déroulement pas à pas du processus d'attribution des termes est décrit par les étapes suivantes :

$$\forall P \left( P_0 \rightarrow \left( \left( \forall n \in \mathrm{N} \left( P_n \rightarrow P_{n+1} \right) \rightarrow \forall n \in \mathrm{N} P_n \right) \right) \right) : X$$

$$\left( P_0 \rightarrow \left( \left( G \mathrm{N} \left( \lambda n . P_n \rightarrow P_{n+1} \right) \rightarrow G \mathrm{N} \left( \lambda n . P_n \right) \right) \right) \right) : X$$

$$P_0 : u_1 \supset \left( \left( G \mathrm{N} \left( \lambda n . P_n \rightarrow P_{n+1} \right) \rightarrow G \mathrm{N} \left( \lambda n . P_n \right) \right) \right) : (X u_1)$$

$$P_0 : u_1 \supset G \mathrm{N} \left( \lambda n . P_n \rightarrow P_{n+1} \right) : u_2 \supset G \mathrm{N} \left( \lambda n . P_n \right) : (X u_1 u_2)$$

La construction du but principal et des buts élémentaires est détaillée par le fragment suivant :

$$P_0 u_1, GN(\lambda n.P_n \rightarrow P_{n+1})u_2 \ \vert- GN(\lambda n.P_n)(Xu_1 u_2)$$

$$P_0 u_1, Nt \supset (\lambda n.P_n \rightarrow P_{n+1})t(u_2 t) \ \vert- Nt \supset (\lambda n.P_n)t(Xu_1 u_2 t)$$

$$Nt, P_0 u_1, (\lambda n.P_n \rightarrow P_{n+1})t(u_2 t) \ \vert- (\lambda n.P_n)t(Xu_1 u_2 t)$$

$$Nt, P_0 u_1, (P_t \rightarrow P_{t+1})(u_2 t) \ \vert- (P_t)(Xu_1 u_2 t)$$

$$Nt, P_0 u_1, (P_t u_3 \supset P_{t+1})(u_2 t u_3) \ \vert- (P_t)(Xu_1 u_2 t)$$

$$\Rightarrow \begin{cases} Nt, P_0 u_1 \ \vert- P_t u_3 \\ Nt, P_0 u_1, (P_{t+1})(u_2 t u_3) \ \vert- (P_t)(Xu_1 u_2 t) \end{cases}$$

$$\Rightarrow R^{(0)} = \begin{cases} (u_1 = Xu_1 u_2 t)_{t=0} \\ (u_3 = Xu_1 u_2 t) \\ (u_2(t)u_3 = Xu_1 u_2(t+1)) \end{cases}$$

Intuitivement, on peut définir les habitants de la proposition en question par la définition inductive suivante :

$$\Rightarrow \begin{cases} Xu_1 u_2 0 = u_1 \\ Xu_1 u_2(t+1) = u_2 t X u_1 u_2 t \end{cases}$$

# Conclusion

Le travail réalisé est inscrit autour de la richesse et la consistance du Eλ-Calcul. Il consiste à définir et implémenter un système de preuve d'ordre supérieur basé sur ce calcul.

Ce travail nous a permis de nous familiariser avec la théorie des démonstrateurs automatiques et de découvrir le principe de base et les particularités des systèmes de preuve les plus connus (Coq, Phox, Agda/Alfa, Isabelle/HOL, PVS), ainsi que les étapes et les grandes lignes à suivre pour implémenter un système de preuve, à travers l'exploration du démonstrateur pédagogique FOLDROL.

L'isomorphisme de Curry Howard est le principe de base de la majorité des systèmes de preuve existants, dont la définition est inspirée de la relation entre les termes et leurs types, ce qui nécessite la manipulation de deux objets différents. Contrairement à ce principe, tous les objets manipulés dans le E-lambda calcul sont des termes.

Notre procédure de preuve consiste à définir l'univers HABIT associé à la proposition en question. Cette procédure est capable de définir l'ensemble des preuves d'une proposition quand ce dernier est fini. Dans l'autre cas, elle permet de définir l'ensemble des preuves dont l'ordre est fixé en paramètre.

La robustesse de la manipulation des arbres, notamment pour implémenter les traitements récurrents, nous a imposé d'utiliser la structure d'arbres binaires pour implémenter notre procédure. Avec cette représentation les feuilles de l'arbre représentent les atomes de la proposition en question et les nœuds internes représentent les connecteurs (qui sont exprimés

en fonction des implications) de cette dernière. Les algorithmes utilisés sont définis dans l'annexe.

Comme perspective de recherche et concernant les travaux futurs nous proposons les axes de recherches suivants :

- ✓ Améliorer et certifier la stratégie proposée en étudiant et en prouvant les propriétés des différents paramètres de l'univers HABIT ; Notamment la réduction des ensembles des relations R et des substitutions σ.
- ✓ Etudier, avec plus de détail, l'utilisation de la skolémisation pour traiter les entités quantifiées. Cette étude peut être vue comme introduction pour étudier et mettre en lumière une interface Eλ-SAT3, qui servira comme une nouvelle stratégie pour générer des solutions pour les problèmes NP-Complet.

# Bibliographie


[Bar 84] H.P. Barendregt, *« The Lambda Calculus: Its Syntax and Semantics »*. Studies in Logic and the Foundations of Mathematics 103. North Holland, 1984. (p. 2)

[Cat 95] P. Catherine, *«Synthèse de Preuves de Programmes dans le Calcul des Construction Inductives»,* thèse de doctorat, l'ENS, Lyon, 1995.

[Chu 33] A. Church, *« A set of postulates for the fondation of logics »* Annals of mathematics, vol. 33 and 34 1932/33, p 346-366 and 839-864.

[Coq 91] T. Coquand, «An introduction to Type Theory », INRIA, 1991.

[DL 03] D. DOUGHERTY, P. LESCANNE, *« Reductions, Intersection Types, and Explicit Substitutions»*, Mathematical Structures in Computer Science 13, 1, 2003.

[Dow 91] G. Dowek, *« Demonstration Automatique dans le Calcul des Construction»,* Thèse de PHD, Université de Paris 7, 1991.

[Dub 01] C. Dubois. *« Les programmes vus comme des preuves,les types vus comme des propositions »*. Notes de cours DEA Informatique DRAFT, octobre 2001.

[Gal 86] Jean H. Gallier. « *Logic for Computer Science : Foundations of Automatic Theorem Proving*». Harper & Row.

[Hal 01] Thomas Hallgren. *Alfa*, 2001. . `http://www.cs.chalmers.se/~hallgren/Alfa/`.

[How 80] W.A. Howard, *« the formulae-as-types notion of construction »*. In Hindley and seldin, editor, To H.B Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pages 479-490.Academic Press,London, 1980.

[Hue 75] Huuet G. *«* A *unification algorithm for typed lambda calculus »*. Theorical computer Science , vol. 1, 1975 pp27-57

[Hue 86] Gérard Huet. *«Formal structures for computation and deduction »*. CourseNotes, Carnegie-Mellon University, First Edition, May 1986.

[KM 97] Matt Kaufmann and J Strother Moore. *« A Precise Description of the ACL2 Logic ».* Technical report, Computational Logic, Inc., 1997.

[Kri 90] J.L. Krivine, *«Lambda-calcul, types et modèles»,* MASSON, 1990.

[Lak 76] Imre Lakatos. *«Proofs and Refutations: The logic of Mathematical Discovery».* Cambridge University Press.

[Lal 90] R. Lalement, *«Logique, Réduction, Résolution»,* MASSON, 1990.

[Law 92] Lawrence C. Paulson. *« Designing a Theorem Prover ».* Handbook of Logic in Computer Science, Vol II (Oxford, 1992), 415-475

[MC 02] Med. MEZGHICHE, Choukri BEN YELLES, *«E_Lambda calculus: An illative combinatory logic interpreting higher order logic»* Workshop on 35 years of Automath Heriot-Watt University, Edinburgh, April 2002.

[MS 05] Manfred Schmidt-Schauß and J¨org H.Siekmann, *«Unification Algebras:An Axiomatic Approach toUnification, Equation Solving and Constraint Solving»,* Institut für Informatik,Germany, 2005

[OSRS 01] S. Owre, N. Shankar , J. M. Rushby, and D. W. J. Stringer-Calvert. *« PVS System Guide ».* SRI International, `http://pvs.csl.sri.com`, v2.4 edition, 2001.

[Pel 86] F. J. Pelletier, *«Seventy-five problems for testing automatictheorem provers»,* Journal of Automated Reasoning, 1988.

[Rob 65] Robinson J.A. *« A machine oriented logic based on the resolution principle ».* JACM, vol.12(1) 1965 pp23-41

[RP 90] J. C. Reynolds and G. D. Plotkin, *« On Functors Expressible in the polymorphic typed lambda calculus »* , in G. Huet, editor, Logical Foundations of Functional Programming, pp. 127–152, Addison-Wesley, 1990

[Sel 97] J. P. Seldin, *«On the proof theory of Coquand's calculus of constructions».* Annals of Pure and Applied Logic 83 (1997) 23–101.

[Tak 87] G. Takeuti. *«Proof Theory».* North Holland, 2nd edition.

[Wer 94] Benjamin Werner. *« Une Théorie des Constructions Inductives ».* PhD Thesis, Université Paris VII, mai 1994.

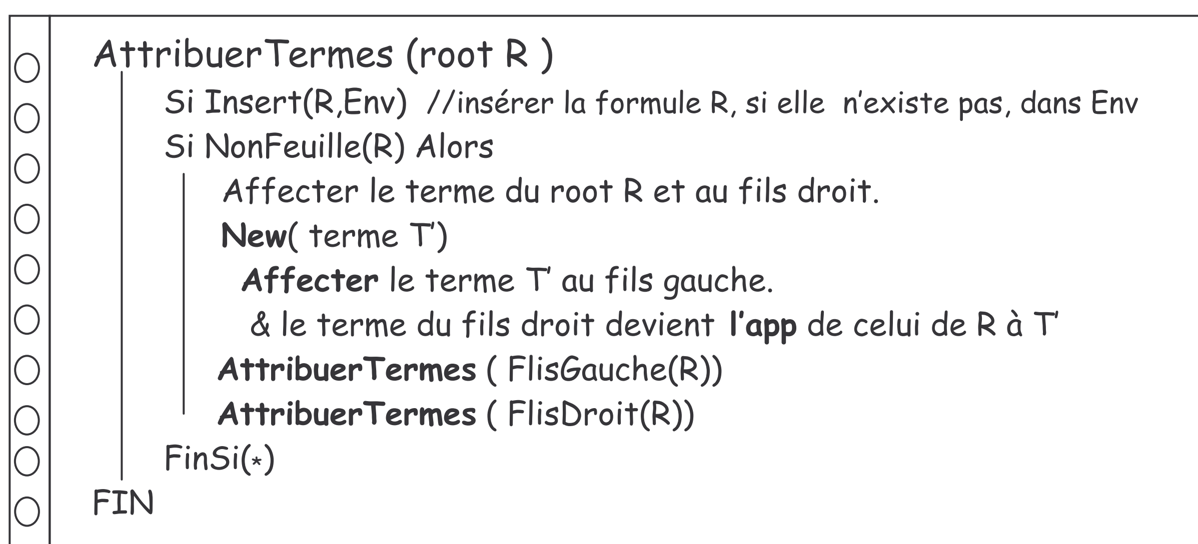
# Annexe

## 1. Algorithme d'assignation

Le rôle de cet algorithme est d'assigner à chaque sous formule un habitant. Si une sous formule apparaît plusieurs fois, elle aura autant d'habitants que leur nombre d'apparition.

Nous utilisons la structure d'arbre binaire pour représenter les entités manipulées. Avec cette représentation les feuilles représentent les atomes (une formule atomique de la logique des prédicats) et les nœuds internes représentent les connecteurs (pour notre cas il y a que des implications).

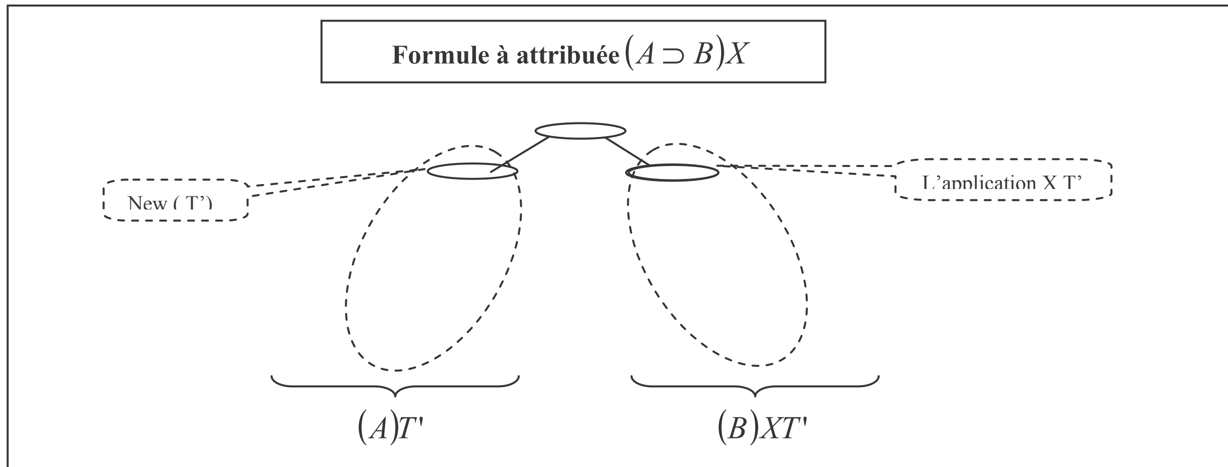
La première étape de ce processus consiste à affecter le terme à déterminer X à la racine de l'arbre qui représente la formule en question. Et par la suite, pour définir les preuves des sous formules, nous proposons l'algorithme suivant :



```
AttribuerTermes (root R )
    Si Insert(R,Env)  //insérer la formule R, si elle  n'existe pas, dans Env
    Si NonFeuille(R) Alors
        Affecter le terme du root R et au fils droit.
        New( terme T')
          Affecter le terme T' au fils gauche.
           & le terme du fils droit devient l'app de celui de R à T'
        AttribuerTermes ( FlisGauche(R))
        AttribuerTermes ( FlisDroit(R))
    FinSi(*)
FIN
```

**Algo 1. Attribution des preuves**

L'exécution d'une itération de l'algorithme précédent est schématisée par la figure suivante :

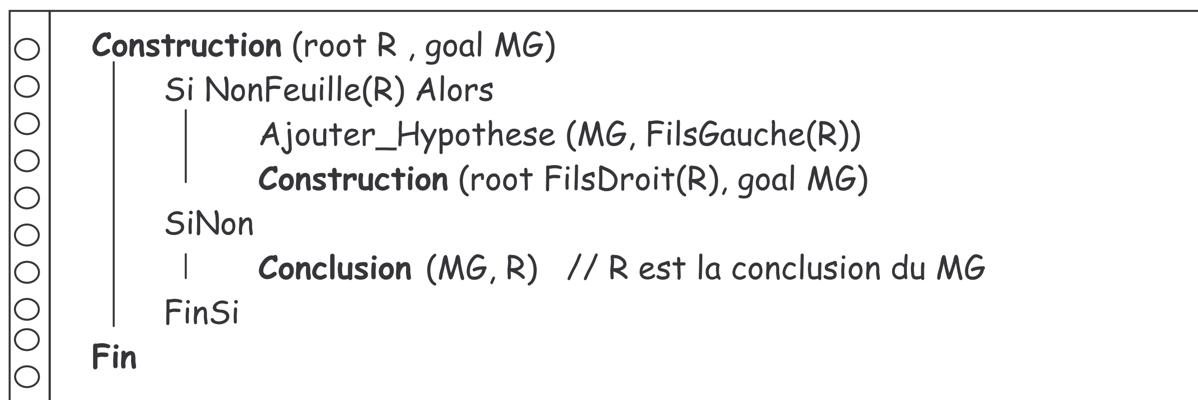



*Une itération d'attribution des preuves*

Le déclenchement du processus d'attribution des termes permet d'attribuer un habitant pour chaque sous formule de la formule en question.

## 2. Algorithme d'assignation

L'algorithme permettant construction du but principal est défini par induction sur la structure du résultat du processus précèdent, de la manière suivante:

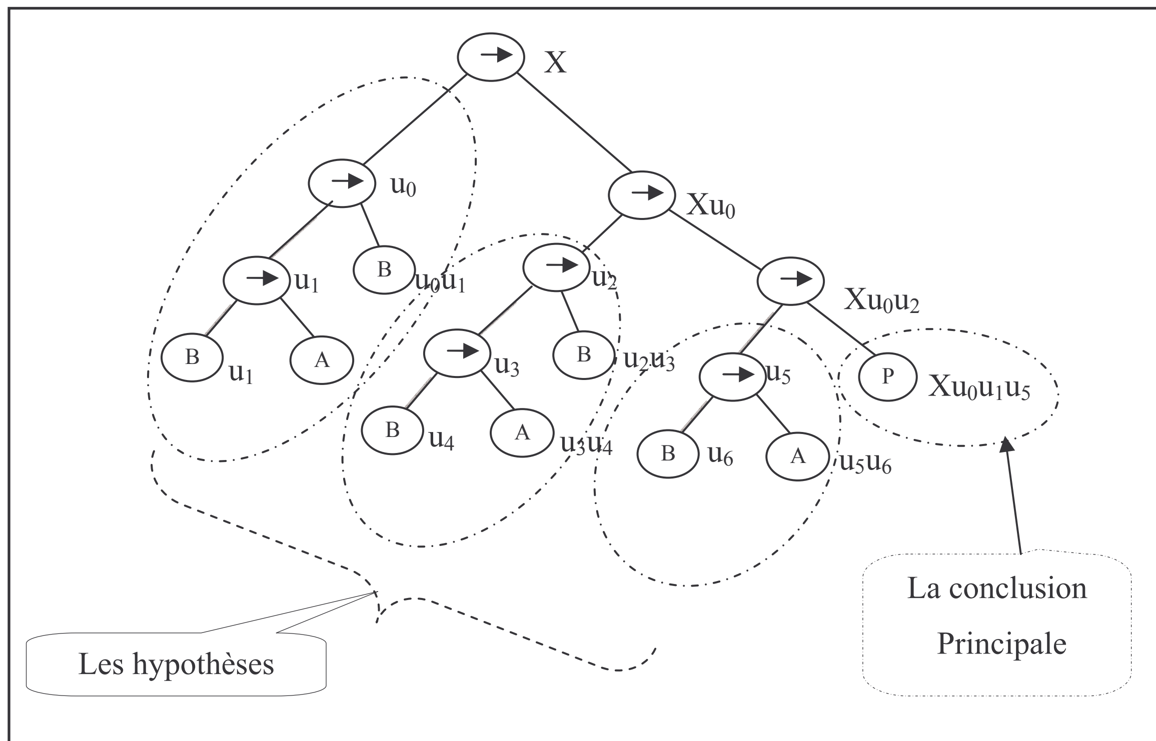


```
Construction (root R , goal MG)
    Si NonFeuille(R) Alors
        Ajouter_Hypothese (MG, FilsGauche(R))
        Construction (root FilsDroit(R), goal MG)
    SiNon
        Conclusion (MG, R)   // R est la conclusion du MG
    FinSi
Fin
```

Le principe de de fonction de cet algorithme peut être schématisé par la figure suivante :

**Algo 2. Construction du but principal**

Le fonctionnement de l'algorithme précédent est schématisé par la figure suivante :




*Construction du But Principal*

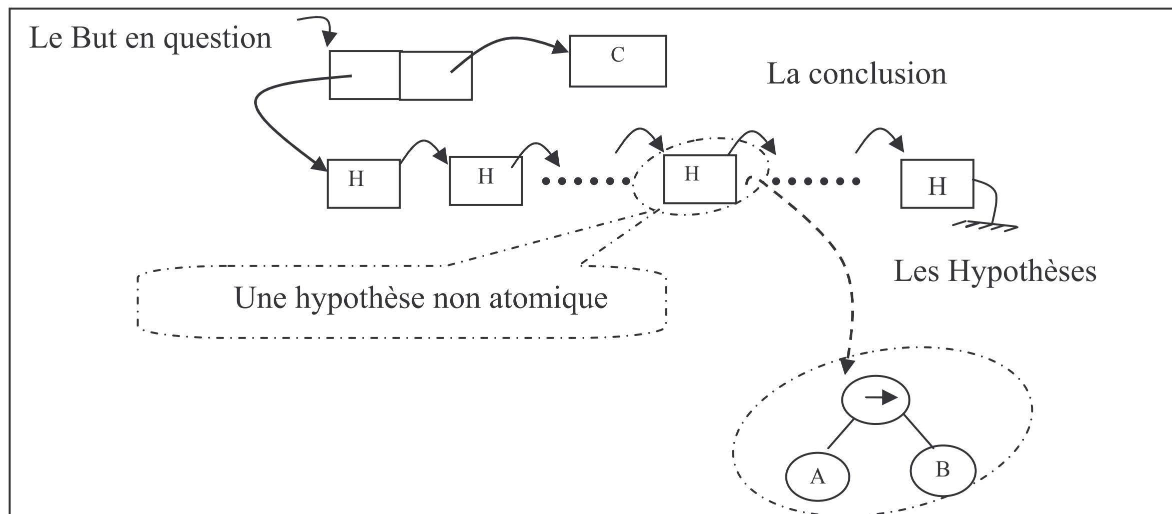
## 3. L'algorithme SubGoals

L'ensemble des sous buts est construit initialement à partir du but principal, en se basant sur la structure des formules de l'ensemble d'hypothèses. Le cas trivial de ce processus est le cas où toutes les hypothèses sont atomiques.

Dans l'autre cas, où il y a au moins une formule non-atomique $F_i$ dans l'ensemble d'hypothèses, ( $F_i = A \supset B$ )

L'éclatement de ce but nous permet la construction de deux sous buts. Le principe de base de cette construction est le suivant :

1) Un nouveau but est construit, dans le quel :
    - l'ensemble des hypothèses est le même que celui du but éclaté (en question) en retirant l'hypothèse non atomique
    - et sa conclusion correspond à la partie gauche de l'hypothèse en question.
2) Le deuxième but est exactement celui en question, à l'exception du changement de l'hypothèse non atomique par sa partie droite.

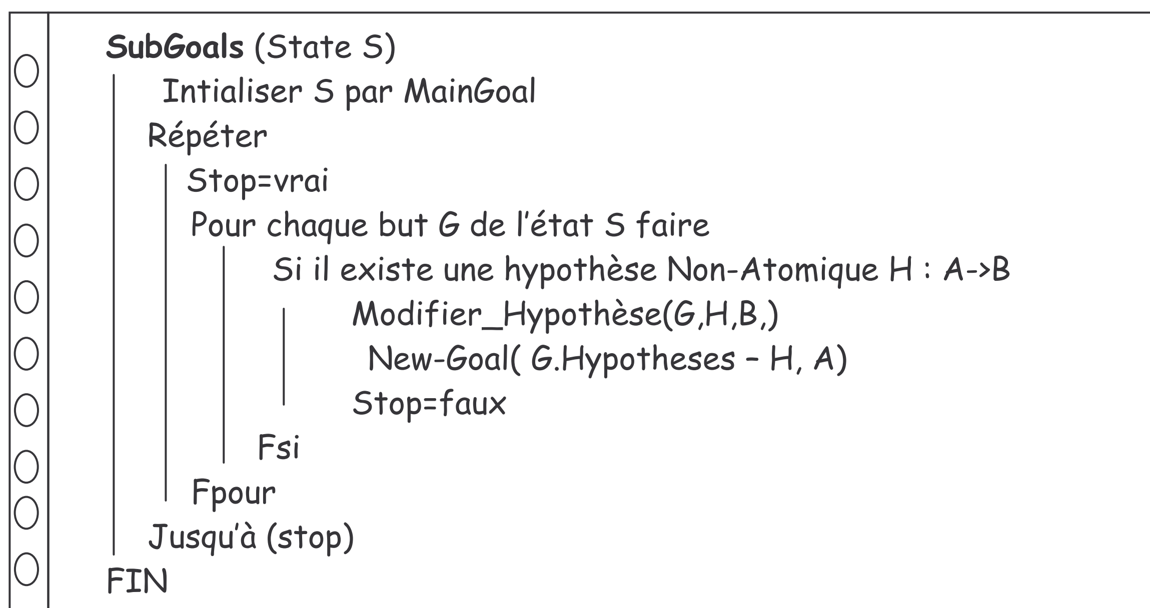



*Un but non élémentaire.*

L'algorithme chargé à la construction des sous buts est le suivant :

Ou La fonction *Non_atomique* retourne faux si l'argument de cette fonction est une formule atomique et vraie sinon et S contient l'ensemble des buts à un instant donné

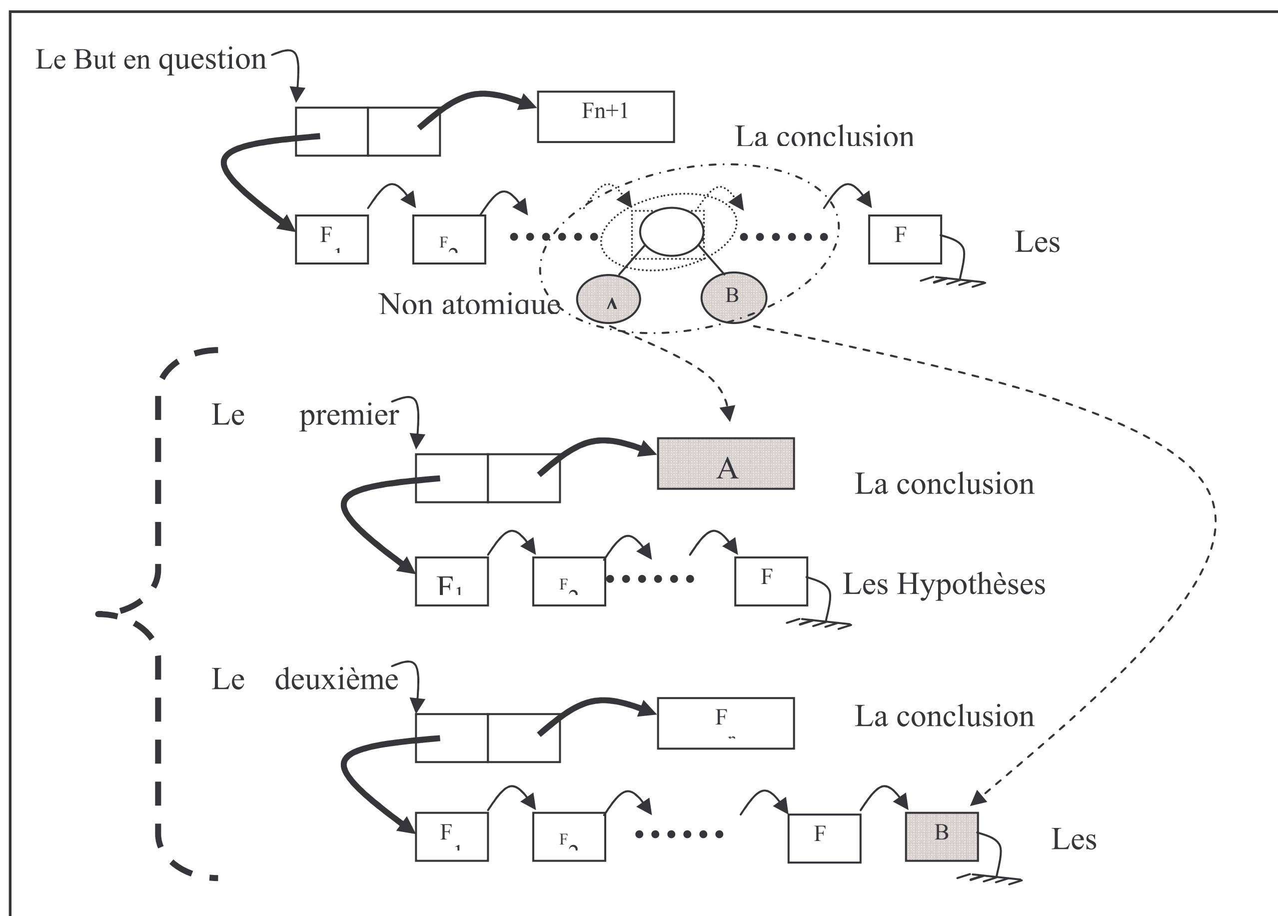
## 4. Construction des buts élémentaires



```
SubGoals (State S)
    Intialiser S par MainGoal
    Répéter
        Stop=vrai
        Pour chaque but G de l'état S faire
            Si il existe une hypothèse Non-Atomique H : A->B
                Modifier_Hypothèse(G,H,B,)
                 New-Goal( G.Hypotheses – H, A)
                Stop=faux
            Fsi
        Fpour
    Jusqu'à (stop)
FIN
```

**Algo 3. Construction de construction des sous buts**

Une itération de cet algorithme est schématisée par la figure suivante :



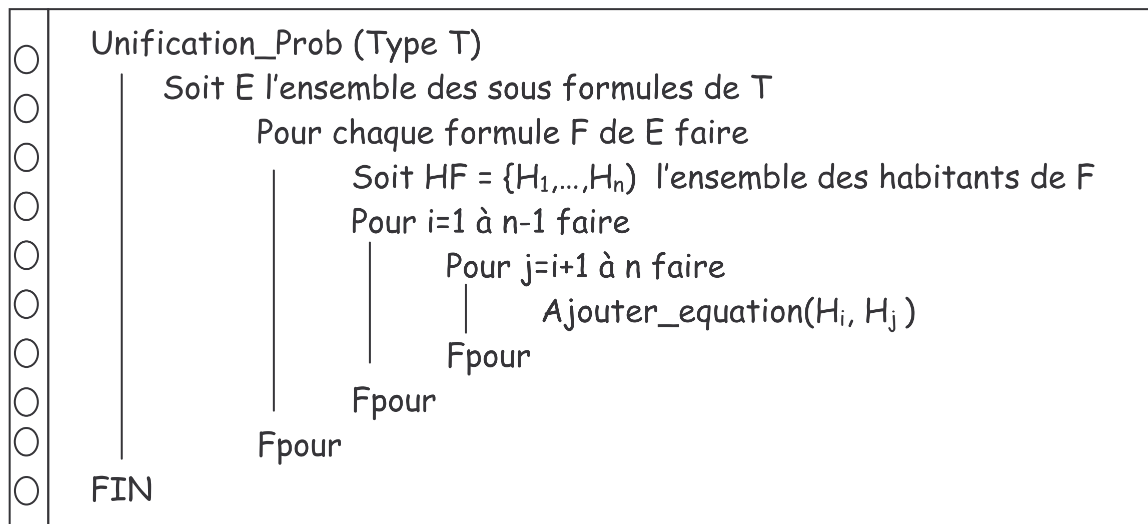

*Construction des sous buts (SubGoals)*

L'exécution de ce processus sur l'ensemble d'hypothèses, nous permet séquentiellement, de passer au traitement des sous buts d'un but, et parallèlement pour le traitement des buts du comportement du système à un instant donné.

Ce processus converge évidemment vers un état final, où tous les buts sont élémentaires (toutes les hypothèses sont des formules atomiques).

## 5. Construction du problème d'unification

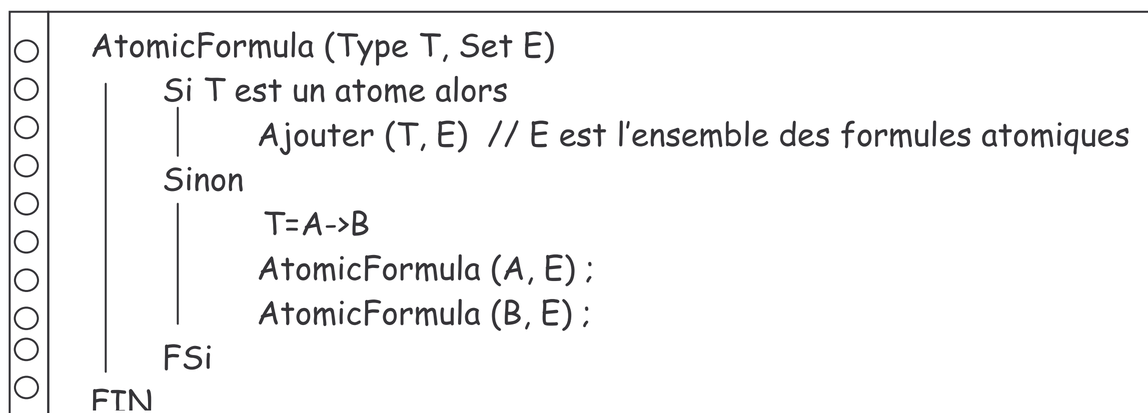
Nous proposons l'algorithme suivant pour construire le problème d'unification, dont la résolution permet de définir les habitants d'un type T.



```
Unification_Prob (Type T)
    Soit E l'ensemble des sous formules de T
        Pour chaque formule F de E faire
            Soit HF = {H1,…,Hn)  l'ensemble des habitants de F
            Pour i=1 à n-1 faire
                Pour j=i+1 à n faire
                    Ajouter_equation(Hi, Hj )
                Fpour
            Fpour
        Fpour
FIN
```

**Algo 4. Construction de système d'équations ($R^{(0)}$)**

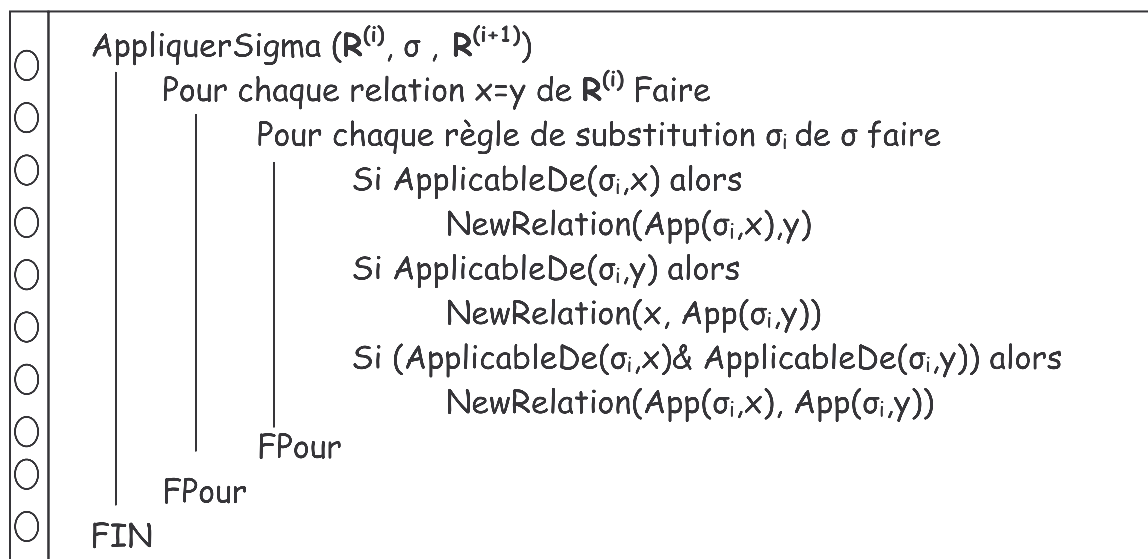
## 6. Construction de l'ensemble des formules atomiques



```
AtomicFormula (Type T, Set E)
    Si T est un atome alors
        Ajouter (T, E)  // E est l'ensemble des formules atomiques
    Sinon
        T=A->B
        AtomicFormula (A, E) ;
        AtomicFormula (B, E) ;
    FSi
FIN
```

**Algo 5. Construction de l'ensemble des formules atomiques**

## 7. Définition de l'ensemble $R^{(i+1)}$



```
AppliquerSigma (R(i), σ , R(i+1))
    Pour chaque relation x=y de R(i) Faire
        Pour chaque règle de substitution σi de σ faire
            Si ApplicableDe(σi,x) alors
                NewRelation(App(σi,x),y)
            Si ApplicableDe(σi,y) alors
                NewRelation(x, App(σi,y))
            Si (ApplicableDe(σi,x)& ApplicableDe(σi,y)) alors
                NewRelation(App(σi,x), App(σi,y))
        FPour
    FPour
FIN
```

**Algo 6. Application des substitutions Sigma**