République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed BOUGARA de BOUMERDES
Faculté des sciences
Département d'Informatique

MEMOIRE DE MAGISTER

Spécialité :Système informatique et génie des logiciels

Option :Spécification logiciel et traitement de l'information
Ecole Doctorale

Présenté par :
SI SMAIL Ouarda

Thème

Génération des habitants d'un type simple dans le lambda-calcul

Devant le jury de soutenance composé de :

M.Ahmed nacer Mohamed	Président
Mme Tebibel Thouraya	Examinatrice
M.Ait Bouziad Ahmed	Examinateur
M.Mezghiche Mohamed	Encadreur

Table des matières

Introduction **5**

1 Rappels sur le λ-Calcul et le système d'assignation **6**
1 Les λ-termes et leur structures 6
1.1 Variable liées, termes fermés, substitution : 7
1.2 Réductions et formes normales 8
1.3 Les λ-termes restreints 9
2 Assignation des types aux termes 10
2.1 Le système d'assignation de types dans λ-calcul (TA_λ) 10
2.2 Substitution des types 12

2 Génération des habitants d'un type **15**
1 Algorithme de Ben-yelles et Hindely 15
1.1 Algorithme de recherche 18
1.2 Algorithme de comptage 20
1.3 Algorithme de comptage et d'énumération des Long (α) 21
2 Algorithme de Broda et Damas 22
2.1 Algorithme arbre-formule 22
2.2 Algorithme terme 25
2.3 Grammaire à contexte libre G_τ 26
2.4 Le schéma général d'une grammaire pour une structure de type donné 28
3 La technique de plus petit point fixe (fixpoint) pour le comptage des termes dans le lambda calcul 30
3.1 Algorithme de plus petit point fixe 32
3.2 Le nombre de termes fermés 32
3.3 Algorithme de construction polynomiale 34
4 Grammaire de génération des habitants d'un type (Takahashi, Akama) 38
4.1 Grammaire de génération des habitants d'un type . . . 39

5 Génération aléatoires des termes en forme β-normale des types simples(J.Wang) 41
5.1 Grammaire de génération des termes en forme β-normale d'un type A avec l'environnement Γ : 42
5.2 Génération des termes d'un type donné 43
6 Enumération des habitants des types simples (G.Dowek et Y.Jiang) 49
6.1 Grammaire pour énumérer les schémas de terme 50
6.2 Grammaire pour énumérer les termes fermés et ouverts 52
6.3 Grammaire infinie pour énumérer les termes normaux fermés 54

3 Proposition d'algorithmes de génération des habitants d'un type 55
1 Machine *Mat* (α, Γ) pour l'énumération des habitants d'un type 55
1.1 Machine d'énumération des habitants en forme β-normale longue 55
2 Les types finement engendrés 57
2.1 Génération des habitants des types de degré 2 à une seule variable 58
2.2 Génération des habitants des types de degré ≤ 3 $(G3)$. 62
2.3 Conclusion 76
3 Génération des habitants d'un type de degré quelconque . . . 76
3.1 Procédure de construction d'un ensemble fini de termes 77

A Programme de génération des termes initiaux, générateurs pour les types finement engendrés de degré ≤ 3 "gen.ml" 93

Bibliographie 104

Remerciement

Ce travail n'aurait pu être effectué sans l'accord, le soutien et l'aide de plusieurs personnes.

J'exprime ma plus vive gratitude au Pr Mohamed Mezghiche pour la confiance qu'il m'a accordé, pour son accueil et son encadrement durant toute la durée de cette thèse, en faisant partager son enthousiasme pour la recherche et sa grande expérience et se rendant toujours disponible malgré son travail.

Je tiens à remercier l'ensemble du jury : son président Pr Ahmed nacer Mohamed, ses trois examinateurs Madame Tebibel Thouraya, Monsieur Ait bouziad Ahmed et Monsieur Rahmoune Fayçal.

Je n'oublierai évidemment pas Monsieur Baddari Kamel Doyen de la faculté des sciences et Monsieur Harzeallah Abdelkarim chef de département informatique.

J'adresse un grand merci à Monsieur Hamdani Chaabane (enseignant à l'université de Tizi Ouzou Ex étudiant poste graduation université de Boumerdes) de m'avoir accueilli pour m'expliquer le travail qu'il a effectué dans sa thèse de magister.

Je ne terminerai pas sans adresser un immense merci à mes parents pour tout ce qu'ils ont fait et pour le soutien qu'ils m'ont apporté durant toutes mes études, et a qui je dois tout.

Dédicaces

Je dédie ce mémoire à :

Mes très chers parents qui se sont toujours dévoués et sacrifiés pour moi, ceux qui m'ont aidé du mieux qu'ils pouvaient pour réussir, ceux qui m'ont accompagné tout au long de ce parcours périlleux, en particulier à ma maman qui été là dans mes moments de détresse qui m'a toujours encouragé et soutenue moralement.

Mon mari qui m'a encouragé et soutenu tout au long de mon parcours, en témoignage de sa gentillesse et sa compréhension.

Ma très chère unique fille (BéBé) Sarah-Rassil à celle que je n'ai pas pu lui donner autant du temps qui lui en faut a cause de mon occupation.

Mes très chers trois frères et ma soeur unique.

Ceux que je n'ai pas cité et qu'ils méritent mon dédicace.

Résumé

Vu l'intérêt des types dans le λ-calcul, nous nous sommes intéressés au problème d'énumération des habitants d'un type.
Un type peut avoir un nombre fini ou infini de termes (habitants), à l'image des types des entiers de church qui peuvent avoir un nombre infini de termes $0, 1, \cdots, \infty$ ces termes peuvent être représentés par un ensemble fini de terme $\{0, S\}$ le zéro et la fonction successeur. Pour un type donné, peut-on définir un ensemble fini de termes qui génère l'ensemble des habitants ?
Dans notre mémoire nous avons étudié plusieurs algorithmes d'énumération proposés par plusieurs auteurs, puis nous avons simplifié et amélioré la procédure proposée par Mr Hamedani (étudiant en Magister Université de Boumerdes) pour les types de degré ≤ 2 , par la suite nous avons proposé une procédure de génération pour une classe plus grande des types, les types finement engendrés de degré ≤ 3 et enfin nous sommes arrivés à exploiter la grammaire à contexte libre de Dowek et Jiang pour générer un ensemble fini de termes (termes initiaux et termes générateurs) qui représente l'ensemble des habitants pour les types de degré quelconque.

Abstract

Given the interest in the types of λ-calculus, we have interested to the problem of enumeration the inhabitants of a type.
A type can have a number finite or infinite of terms, for example the types of integers of Church that may have an infinite number of terms $0, 1, \cdots, \infty$ these terms can be represented by a finite set of term $\{0, \mathrm{S}\}$ the zero and the successor function. For a given type, can we define a finite set of terms that generate all of inhabitants ?
In our thesis we have studied several algorithms for enumeration proposed by authors, first we have simplified and improved a procedure proposed by Mr Hamedani (Ex student university of Boumerdes) for types of degree ≤ 2, next we have proposed a procedure for generation of finely generated types of degree ≤ 3 and finally we have exploited the grammar Dowek and Jiang to generate a finite set of terms (Under the original terms and generators), which represents all inhabitants types of any degree.

Introduction

Les types dans le λ-calcul ont joué souvent un rôle très important (voir [1]). Si on considère d'une part le λ-calcul comme le modèle théorique des langages fonctionnels (Ocaml $\cdots$) les programmes qui se terminent seront interprétés par les λ-termes typés. D'autres part si on considère les types comme les propositions logiques (paradigmes Cury-Howard) (voir [2]) les λ-termes seront interprètes comme les preuves de ces propositions. Dans le cas général dans le λ-calcul, on peut attribuer à chaque λ-terme (habitant de ce type) un type et à chaque type un ensemble de termes (habitants).
Pour résoudre ce problème d'assignation de type et de synthèse de terme, plusieurs auteurs ont proposé des solutions comme Hindely, Hirokawa, Broda, Zaionc, Jue wang $\cdots$ (Voir chapitre 2), nous nous intéressons dans le présent travail au problème d'énumération des habitants d'un type [3]. L'ensemble des habitants d'un type est généralement infini, la question légitime qui reste posée est : peut-on définir un ensemble fini de termes qui génèrent l'ensemble des habitants ?
Dans ce mémoire nous nous proposons d'étudier la solution du problème posé par cette question.
Notre travail sera présenté de la façon suivante :
Dans le chapitre 1 Nous donnons les principales définitions et notations relatives au λ-calcul et au système d'assignation de types (voir [3]). Nous décrirons dans ce chapitre les principaux résultats de λ-calcul.
Dans le chapitre 2 nous exposerons l'état de l'art concernant les différentes solutions proposées pour l'énumération des habitants d'un type.
Dans le chapitre 3 nous présenterons notre contribution.
Enfin, nous terminerons par une conclusion et perspectives.

Chapitre 1

Rappels sur le λ-Calcul et le système d'assignation

Dans ce chapitre nous introduisons les notions de base sur la syntaxe du λ-calcul et les réductions (voir [3, 4, 5]).

1 Les λ-termes et leur structures

Définition 1.1. *L'ensemble des λ-termes est défini récursivement à partir d'un ensemble infini dénombrable de variables de termes selon les trois règles suivantes :*

1. *Toute variable de terme est un λ-terme.*
2. *Si M et N sont des λ-termes alors (MN) est λ-termes.*
3. *Si x est une variable de terme et M un λ-terme alors $(\lambda x.M)$ est un λ-terme.*

On appelle application, tout λ-terme de la forme (MN), et abstraction tout λ-terme de la forme$(\lambda x.M)$. Dans l'abstraction $(\lambda x.M)$, λx est l'abstracteur et M le corps de l'abstraction. Le corps de l'abstraction constitue la portée de l'abstracteur.

Notation 1.1.

1. *$x, y, z, \cdots$ avec ou sans indices, dénoteront des variables de termes.*
2. *$M, N, P, Q, \cdots$ dénoteront des λ-termes arbitraires.*
3. *le symbole '$\equiv$' dénotera l'identité syntaxique ($P \equiv Q$ signifie que P est le même λ-terme que Q), ou la définition.*
4. *les parenthèses seront souvent omises, ainsi nous écrirons :*

(a) $MN_1N_2\cdots N_n$ *au lieu de* $(\cdots((MN_1)N_2)\cdots N_n$ *), et*
(b) $\lambda x_1\lambda x_2\cdots\lambda x_n.M$ *au lieu de* $(\lambda x_1(\lambda x_2\cdots(\lambda x_n.M)\cdots))$.

Définition 1.2. *La longueur d'un λ-terme M, noté /M/ est le nombre d'occurrences de variables dans M, c.à.d :*

1. */ x /=1.*
2. */ M N /= /M / + / N /.*

Définition 1.3. *Les sous-termes d'un λ-terme M sont définis par induction sur la structure de M comme suit :*

1. *Toute variable de terme est sous-terme d'elle même.*
2. *Si $M \equiv \lambda x.P$, alors les sous-termes de M sont tous les sous-termes de P et le terme M.*
3. *Si $M \equiv PQ$, ses sous-termes sont tous les sous-termes de P et de Q et M lui même.*

1.1 Variable liées, termes fermés, substitution :

L'ensemble des variables libres d'un λ-terme M, notation FV(M), est défini inductivement comme suit :

- FV(x) = x.
- FV(MN) =FV(M) U FV(N).
- FV(λx.M) =FV(M) - x.
- Un λ-terme M est dit fermé si FV(M) =$\varnothing$.

Définition 1.4 (Substitution)**.** *On définit [N /x]M le résultat de la substitution de N pour toute occurrence libre de x dans M, plus précisément, on définit pour tout N, x, P, Q et tout $Y \not\equiv x$:*

1. $[N/x]x \equiv N$,
2. $[N/x]y \equiv y$,
3. $[N/x](P,Q) \equiv (([N/x]P)([N/x]Q))$,
4. $[N/x](\lambda x.P) \equiv \lambda x.P$
5. $[N/x](\lambda y.P) \equiv \lambda y.P$ *si* $x \notin FV(P)$
6. $[N/x](\lambda y.P) \equiv \lambda y.[N/x]P$ *si* $x \in FV\ (P)$ *et* $y \notin FV(N)$
7. $[N/x](\lambda y.P) \equiv \lambda z.[N/x][z/y]P$ *si* $x \in FV\ (P)$ *et* $y \in FV(N)$ *et* $z \notin (NP)$.

Le résultat de la substitution simultanée de plusieurs λ-termes $N_1,\cdots,N_n$ à des variables libres distinctes $x_1,\cdots,x_n$ d'un λ-terme M est défini de façon similaire à [N/ x] M et est noté

$$[N_1/x_1, \cdots, N_n/x_n]M.$$

Définition 1.5 (α-conversion)**.** *Soit M un λ-terme et y une variable de terme $\notin FV(M)$ nous avons alors la règle α suivante :*

$$\lambda x.M \equiv_\alpha \lambda y.[y/x]M \quad (\alpha)$$

Le remplacement d'une occurrence de $(\lambda x.M)$ par $\lambda y.[y/x]M$ est appelé changement de nom de variable libre .
Si P est obtenu à partir d'un autre λ-terme Q par une suite finie de changements de nom de variables liées, nous dirons que P et Q sont α-convertibles et nous écrirons

$$P \equiv_\alpha Q.$$

1.2 Réductions et formes normales

Dans cette section nous rappelons des définitions et les propriétés principales de la procédure de réécriture des termes appelé β-réduction ([3][4]).

Définition 1.6 (La β-réduction)**.**

$$P \triangleright_\beta Q$$

1. *Un β-redex est tout λ-terme de la forme $(\lambda.M)N$, son contractum est $[N/x]M$ et sa règle de réécriture est :*

   $$(\lambda x.M)N \to_\beta [N/x]M \quad (\beta)$$

   Cette règle constitue le principal schéma d'axiome du λ-calcul. Si Q est le résultat du remplacement d'un β-redex par son contractum dans un terme P, alors nous dirons que P se β-contract à Q, et on la note :

   $$P \to_\beta Q$$

2. *Une β-réduction d'un terme P est une suite finie ou infinie (pouvant être vide) de β-contractions.*
   On dit que P se β-réduit à Q, notation $P \twoheadrightarrow_\beta Q$, si et seulement si nous avons
   (a) *$P \equiv_\alpha Q$ ou*
   (b) *$P \to_\beta P_1 \to_\beta P_2 \to_\beta \cdots \to_\beta P_n$ avec $n \geq 1$ et $P_n \equiv_\alpha Q$*
   Une β-conversion est une suite finie de β-expansions et de β-contraction.
   Si P s'obtient de Q par β-conversion, nous dirons que P et Q sont β-convertibles ou β-égaux et on note :

$$P =_\beta Q$$

Un λ-terme est en forme β-normale, en abrégé β-nf, s'il ne contient aucun β-redex.
Si $M \twoheadrightarrow_\beta N$ et N est en β-nf, alors nous dirons que M a N pour β-nf.

3. *Une β-expansion est une β-contraction dans le sens inverse.*

Propriété 1.1. *(Unicité et structure d'une β-nf)*

1. *Tout λ-terme a au plus une β-nf (modulo α-conversion).*
2. *Toute β-nf M s'écrit de la forme unique $N \equiv \lambda x_1 \cdots x_m.yN_1 \cdots N_n$ $(m \geq 0, n \geq 0$) Avec $N_1 \cdots N_n$ en β-nf et si N est fermé alors $y \in \{x_1, \cdots x_m\}$.*

Définition 1.7 (La η-réduction et la η-conversion)**.** *Un η-redex est tout terme $\lambda x.Mx$ avec $x \notin FV(M)$, la règle de réécriture est*

$$\lambda x.Mx \rhd_\eta M \quad (\eta)$$

Son contractum est M. Les définitions de η-contraction, η-réduction, η-conversion sont analogues à celles données pour β-concepts.

Définition 1.8 ($\beta\eta$-réduction, $\beta\eta$-conversion)**.** *Un $\beta\eta$-redex est un β- ou η-redex. Les définitions de $\beta\eta$-contracts, $\beta\eta$-réduction, $\beta\eta$-conversion sont analogues au β-concepts.*

1.3 Les λ-termes restreints

Définition 1.9 (λI-terme)**.** *Un λ-terme P est appelé λI-terme ssi pour chaque sous-terme $\lambda x.M$ dans P, x a au moins une occurrence libre dans M.*

Exemple 1.1.
Les λ-termes $\lambda xy.xy, \lambda xy.x(xy)$ sont des λI-termes.
Le λ-terme $\boldsymbol{K} \equiv \lambda xy.x \equiv \lambda x.(\lambda y.x)$ n'est pas un λI-terme.

Définition 1.10 (BCK λ-terme)**.** *Un BCK λ-terme est λ-terme P tel que :*

1. *Pour chaque sous-terme $\lambda x.M$ de P, x a au plus une occurrence libre dans M.*
2. *Toute variable libre de P a exactement une occurrence libre dans P.*

Exemple 1.2.
Les temes $\lambda xy.x, \lambda xy.y$ sont des BCK λ-termes.
Les temes $\lambda xy.x(xy), \lambda xy.x(x(xy))$ ne sont pas des BCK λ-termes.

Définition 1.11 (BCI λ-terme). *Un BCI λ-terme ou bien linéaire λ-terme est un λ-terme P tel que :*

1. *Pour chaque sous-terme $\lambda x.M$ de P, x a exactement une occurrence libre dans M.*
2. *Chaque variable libre de P a exactement une occurrence libre dans P.*

Exemple 1.3.
Les λ-termes $\boldsymbol{C} \equiv \lambda xyz.xyz$, $\boldsymbol{B} \equiv \lambda xyz.x(yz)$ sont des BCI λ-termes.
Le λ-terme $\lambda xy.xyy$ n'est pas un BCI λ-terme.

Définition 1.12. *La β-contraction $(\lambda x.M)N \triangleright_\beta [N/x]M$ est dite une élimination de N ssi l'occurrence de x n'est pas libre dans M, est dite une duplication de N ssi x a au moins deux occurrences dans M.*

2 Assignation des types aux termes

2.1 Le système d'assignation de types dans λ-calcul (TA_λ)

Définition 2.1 (Les types). *Etant donnée une séquence infinie de variables de type, les types sont définis inductivement comme suit :*
- *Chaque variable de type est un type (type atomique).*
- *Si α et β sont des types alors $(\alpha \to \beta)$ est un type (type composé).*

Notation 2.1.
- *Les variables de types sont notées par : 'a','b','c','o','p', avec ou sans indice.*
- *Les types arbitraires sont notés par les symboles grec à l'exception de 'λ'.*
- *Les parenthèses sont souvent omises dans les types (en appliquant l'associative à droite)*

  Exemple 2.1. $a \to b \to c \equiv (a \to (b \to c))$.

Formellement, pour interpréter les types on représente chaque variable de type comme un ensemble et $a \to b$ comme un ensemble de fonctions de a dans b.

Définition 2.2 (Assignation des types). *L'assignation des types est toute expression de la forme $M : \tau$ où, M est un λ-terme et τ est un type donné. On appelle M le sujet et τ le prédicat.*

Définition 2.3 (Contexte type Γ)**.** *Le contexte type* Γ *est un ensemble fini tel que :*

$$\Gamma = \emptyset \text{ ou } \Gamma = \{x_1 : p_1, \cdots, x_n : p_n\} .$$

où chacun des $x_i : p_i$ $(0 < i \leq n)$ *est une assignation de type, et toute variable de terme* x_i *est* ***consistante*** *(****monovalente****) avec le contexte* Γ*, c.à.d chaque variable de terme* $x_i \in \Gamma$ *doit avoir une seule assignation.*

Définition 2.4 (La formule de TA_λ)**.** *Pour tout* Γ*,* M *et* τ *le triplet* $< \Gamma, M, \tau >$ *est appelé une formule du système est noté comme suit :*
$\Gamma \vdash M : \tau$ *ou bien* $\vdash M : \tau$ *si* $\Gamma = \emptyset$*.*

Définition 2.5 (Le système de TA_λ)**.** *Le système de* TA_λ *a un ensemble infini d'axiomes et deux règles de déductions appelées (*$(\rightarrow E)$ *règle d'élimination,* $(\rightarrow I)$ *règle d'introduction). Il est défini comme suit :*

1. *Les axiomes : Pour chaque variable de terme* x *et pour chaque variable de type* τ*, la formule*

   $$x : \tau \vdash x : \tau$$

   est un axiome de TA_λ
2. *Les règles de déductions :*

$(\rightarrow E)$	$\dfrac{\Gamma_1 \vdash P : (\sigma \rightarrow \tau) \quad \Gamma_2 \vdash Q : \sigma}{\Gamma_1 U \Gamma_2 \vdash (PQ) : \tau}$	(*si* $\Gamma_1 U \Gamma_2$ *est consistent*)
$(\rightarrow I)$	$\dfrac{\Gamma \cup \{x : \sigma\} \vdash P : \tau}{\Gamma \vdash \lambda x.P : \sigma \rightarrow \tau}$	(*si* Γ *est consistent avec* x)

Une déduction Δ *de* TA_λ *est un arbre de* TA_λ*-formules. Les sommets des branches représentent les axiomes. Les noeuds intérieurs sont déduits immédiatement des noeuds précédents par l'application des règles de déductions . Les formules des noeuds de l'arbre représentent les conclusions.*
Si on a $\Gamma \vdash M : \tau$*, alors on appelle l'arbre* Δ *la déduction de* $\Gamma \vdash M : \tau$ *. Si* $\Gamma = \emptyset$*, alors dans ce cas, la déduction* Δ *est appelée preuve de* $M : \tau$ *(voir [3]).*

Exemple 2.2. *Soit* $B \equiv \lambda xyz.x(yz)$ *l'assignation de type est comme suit :*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x:a\to b\vdash x:a\to b}\quad\dfrac{\dfrac{}{y:c\to a\vdash y:c\to a}\quad\dfrac{}{z:c\vdash z:c}(\to E)}{y:c\to a,z:c\vdash yz:a}(\to E)}{x:a\to b,y:c\to a,z:c\vdash x(yz):b}(\to I)}{x:a\to b,y:c\to a\vdash \lambda z.x(yz):(c\to b)}(\to I)}{x:a\to b\vdash \lambda yz.x(yz):(c\to a)\to(c\to b)}(\to I)}{\vdash \lambda xyz.x(yz):(a\to b)\to(c\to a)\to(c\to b)}(\to I)$$

Définition 2.6 (Typabilité)**.** *Un terme M est appelé (TA_λ)-typable ssi ils existent Γ et τ tel que :*

$$\Gamma \vdash_\lambda M : \tau.$$

Lemme 2.1. *La classe de tous les λ-termes typables est fermée sous les opérations suivantes :*

1. *Tous les sous-termes des termes typables sont typables.*
2. *$\beta\eta$-réduction.*
3. *La β-expansion sans élimination et sans duplication.*
4. *Si M est typable alors $\lambda x.M$ est typable.*

Définition 2.7 (Types(M))**.** *Si M est fermé on définit Type(M) comme suit :*

$$Type(M) = \{\tau | \vdash_\lambda M : \tau\}$$

Propriété 2.1. *Soit P un terme fermé alors :*

1. *$P \rhd_\beta Q \Rightarrow Types(P) \subseteq Type(Q)$*
2. *Si $P \rhd_\beta Q$ par une réduction qui n'utilise pas une élimination ou une duplication alors : $Types(P) = Types(Q)$.*

Théorème 2.1. *La classe de tous les termes TA_λ-typable est décidable, tel que il existe un algorithme qui décide pour un terme donné dans TA_λ s'il est typable.*

2.2 Substitution des types

La substitution **s** de type est toute expression de la forme :

$$[\sigma_1/a_1, \cdots, \sigma_n/a_n]$$

où $a_1, \cdots, a_n$ sont des variables de type et $\sigma_1, \cdots, \sigma_n$ sont des types quelconques. pour tout type τ on définit $\mathbf{s}(\tau)$ la substitution simultanée de σ_1 par $a_1, \cdots, \sigma_n$ par a_n dans τ .

Définition 2.8.

1. $\boldsymbol{s}(a_i) = \sigma_i$.
2. $\boldsymbol{s}(b) = b$. *si* b *est un atome* $\notin \{a_1 \cdots a_n\}$.
3. $(\rho \to \sigma) = s(\rho) \to s(\sigma)$.

On appelle $s(\tau)$ *une instance de* τ.

Notation 2.2. *On appelle les lettres '*$\boldsymbol{r}$*','*$\boldsymbol{s}$*','*$\boldsymbol{t}$*','*$\boldsymbol{u}$*','*$\boldsymbol{v}$*' les substitutions des types .*
Si $\boldsymbol{s}$=$[a_1/\sigma_1, \cdots, a_n/\sigma_n]$ *alors* $\boldsymbol{s}(\tau)$= $[a_1/\sigma_1, \cdots, a_n/\sigma_n]\tau$.
On appelle $Vars(\tau)$ *l'ensemble de toutes les occurrences des variables dans le type* τ.
On appelle $Vars(\tau_1, \cdots, \tau_n)$, *l'ensembles de toutes les occurrences des variables de types dans la séquence finie de types*$< \tau_1, \cdots, \tau_n >$
On appelle $Vars(\Delta)$, *l'ensemble des occurrences de variables de types dans la déduction* $\triangle$.

Définition 2.9. *Soient la séquence de type* $< \tau_1, \cdots, \tau_n >$, *le contexte* Γ *et la* TA_λ*-formule* $M : \tau$.
L'action de substitution $\boldsymbol{s}$ *est définie :*
$\boldsymbol{s}(<\tau_1, \cdots, \tau_n>) = < \boldsymbol{s}(\tau_1), \cdots, \boldsymbol{s}(\tau_n)>$.
$s(\Gamma) = \{x_1 : \boldsymbol{s}(\tau_1), \cdots, x_m : \boldsymbol{s}(\tau_m)\}$. *si* $\Gamma = \{x_1 : \tau_1, \cdots, x_m : \tau_m\}$.
$\boldsymbol{s}(\Gamma \vdash M : \tau) = \boldsymbol{s}\ (\Gamma) \vdash M : \boldsymbol{s}(\tau)$.

Propriété 2.2.

1. *Si* Γ *est consistent alors* $\boldsymbol{s}(\Gamma)$ *est consistent.*
2. *Si* Δ *est un* TA_λ*-déduction alors la propriété reste vraie pour* $\boldsymbol{s}(\Delta)$*(voir [3] page 31)*

Définition 2.10 (type principal dans TA_λ)**.** *Le type principal (PT) d'un terme* M *est le type* τ *tel que :*
- $\Gamma \vdash_\lambda M : \tau$ *pour un certain* Γ.
- *Si* $\Gamma' \vdash_\lambda M : \sigma$ *pour un certain* Γ' *et* σ *alors* σ *est une instance de* τ.

Définition 2.11 (Unificateur)**.**

1. *S'il existe une substitution* $\boldsymbol{s}$ *tel que* $\boldsymbol{s}(\rho)$=$\boldsymbol{s}(\tau)$ *on dit que* $<\rho,\tau>$ *est unifiable .*
   On appelle $\boldsymbol{s}$ *l'unificateur de* $<\rho,\tau>$ *et* $\boldsymbol{s}(\rho)$ *une unification de* $<\rho,\tau>$.

2. *L'unificateur* $\boldsymbol{s}$ *de la séquence paire* $\boldsymbol{s} \ll \rho_1, \cdots, \rho_n >, \boldsymbol{s} < \tau_1, \cdots, \tau_n \gg$ *de même taille est :*

$$\boldsymbol{s}(< \rho_1, \cdots, \rho_n >) \equiv \boldsymbol{s}(< \tau_1, \cdots, \tau_n >)$$

Définition 2.12 (M.g.u)**.** *L'unificateur le plus général (m.g.u) de la paire* $<\rho,\tau>$ *est l'unificateur* $\boldsymbol{u}$ *tel que pour tout autre unificateur* $\boldsymbol{s}$ *de* $<\rho,\tau>$ *on a :*

$$\boldsymbol{s}(\rho) \equiv \grave{\boldsymbol{s}}(\boldsymbol{u}(\rho)) \qquad \textit{pour tout } \grave{\boldsymbol{s}}$$

Si $\boldsymbol{v} \equiv \boldsymbol{u}(\rho)$ *pour tout (m.g.u)* $\boldsymbol{u}$ *de* $< \rho, \tau >$ *on appelle* $\boldsymbol{v}$ *unificateur le plus général (m.g.u) de* $< \rho, \tau >$.

Exemple 2.3. *La paire* $< a \to (b \to b), (c \to c) \to a >$ *est unifiée par le plus général unificateur* $u \equiv [(b \to b)/a, b/c]$ *le résultat est le type :*

$$(b \to b) \to (b \to b)$$

Chapitre 2

Génération des habitants d'un type

1 Algorithme de Ben-yelles et Hindely

Dans cette partie nous présentons l'algorithme de Ben-yelles et Hindely (voir[3] page 108-139 et [6]) qui répond à la question "combien de termes fermés en forme normale peut avoir le type τ dans le système TA_λ ?". Pour chaque type τ l'algorithme décidera en un nombre finis d'étapes si le nombre de termes en forme β-normal est fini ou infini.
L'algorithme calculera ce nombre dans le cas fini et énumère (lister) les termes dans les deux cas.

Définition 1.1 (Termes typés, $TT(\Gamma)$)**.** *Soit le contexte Γ, l'ensemble $TT(\Gamma)$ des termes typés relatifs à Γ est l'ensemble des expressions définit comme suit :*

1. *Si $x : \sigma \in \Gamma$ alors l'expression $x^\sigma \in TT(\Gamma)$ est appelée variable typée.*
2. *Si $\Gamma_1 \cup \Gamma_2$ est consistent et $M^{\sigma\to\tau} \in TT(\Gamma_1)$ et $N^\sigma \in TT(\Gamma_2)$, alors $(M^{\sigma\to\tau}N^\sigma)^\tau \in TT(\Gamma_1 \cup \Gamma_2)$.*
3. *Si Γ est consistent avec $\{x : \sigma\}$ et $M^\tau \in TT(\Gamma)$ alors $(\lambda x^\sigma.M^\tau)^{\sigma\to\tau} \in TT(\Gamma - x : \sigma)$.*
   Si M^τ est un terme typé (Pour un certain Γ), τ est appelé le type de M^τ.

Lemme 1.1 (Structure de β-nf typé)**.** *Soit le contexte Γ, chaque β-nf $M^\tau \in TT(\Gamma)$ (terme typé) peut être exprimer sous l'unique forme suivante :*

- $(\lambda x_1^{\tau_1} \cdots x_m^{\tau_m}.(\nu^{(\rho_1\to\cdots\rho_n\to\tau^*)}M_1^{\rho_1} \cdots M_n^{\rho_n})^{\tau^*})^{(\tau_1\to\cdots\to\tau_m\to\tau^*)}$
  où $(m \geq 0, n \geq 0)$ et $\tau \equiv \tau_1 \to \cdots \to \tau_m \to \tau^*$ pour tout τ^* (τ^* peut être composé) et pour tout $M_j^{\rho_j}$ en β-nf relatif à $\Gamma \cup \{x_1 : \tau_1, \cdots, x_m : \tau_m\}$.

Si M^τ est fermé alors $m \geq 1$ et il existe $i \leq m$ tel que : $\nu \equiv x_i$, $\tau_i \equiv (\rho_1 \to \cdots \to \rho_n \to \tau^*)$

Définition 1.2.

1. *Depth(M) (Profondeur d'un terme) :*

   - $Depth(y) = Depth(\lambda x_1 \cdots x_m.y) = 0.$
   - $Depth(\lambda x_1 \cdots x_m.yM_1 \cdots M_n) = 1 + Max_{1 \leq j \leq n} Depth(M_j) \quad (n > 0)$

2. *Long β-nfs : le terme M^τ typé en β-nf est long ou maximal ssi chaque occurrence de variable Z dans M^τ est suivie par la plus longue séquence d'arguments autorisée par son type c.à.d ssi chaque composante de la forme $(Zp_1 \cdots p_n)$ $(n \geq 0)$ qui n'est pas dans la position de fonction a un type atomique .*

   Exemple 1.1. *Soit le type $\tau \equiv ((a \to b) \to c) \to (a \to b) \to c$ l'habitant normal suivant n'est pas en forme normale longue :*

   $$M^\tau \equiv \lambda x^{(a \to b) \to c} y^{a \to b}.x^{(a \to b) \to c} y^{a \to b}$$

   Par contre l'habitant suivant est en forme normale longue :

   $$N^\tau \equiv \lambda x^{(a \to b) \to c} y^{a \to b}.x^{(a \to b) \to c} (\lambda z^a.y^{a \to b} z^a)$$

3. *Un habitant de type τ est le terme fermé M^τ, l'ensemble des habitants de type τ est noté :*

   $$Habs(\tau)$$

4. *L'ensemble des habitants en β-nf de type τ est noté :*

   $$Nhabs(\tau)$$

5. *L'ensemble de tous les habitants en forme normale longue de type τ est noté :*

   $$Long(\tau)$$

6. *L'ensemble des habitants en forme normale longue de type τ avec $Depth \leq d$ est noté :*

   $$Long(\tau, d)$$

7. *L'ensemble de tous les termes obtenus pas la η-réduction M^τ est appelé la η-famille de M^τ est noté :*

   $$\{M^\tau\}_\eta$$

Lemme 1.2. *Chaque habitant normal de τ peut être η-expansion à un habitant normal long de type τ, et cet habitant long est unique (modulo $\equiv_\alpha$) c.à.d*

$$\{M^\tau, N^\tau \in Long(\tau) \text{ et } M^\tau =_\eta N^\tau\} \Rightarrow M^\tau \equiv_\alpha N^\tau.$$

Corollaire 1.1.
$Nhabs(\tau) = \emptyset \Longleftrightarrow Long(\tau) = \emptyset$

Note 1.1.
- *Soit $M^\tau \in TT(\Gamma)$ pour tout Γ, alors $\{M^\tau\}_\eta$ est fini et tous ses membres sont dans $TT(\Gamma)$.*
- *Si M^τ est en $\beta\eta$-forme alors tous les membres de $\{M^\tau\}_\eta$ sont en $\beta\eta$-forme.*
- *$M^\tau \in Nhabs(\tau) \Longrightarrow \{M^\tau\}_\eta \subseteq Nhabs(\tau)$ aussi, si M^τ est en β-nf sa η-famille contient exactement un $\beta\eta$-nf.*
- *Chaque habitant normal de τ dans la η-famille a exactement un habitant normal long.*

Notation 1.1. *Chaque type est représenté sous la forme suivante :*

$$\tau \equiv \tau_1 \to \cdots \tau_m \to e \quad (m \geq 0)$$

e : atome.
On appelle $\tau_1, \cdots, \tau_m$: prémisses.
Deux occurrences type sont isomorphes ssi elles sont des occurrences d'un même type, tel que tail (σ)$\cong$ tail (τ)

Définition 1.3 (Long β-nf typé)**.** *Soit le type τ tel que $\tau \equiv \tau_1 \to \cdots \to \tau_m \to e$ ($m \geq 0$, e un atome) et M^τ un β-nf de type τ a la forme :*

$$(\lambda x_1^{\tau_1} \cdots x_k^{\tau_k}.(\nu^{(\rho_1 \to \cdots \rho_n \to \tau^*)} M_1^{\rho_1} \cdots M_n^{\rho_n})^{\tau^*})^{(\tau_1 \to \cdots \to \tau_k \to \tau^*)}$$

$0 \leq k \leq m$ *et* $\tau^* \equiv \tau_{k+1} \to \cdots \to \tau_m \to e$.
Si M^τ est long alors :

1. $k = m$
2. $\tau^* = e$
3. *Les types de $x_1, \cdots, x_m$ coïncide avec les prémisses de τ.*
4. *La queue (tail) du type ν est isomorphe à celle de τ.*
5. *Si M^τ est fermé alors $m \geq 1$ et $\nu \in x_i$ $(1 \leq i \leq m)$ et $\tau_i \equiv \rho_1 \to \cdots \to \rho_n \to e$*

Définition 1.4 (nf-scheme)**.** *On suppose une séquence infinie des expressions appelées méta-variables notées "V", "V_1", "V_2", $\cdots$ distinctes les unes des autres, et distinctes des variables de termes.*
Un nf-scheme est défini comme λ-terme mais il doit respecter les règles suivantes.

1. *Tout nf-scheme est en β-nf.*
2. *Les méta-variables ne sont pas liées c.à.d. λV n'existe pas.*
3. *Les méta-variables occupent toujours les positions d'arguments.*
4. *Chaque méta-variable dans nf-scheme apparait seulement une seule fois.*

nf-scheme : *est propre s'il contient au moins une méta-variable, un nf-scheme typé est défini comme un λ-terme typé en plus il doit satisfaire les conditions précédentes, de la même manière pour une β-nf longue on définit un nf-scheme long.*

Théorème 1.1. *L'algorithme de recherche suivant, accepte en entrée tout type α et donne en sortie une séquence finie ou infinie, d'ensembles $A(\alpha, d)$ (d=0,1,2,$\cdots$), tels que pour tout $d \geq 0$:*

1. *Chaque membre de A(α,d) est un nf-scheme long, fermé de type α, et il est soit un nf-scheme propre de depth d, ou un terme de depth (d-1).*
2. *A(α,d) est fini.*
3. *Long (α,d) $\subseteq$ A(α,0)$\cup$ A(α,1)$\cup\cdots\cup$A(α,d+1) où Long(α,d) est l'ensemble de tous les termes dans Long (α) de profondeur inférieure ou égale à d+1,*
4. *Long(α) = $A_{termes}(\alpha, 0) \cup A_{termes}(\alpha, 1) \cup \cdots$*
   où $A_{termes}(\alpha, d)$ est l'ensemble de tous les termes dans A(α,d)

La base de l'algorithme de comptage est l'algorithme de recherche, ce dernier va chercher les habitants en forme normale longue de τ de plus en plus profond d=0,1,2,$\cdots$
La stratégie de cet algorithme dépend essentiellement de la structure des termes typés longs.

1.1 Algorithme de recherche

Entrée : Un type τ.
Sortie : Une séquence finie ou infinie de $A(\tau, d)$ tel que $d \geq 0$.
Si $\tau \equiv e$ (e est un atome) **alors** "pas d'habitant"

Sinon
Etape 0 : Choisir une méta-variable quelconque V et définir

$$A(\tau, 0) = \{V^\tau\}$$

Etape d+1 : On suppose que $A(\tau, d)$ est défini
Si $A(\tau, d) = \emptyset$ ou $A(\tau, d)$ ne contient aucune méta-variable **alors** arrêter l'algorithme de recherche, la sortie est la séquence finie suivante :

$$A(\tau, 0), \cdots, A(\tau, d)$$

Sinon commencer la construction de $A(\tau, d+1)$, lister tous les nf-schemes propres dans $A(\tau, d)$ et appliquer (1) et (2) pour chacun.

(1) Soit le nf-scheme $M^\tau \in A(\tau, d)$, lister toutes les méta-variables dans M^τ tels que :

$$V_1^{\beta_1}, \cdots, V_n^{\beta_n}$$

appliquer (a) et (b) pour chacune des méta-variables.

(a) Soit V^σ une méta-variable dans $M^\tau \in A(\tau, d)$, et soit

$$\sigma \equiv \sigma_1 \to \cdots \to \sigma_m \to a \qquad (m \geq 0) \tag{1.1}$$

Si m=0 **alors** appliquer directement (b)
Sinon
pour tout $i \leq m$ tel que tail (σ_i)= tail (σ)= $\underline{a}$

$$\sigma_i \equiv \sigma_{i_1} \to \cdots \to \sigma_{i_{ni}} \to a \qquad n_i \geq 0 \tag{1.2}$$

définir

$$M_i^\sigma \equiv \lambda x_1^{\sigma_1} \cdots x_m^{\sigma_m}.(x_i^{\sigma_i} V_{i1}^{\sigma_{i1}} \cdots V_{ini}^{\sigma_{ini}})^a \tag{1.3}$$

comme remplaçant de V^σ. Où les x_s et V_s sont des nouvelles variables et des méta-variables distinctes.

(b) Citer les abstracteurs $\lambda y_j^{\gamma_j}$ qui couvrent l'unique occurrence de V^σ dans M^τ tels que :
Tail $(y_j) \cong a$ pour tout j, y_j de la forme .

$$y_j \equiv y_{j_1} \to \cdots \to y_{j_{hj}} \to a \qquad (hj \geq 0) \tag{1.4}$$

définir

$$N_j^{\sigma} \equiv \lambda x_1^{\sigma_1} \cdots x_m^{\sigma_m}.(y_j^{y_j} V_{j1}^{y_{j1}} \cdots V_{j_{hj}}^{y_{j_{hj}}})^a \tag{1.5}$$

Comme remplaçant de V^{σ}, où les x_s et les V_s sont des nouvelles variables et des méta-variables distinctes.

(2) Si (a) et (b) sont appliqués à toutes les méta-variables dans M^{τ} le résultat est une liste d'une suite de remplacement pour chaque $V_i \in M^{\tau}$. Si une ou plusieurs méta-variables n'ont pas une suite de remplacement, abandonner M^{τ} (appelé rejet), et passer au membre suivant de $A(\tau, d)$ (le rejet ne génère aucun membre).
Si $\forall V_i \in \{V_1, \cdots, V_n\}$ dans M^{τ} ont tous une suite de remplacement, M^{τ} est appelé extensible, dans ce cas citer toutes les séquences possibles :

$$\langle w_1^{\beta_1}, \cdots, w_n^{\beta_n} \rangle$$

remplacer simultanément les v_i par w_i (pour i=1 à n) dans M^{τ} et construire le nouveau nf-scheme $M^{*\tau}$.
On appelle $\langle w_1^{\beta_1}, \cdots, w_n^{\beta_n} \rangle$ une suite de multi-remplacements et $M^{*\tau}$ une extension de M^{τ}, si cette extension est un terme alors il est appelé un succès.

1.2 Algorithme de comptage

L'algorithme de recherche sera étendu pour compter et énumérer les habitants d'un type.

Définition 1.5. *$A(\tau, d)$ et $A_{terms}(\tau, d)$ sont définis comme suit :*
$A(\tau, \leq d) = A(\tau, 0) \cup \cdots \cup A(\tau, d)$
$A_{terms}(\tau, \leq d) = A_{terms}(\tau, 0) \cup \cdots \cup A_{terms}(\tau, d)$

Rappel : pour tout τ.
$|\tau|$: est le nombre des occurrences d'atomes dans τ.
$||\tau||$: est le nombre des atomes distincts dans τ.

$$\mathrm{D}(\tau) = |\tau| \times ||\tau||$$

Nous avons les propriétés suivantes établies par Ben-yelles (voir preuve [3] page 125) :

- Si Long (τ) a un membre M^{τ} avec Depth d $\geq ||\tau||$ alors il est infini.

– Long (τ) a un membre M^τ avec Depth d $\geq D(\tau)$ alors il a un membre N^τ avec :

$$||\tau|| \leq Depth(N^\tau) < D(\tau)$$

L'algorithme suivant donne # (Long(α)) (le nombre d'habitants en forme normale longue) et Long(α)

1.3 Algorithme de comptage et d'énumération des Long (α)

Si $\alpha = e$ (atome) alors Long (α)= $\emptyset$.
Si $\alpha = \alpha_1 \to \cdots \alpha_n \to e$ (α composé) alors appliquer l'algorithme de recherche à α, on obtient en sortie une séquence finie ou infinie d'ensemble A(α) ($d = 0, 1, 2, \ldots$), arrêter l'algorithme de recherche à la profondeur $d = D(\alpha)$.
cas 1 : Si $A_{terms}(\alpha, \leq D(\alpha)) = \emptyset$ alors Long (α)= $\emptyset$.
cas 2 : Si $A_{terms}(\alpha, \leq D(\alpha))$ a un membre avec depth $\geq ||\alpha||$, alors Long (α) est infini.
Pour énumérer Long (α), appliquer l'algorithme de recherche pour $A_{terms}(\alpha, d)$ ($d = 0, 1, 2, \cdots$)
cas 3 : Si $A_{terms}(\alpha, \leq D(\alpha))$ a des membres qui ont tous une profondeur $Depth < ||\alpha||$, alors Long (α)= $A_{terms}(\alpha, \leq D(\alpha))$ est fini.

Exemple 1.2. *Soit le type* $\tau \equiv (a \to b \to c) \to (a \to b) \to a \to c$
L'algorithme produira le résultat :
$A(\tau, 0) = \{V^\tau\}$
$A(\tau, 1) = \{\lambda x_1^{a\to b\to c} x_2^{a\to b} x_3^a . x_1^{a\to b\to c} V_1^a V_2^b\}$
$A(\tau, 2) = \{\lambda x_1^{a\to b\to c} x_2^{a\to b} x_3^a . x_1^{a\to b\to c} x_3^a (x_2^{a\to b} V_3^a)\}$
$A(\tau, 3) = \{\lambda x_1^{a\to b\to c} x_2^{a\to b} x_3^a . x_1^{a\to b\to c} x_3^a (x_2^{a\to b} x_3^a)\}$

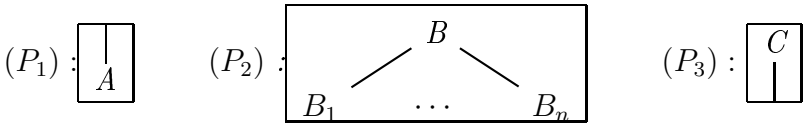
Exemple 1.3. *Soit le type* $((a \to b) \to a) \to a$
L'algorithme produira le résultat suivant :
$A(\tau, 0) = \{V^\tau\}$
$A(\tau, 1) = \{\lambda x_1^{(a\to b)\to a} . x_1^{(a\to b)\to a} V_1^{a\to b}\}$
$A(\tau, 2) = \emptyset$.

2 Algorithme de Broda et Damas

Dans cette partie (voir[7, 8, 9, 10, 11]) il sera montré que l'ensemble des habitants d'un type τ peut être représenté à l'aide des grammaires à contexte libre .
La méthode présentée dans cette section permet d'étudier la relation entre la structure des formules (types) et leurs preuves (termes), la définition d'un schéma de grammaire commun est basée sur la représentation des types.

Définition 2.1 (Arbre de formule (méthode preuve))**.** *Chaque type φ est devisé en parties primitives, et les parties primitives forment une structure d'arbre appelée arbre formule de φ (tree (φ)), l'arbre formule de φ est utilisé pour construire l'arbre-preuve de φ, chaque arbre-preuve sera une représentation des habitants normaux longs de φ. φ peut avoir un nombre fini n ($n \geq o$) ou infini des différents arbre-preuve.*

Définition 2.2. *(Parties primitives) les parties primitives ont les formes suivantes (P_1) , (P_2), (P_3) oú A, B, B_1, $\cdots$, B_n, C dénotent les variables de types :*



Ici A, B_1, $\cdots$, B_n sont appelés les feuilles (variables). B et C sont appelés têtes de variables, le nombre d'arités des parties primitives égale au nombre des feuilles (variables).

Définition 2.3. *Un arbre formule est en forme d'une structure d'arbre, les noeuds sont des parties primitives, la structure est appelée arbre formule ssi :*

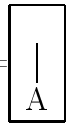
- *La racine de l'arbre formule est sous la forme (P_1)*
- *Chaque noeud de la forme (P_2) et (P_3) dans l'arbre formule est un descendant d'une feuille (variable) d'une autre partie primitive.*
- *Chaque partie primitive étiquetée seulement une seule fois dans l'arbre formule.*

2.1 Algorithme arbre-formule

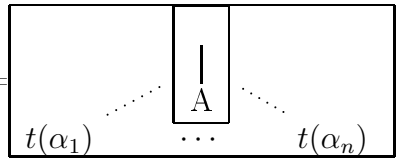
Entrée : le type $\varphi = \alpha_1 \to \cdots \alpha_n \to A$ où A est un atome et n ≥ 0.
Sortie : arbre formule de type φ (tree(φ)).

L'arbre formule tree(φ) est donné par :

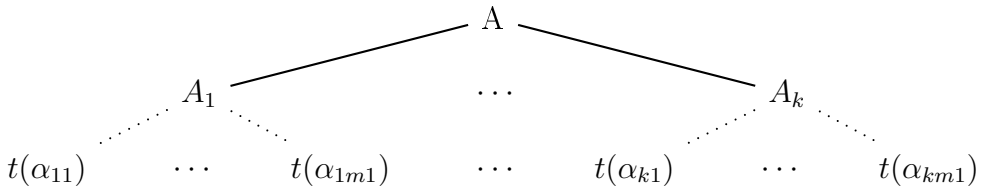
– Si $n = 0$, c.à.d $\varphi \equiv A$, alors tree(φ) =



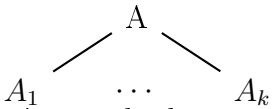
– Si $n \geq 1$, alors tree(φ)=



pour k ≥ 1 et $m_1, \cdots, m_k \geq 0$ on définit récursivement
$t((\alpha_{11} \to \cdots \to \alpha_{1m1} \to A_1) \to \cdots \to (\alpha_{k1} \to \cdots \to \alpha_{km_k} \to A_k) \to A)$

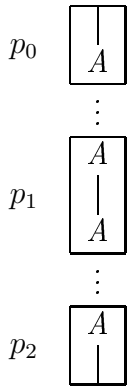


pour $1 \leq j \leq k$, on appelle les parties primitives aux sommets de $t(\alpha_{j1}), \cdots, t(\alpha_{jm_j})$ les descendants de la partie primitive



à la branche j, dans le cas $m_j = 0$, la feuille variable A_j n'a pas de descendants.

Exemple 2.1. *Soit le type* $((A \to A) \to A) \to A$ *l'arbre-formule correspondant est :*



Lemme 2.1. *Soit M un habitant en forme normale longue de type* φ *avec l'arbre formule tree* (φ)*, on considère la variable x dans M et soit P une*

partie primitive dans tree(φ) correspondant à x, si P à k arités (k≥0), alors x apparait toujours avec exactement k arguments dans M, de plus, pour k ≥1, si i ∈ {1,$\cdots$,k} et P à des descendants P_1, $\cdots$,P_m à la branche/feuille variable i dans tree (φ), alors le i^{eme} argument de x est de la forme $\lambda y_1 \cdots y_m.N$ et les parties primitives correspondantes à $y_1, \cdots, y_m$ sont respectivement $P_1, \cdots, P_m$.

Définition 2.4 (Structure d'un arbre-preuve (PT)). *L'arbre-preuve d'un type φ a une forme d'une structure d'arbre, ses noeuds sont des parties primitives dans pp(φ) =$\{p_0, \cdots, p_n\}$ l'ensemble des parties primitives dans l'arbre-formule de φ et p_0 dénote la partie primitive racine de tree (φ) (la seule partie primitive de la forme p_1 dans pp(φ)). Dans la suite on doit utiliser la notation* $\left|\begin{array}{c}\vdots\\ A\end{array}\right|$ *pour désigner l'arbre-preuve qui possède une partie primitive feuille (tail) variable A.*
L'ensemble arbres-preuve pour le type φ est donné par les points suivants :

- *Si* $p_0 = \left|\begin{array}{c}|\\ A\end{array}\right| \in pp(\varphi)$, *alors* $\left|\begin{array}{c}|\\ A\end{array}\right|$ *est l'arbre-preuve de φ*
- *Si* $\left|\begin{array}{c}\vdots\\ B\end{array}\right|$ *est l'arbre preuve de φ et* $\left|\begin{array}{ccc} & B & \\ B_1 & \cdots & B_k\end{array}\right| \in pp(\varphi)$ *oú* $k \geq 1$, *alors* $\begin{array}{c}\left|\begin{array}{c}|\\ B\end{array}\right|\\ \|\\ \left|\begin{array}{ccc} & B & \\ B_1 & \cdots & B_k\end{array}\right|\end{array}$ *est l'arbre-preuve de φ*
- *Si* $\left|\begin{array}{c}\vdots\\ B\end{array}\right|$ *est l'arbre-preuve de φ et* $\left|\begin{array}{c}B\\ |\end{array}\right| \in pp(\varphi)$ *alors* $\begin{array}{c}\left|\begin{array}{c}\vdots\\ B\end{array}\right|\\ \|\\ \left|\begin{array}{c}B\\ |\end{array}\right|\end{array}$ *est l'arbre-preuve de φ.*

Définition 2.5 (Arbre preuve-valide). *L'arbre-preuve PT d'un type φ est appelé arbre-preuve valide de φ ssi :*

- *Le nombre de sous-arbres enracinés dans chaque noeud/primitive dans PT (arbre-preuve) est égale au nombre d'arités de cette partie primitive.*
- *Si* $p' = \left|\begin{array}{ccc} & B & \\ B_1 & \cdots & B_k\end{array}\right|$, *avec $k \geq 1$ est une partie primitive dans*

l'arbre de formule tree (φ), et p est un descendant de p' dans tree (φ) à la branche i, avec 1$\leq i \leq k$, alors en dessus de chaque occurrence $\bar{p}$ de p dans PT il y'a au moins une occurrence $\bar{p}'$ de p' dans PT, tel que l'occurrence $\bar{p}$ apparait dans le sous-arbre de PT enraciné dans la i^{eme} branche de $\bar{p}'$.
Cette définition décrit la hiérarchie de tree(φ) qui sera imposée pour construire l'arbre-preuve valide.

Exemple 2.2. *Arbres preuve-valide pour l'exemple 2.1 sont :*

$$\begin{matrix} p_0 \\ p_1 \\ p_2 \end{matrix} \quad , \quad \begin{matrix} p_0 \\ p_1 \\ p_1 \\ p_2 \end{matrix} \quad , \quad \begin{matrix} p_0 \\ p_1 \\ p_1 \\ p_1 \\ p_2 \end{matrix} \quad ,\cdots \qquad c.à.d \qquad \begin{matrix} \vert \\ A \\ \vert \\ A \\ \vert \\ A \\ \vert \end{matrix} \quad , \quad \begin{matrix} \vert \\ A \\ \vert \\ A \\ \vert \\ A \\ \vert \end{matrix} \quad , \quad \begin{matrix} \vert \\ A \\ \vert \\ A \\ \vert \\ A \\ \vert \end{matrix} \quad ,\cdots$$

2.2 Algorithme terme

Entrée : Un type φ, et l'arbre-preuve valide de φ.
Sortie : l'ensemble fini non vide des habitants normaux long fermé de φ(termes (PT)).
Soient $p_0, \cdots, p_n$ les parties primitives dans tree(φ), p_0 est la racine et $n \geq 0$, l'ensemble termes (PT) est construit comme suit :

a Représenter PT sous forme d'application pour $0 \leq i \leq n$, remplacer p_i par x_i, alors pour chaque occurrence de variable x_i dans l'application tel que la primitive p_i a k_i arités ($k_i \geq 0$) insérer une séquence d'abstraction avant ses k_i arguments. La séquence d'abstraction $\lambda x_{j1} \cdots x_{jl_j}$ sera insérer avant le j^{eme} argument ($1 \leq j \leq k$) qui correspond aux descendants $p_{j1}, \cdots, p_{jl_j}$ à la branche j de p_i dans tree (φ).

b éliminer la variable x_0.

c Pour le schéma-terme T (TS(T)) obtenu dans les étapes précédentes calculer les Termes de l'arbre preuve-valide (terms(PT)), $C(T) = T^{0\cdots 0}_{x_1\cdots x_n}$ comme suit, ou $x^0 = x$ (pour toute variable).
- $x_{i\,x_1\ldots x_i\ldots x_n}^{k_1\ldots 0\ldots k_n} = \{\ \bar{x_i}\}$.
- $x_{i\,x_1\ldots x_i\ldots x_n}^{k_1\ldots k_i\ldots k_n} = \{x_i, \acute{x_i}, \acute{\acute{x_i}}, \cdots, x_i^{k_i-1}\ \}$, $k_i \geq 1$.
- $(ST)^{k_1\cdots k_n}_{x_1\cdots x_n} = \{S_iT_j | S_i \in S^{k_1\cdots k_n}_{x_1\cdots x_n}, T_j \in T^{k_1\cdots k_n}_{x_1\cdots x_n}\ \}$.
- $(\lambda x_i.T)^{k_1\cdots k_n}_{x_1\cdots x_n} = \{\ \lambda x_i^{k_i}.T_i | \in T^{k_1\ldots(k_{i+1})\cdots k_n}_{x_1\ldots x_i\ldots x_n}\}$.

Les propositions suivantes sont données et vérifiées par Broda (voir [7] page 13)

- Chaque habitant normal de type φ correspond exactement à un arbre-preuve valide construit à partir de tree(φ), tout arbre preuve-valide construit à partir de tree(φ) a un ensemble fini des habitants normaux de φ, et distincts arbre-preuve valide correspondent aux distincts ensembles disjoints des habitants normaux.
- Soit le type τ et soit l'arbre de preuve-valide PT construit à partir de tree(τ). alors modulo α-conversion,

  C(TS(PT))=Terms (PT). où C est l'ensemble des λ-termes.

Chaque arbre preuve-valide est une représentation d'un seul term-scheme.

On applique l'algorithme terme pour le premier arbre preuve-valide $\begin{matrix} p_0 \\ p_1 \\ p_2 \end{matrix}$ obtenu dans l'exemple 2.2 on aura :
$x_0\lambda x_1.x_1(\lambda x_2.x_2)$(application de (a))
$\lambda x_1.x_1(\lambda x_2.x_2)$ (Elimination de x_0 (application de (b)))
En suite, appliquant (c) pour le terme-schem $\lambda x_1.x_1(\lambda x_2.x_2)$, on obtient un seul habitant en longue normale forme.

$$\{\ \lambda x_1.x_1(\lambda x_2.x_2)\}$$

Pour le deuxième arbre preuve-valide $\begin{matrix} p_0 \\ p_1 \\ p_1 \\ p_2 \end{matrix}$, appliquant l'algorithme terme, on obtient les deux habitants en longue normale forme suivants :

$$\{\lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2^{'}.x_2)), \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2^{'}.x_2^{'}))\}$$

De même pour le troisième arbre preuve-valide $\begin{matrix} p_0 \\ p_1 \\ p_1 \\ p_1 \\ p_2 \end{matrix}$, on obtient

$\{\lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2^{'}.x_1(\lambda x_2^{''}.x_2))), \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2^{'}.x_1(\lambda x_2^{''}.x_2^{'}))),$
$\lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2^{'}.x_1(\lambda x_2^{''}.x_2^{''})))\}, \cdots$

2.3 Grammaire à contexte libre G_τ

Dans cette partie, nous présentons une grammaire à contexte libre pour un type τ.

Cette grammaire génère l'ensemble des term-schemes, qui correspondent exactement aux habitants normaux fermés pour un type τ donné.

Définition 2.6. *Soit τ un type avec les atomes $A_1, \cdots, A_m$ tel que tree(τ) a les parties primitives $P_0, \cdots, P_n$ ou P_0 est la racine de tree(τ).*
Soit G(τ)= (T,N,R,S) est la grammaire à contexte libre avec :

- *L'ensemble des symboles terminaux T =$\{(,), \lambda, ., x_1, \cdots, x_n\}$*
- *L'ensemble des symboles non-terminaux N= $\{S\} \cup \{A_i^p | 1 \leq i \leq m, p \in 2^{\{1,\cdots,n\}}\}$*
- *Symbole initial S.*

et R est le plus petit ensemble satisfait les conditions suivantes :

- *Si $P_0 =$* $\begin{array}{c} | \\ A_s \end{array}$ *a les descendants $P_{i1}, \cdots, P_{it}$ dans l'arbre (τ), alors R a exactement une règle de production :*
$$S \to \lambda x_{i1} \cdots x_{it}.A_s^{\{i_1,\cdots,i_t\}}$$
- *Si le non-terminal A_j^p apparait dans la partie la plus à droite de la règle de production R, alors pour tout $i \in p$ tel que P_i de la forme*
$$\begin{array}{c} | \\ A_j \end{array}$$
on ajoute la règle
$$A_j^p \to x_i$$
- *Si le symbole non-terminal A_j^p apparait dans la partie la plus à droite d'une production R, alors pour tout $i \in P$ tel que P_i de la forme*
$$\begin{array}{ccccc} & & A & & \\ & \swarrow & & \searrow & \\ A_{j1} & & \cdots & & A_{js} \end{array}$$
et pour tous les descendants $P_{j_l^1}, \cdots, P_{j_l^{t_l}}$ de A_{jl} alors ajouter une règle de production dans R de la forme
$$A_j^p \to x_i(\lambda x_{j1}^1 \cdots x_{j1}^{t1}.A_{j1}^{p_1}) \cdots (\lambda x_{js}^1 \cdots x_{js}^{ts}.A_{js}^{ps})$$
où $P_l = P \cup \{j_l^1, \cdots, j_l^{tl}\}$

Exemple 2.3. *Soit le type $\tau = ((A \to A) \to A) \to A$ la grammaire correspondante est :*

$$\begin{array}{lcl} S & \to & \lambda x_1.A^1 \\ A^1 & \to & x_1(\lambda x_2.A^{12}) \\ A^{12} & \to & x_1(\lambda x_2.A^{12}) \\ A^{12} & \to & x_2 \end{array}$$

La grammaire génère l'ensemble infini des term-schemes :
$L(G_\tau) = \{\lambda x_1.x_1(\lambda x_2.x_2), \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2.x_2)), \lambda x_1.x_1(\lambda x_2.x_1(\lambda x_2.x_1(\lambda x_2.x_2))), \cdots\}$

Proposition 2.1. *(voir [9] page 9) Pour le type τ la grammaire G_τ décrit les habitants normaux $Nhabs(\tau)$ dans le sens :*

$$Nhabs\ (\tau) = \{M | \exists T \in L(G_\tau) \exists N \in C(T))\ tel\ que\ N \rightarrow_\eta M\}$$

La définition de G_τ est une description des possibilités de construction des arbres preuves valides de tree(τ), et les règles de productions pour les symboles non-terminaux $A^{i_1,\cdots,i_n}$ décrivant tous les chemins possibles pour décrire les occurrences de A avec d'autres occurrences de A dans la partie primitive P_i de tree(τ).

2.4 Le schéma général d'une grammaire pour une structure de type donné

Soit le type τ on représente la structure de type par le remplacement des occurrences des atomes par des boites indexées comme le montre l'exemple suivant :
$\tau = ((A \rightarrow B) \rightarrow A \rightarrow B) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow B)$ ce type sera représenté par la structure suivante :
$\ominus = ((\square_1 \rightarrow \square_2) \rightarrow \square_3 \rightarrow \square_4) \rightarrow (\square_5 \rightarrow \square_6) \rightarrow \square_7 \rightarrow \square_8$

Définition 2.7. *Soit $\ominus$ une structure de type avec les boites indexées $\square_1, \cdots, \square_n$ avec le tail $\square_n$ ($\square_n$ est la racine dans tree ($\ominus$)), et G ($\ominus$)= (T,N,R,S) une grammaire à contexte libre avec :*

- *L'ensemble des symboles terminaux T ={(,),λ,$x_1, \cdots, x_n$}.*
- *L'ensemble des symboles non-terminaux N=$\{S\} \cup \{\square_i^p | 1 \leq i \leq m, p \in 2^{\{1,\cdots,n\}}\}$.*
- *L'axiome S.*
  Soit R le plus petit ensemble satisfait les conditions suivantes :
  - *Si la racine* $\begin{array}{c}|\\ \square_i\end{array}$ *a les parties primitives descendantes dans l'arbre (τ) les racines $\square_{i1} \cdots \square_{it}$ respectivement , alors R a exactement une règle de production de la forme : $S \rightarrow \lambda x_{i_1} \cdots x_{i_t}.\square_s^{i_1,\cdots,i_t}$*
  - *A chaque apparence du non-terminal $\square_j^p$ dans la partie la plus à droite d'une règle de production R, alors pour chaque $i \in p$ tel que P_i de la forme* $\begin{array}{c}|\\ \square_i.\end{array}$ *, ajouter la règle $\square_j^p \rightarrow x_i$*
  - *Pour chaque symbole non-terminal $\square_j^p$ apparait dans la partie la plus à droite d'une règle de production R, et pour tout $i \in P$ tel que P_i de la forme* $\begin{array}{ccc} & \square & \\ \swarrow & & \searrow \\ \square_{i1} & \cdots & \square_{is}\end{array}$ *et si $\square_{il}$ possède les descendants $\square_{i_l^1}, \cdots \square_{i_l^{tl}}$, alors ajouter la règle de production :*
    $\square_i^p \rightarrow x_i(\lambda x_{i_1^1} \cdots x_{i_1^{t_1}}.\square_{i_1}^{p_1}) \cdots (\lambda x_{i_s^1} \cdots x_{i_s^{t_s}}.\square_{i_s}^{p_s})$.
    Où $P_l = P \cup \{i_l^1, \cdots, i_l^{t_l}\}$

Exemple 2.4. *:Soit* $\ominus = (T, N, R, S)$ *avec*
- $T = \{(,), \lambda, ., x_1, \cdots, x_n\}$,
- $N = \{S\} \cup \{\Box_i^p | 1 \leq i \leq 8, P \in 2^{\{1,\cdots,8\}}\}$
- *Symbole initial* S

Les règles de production R *sont données comme suit :*

S	$\rightarrow \lambda x_4 x_6 x_7 . \Box_8^{467}$	$\Box_1^{1467}$	$\rightarrow x_1$
$\Box_4^{467}$	$\rightarrow x_4(\lambda x_1 . \Box_2^{1467})(\Box_3^{467})$	$\Box_4^{1467}$	$\rightarrow x_4(\lambda x_1 . \Box_2^{1467})(\Box_3^{1467})$
$\Box_6^{467}$	$\rightarrow x_6(\Box_5^{467})$	$\Box_6^{1467}$	$\rightarrow x_6(\Box_5^{1467})$
$\Box_7^{467}$	$\rightarrow x_7$	$\Box_7^{1467}$	$\rightarrow x_7$

La propriété suivante est vérifiée par Broda (voir [9] page 12) :
Soit le type τ avec la structure $\ominus$ c.à.d. $\tau = [A_1/\Box_1, \cdots A_n/\Box_n]$ (ou les A_i ne sont pas nécessairement distincts), soit $G_\tau/\ominus$ une grammaire à contexte libre obtenue à partir de $G_\ominus$ par la substitution de toutes les occurrences de $\Box_1, \cdots, \Box_n$ respectivement par $A_1, \cdots A_n$, alors modulo α-conversion :

$$\begin{aligned} Nhabs(\tau) &= \{M | \exists T \in L(G_\tau) \exists N \in C(T) \text{ tel que : } N \rightarrow_\eta M\} \\ &= \{M | \exists T \in L(G_\tau/\ominus) \exists N \in C(T) \text{ tel que : } N \rightarrow_\eta M\} \end{aligned}$$

3 La technique de plus petit point fixe (fixpoint) pour le comptage des termes dans le lambda calcul (Zaionc)

Dans cette section nous présentons l'algorithme de Zaionc (voir [13]) qui permet de compter les habitants d'un type donné. Le principe de cet algorithme nécessite dans un premier temps de chercher le polynôme d'un type puis le calcul du plus petit point fixe de ce polynôme.
Zaionc a montré qu'il y'a une vraie correspondance entre le problème de comptage des termes en forme normale longue et la recherche de plus petit point fixe d'un polynôme dans un treillis complet (voir [13, 14, 15, 16, 17, 18]).

Notation 3.1. *L'ensemble des types sont définis comme suit :*

1. *Les variables de type (atomes) sont notées par les lettres {A,B,...}.*
2. *Chaque type τ a la forme suivante $\bigotimes_{j=1}^{n} \tau_j \to \mu$ où μ est une variable de type.*
3. *Si τ est un type de la forme $\bigotimes_{j=1}^{n} \tau_j \to \mu$ alors les τ_i $(i \leq n)$ sont appelés les composantes de τ notées $\tau[i]$.*
4. *Pour chaque type τ on définit $rank(\tau)$ et $arg(\tau)$ tel que :*
   si $\tau = A$ (atome) alors $arg(\tau) = rank(\tau) = 0$ et $arg(\bigotimes_{j=1}^{n} \tau_j \to \mu) = n$ et $rank(\bigotimes_{j=1}^{n} \tau_j \to \mu) = 1 + Max_{i=1\cdots n} rank(\tau[i])$.
5. *On définit inductivement les types $\tau[i_1, \cdots, i_k]$ par $(\tau[i_1, \cdots, i_{k-1}])[i_k]$ pour $1 \leq i_k \leq arg([i_1, \cdots, i_{k-1}])$.*
6. *Les deux atomes τ et μ sont dit équivalents (noté par $\tau \sim \mu$) si $\tau = \mu$ cette relation est étendue pour tous les types par les règles suivantes : $(\tau \to \nu) \sim \mu$ ssi $\nu \sim \mu$ et $\tau \sim (\mu \to \nu)$ ssi $\tau \sim \nu$.*
7. *La complexité π pour les termes fermés est définie comme suit :*
   Si $T = \lambda x_1 \cdots x_n.x_i$ alors $\pi(T) = 1$ est un terme fermé en forme normale longue et
   Si $T = \lambda x_1 \cdots x_n.x_i T_1 \cdots T_k$ alors $\pi(T) = 1 + Max_{j=1\cdots k}(\pi(\lambda x_1 \cdots x_n.T_j))$.
8. *$|\tau|$: est le nombre de termes fermés T en forme normale longue de type τ.*
9. *$|\tau|_\alpha$: est le nombre de termes fermés T en forme normale longue de complexité $\pi(T) \leq \alpha$.*

Définition 3.1. *Le type τ est recouvert par μ si $\tau \sim \mu$ et s'il y a une surjection $f : \{1, \cdots, arg(\tau)\} \to \{1, \cdots, arg(\mu)\}$ tel que $\tau[i] = \mu[f(i)]$ pour tout $i \leq arg(\tau)$.*

Définition 3.2. *le type τ est un prolongement direct d'un type μ si $\tau = \bigotimes_{j=1}^{n} \mu[j] \to \mu[i,p]$ pour tout $i \leq arg(\mu)$ et $p \leq arg([i])$ où $n = arg(\mu)$ et $\mu[i] \sim \mu$ la relation de prolongation est transitive et réflexive.*

Définition 3.3. *Le type τ est une extension d'un type μ s'il y a une séquence finie des types $\tau_1, \cdots, \tau_n$ pour $n \geq 1$ tel que $\tau_1 = \mu$ et $\tau_n = \tau$ et si les conditions suivantes sont satisfaites :*

1. *τ_i est un prolongement direct de τ_{i-1} pour tout $2 \leq i \leq n$.*
2. *τ_i n'est pas recouvert par aucun τ_k pour $k \leq i$.*

Théorème 3.1. *Chaque type admit un nombre fini d'extension (preuve voir[13])*

Définition 3.4. *L'arbre d'extension d'un type τ est un arbre des branches finis tel que chaque noeud est étiqueté par un type, l'arbre d'extension est construit comme suit :*

1. *La racine de l'arbre est τ.*
2. *Si le noeud courant μ est une extension de τ et μ a des prolongements directs $\mu_1, ..., \mu_n$ alors dessiner un arc de μ vers chaque μ_i $(1 \leq i \leq n)$.*
3. *Si le noeud courant μ est une extension de τ et qu'il n 'a pas des prolongements directs alors μ n'a pas de successeurs dans l'arbre, tel que μ est un noeud terminal dans l'arbre.*
4. *Si le noeud courant μ n'est pas une extension de τ (μ est recouvert par l'un de ces prédécesseurs), alors μ n'a pas de successeur dans l'arbre, tel que μ est un noeud terminal.*

Définition 3.5.
- *On définit l'ensemble $\bar{N} = N \cup \{\omega\}$ tel que $i \leq \omega$ pour tout $i \in N$.*
  On définit aussi :
  L'addition $a + \omega = \omega$.
  La multiplication $0.\omega = 0$ et $a.\omega = \omega \qquad (a \neq 0)$
  L'ensemble $(\bar{N}^k, \leq)$ est ordonnés dans N tel que :
  $(a_1, ..., a_k) \leq (b_1, ..., b_k)$ si $\forall i \leq k \quad a_i \leq b_i$
- *$\bar{N}^K$ est un treillis complet, $\bar{N}^K$ a une borne inférieure et une borne supérieure.*
- *La fonction $f : \bar{N}^n \to \bar{N}^k$ est appelée un système des polynômes (ou un polynôme) si $f = (f_1, ..., f_k)$ et toute $f_i : \bar{N}^n \to \bar{N}$ est une composition de l'addition de multiplication et des fonctions constantes, toutes les fonctions sont monotones dans $\bar{N}^k$.*
- *Pour tout polynôme $f : \bar{N}^k \to \bar{N}^k$ $(i \in N)$ on définit le polynôme $f^i : \bar{N}^k \to \bar{N}^k$ par $f^{i+1} = f \circ f^i$ et $f^1 = f$.*

– *On dit que* $x_0 \in \bar{N}^k$ *est le plus petit point fixe de* $f : \bar{N}^k \to \bar{N}^k$ *si* x_0 *est un point fixe tel que* $f(x_0) = x_0$ *et pour tout point fixe* x *de* f *on a* $x_0 \sqsubseteq x$.

Le théorème suivant est issu du lemme Knaster-tarski :

Théorème 3.2. *Soit* $f : \bar{N}^k \to \bar{N}^k$ *un polynôme alors* $\cup_{i=0}^{\infty} f^i(\underbrace{0, ...0}_{k\ fois})$ *est le plus petit point fixe de f.*

3.1 Algorithme de plus petit point fixe

Entrée : $f : \bar{N}^k \to \bar{N}^k$.
Sortie : $x_0 \in \bar{N}^k$
Spécification : x_0 est le plus petit fixe point de f.
k :la dimension du problème de la fonction f. l'algorithme est récursif sur k.
Cas 1 : On résout le problème pour k = 1
chaque polynôme $f : \bar{N} \to \bar{N}$ peut être transformé sous la forme $f(x) = \sum_{i=1}^{p} \alpha_i x^i + \alpha_0$ $(\alpha_i \in \bar{N})$, le plus petit point fixe de f est $\alpha_0 + \alpha_0 \alpha \omega$ où $\alpha = \sum_{i=1}^{p} \alpha_i$.
Cas 2 : On résout le problème pour une dimension de k + 1
soit $(f, F) : \bar{N} \times \bar{N}^k \to \bar{N}^{k+1}$
soit le polynôme $f : \bar{N} \times \bar{N}^k \to \bar{N}$ et soit le système polynomial $F : \bar{N} \times \bar{N}^k \to \bar{N}^k$
le polynôme f peut être transformé : $f(x, \overrightarrow{y}) = \sum_{i=1}^{p} \alpha_i(\overrightarrow{y}) x^i + \alpha_0(\overrightarrow{y})$ tel que $\alpha(\overrightarrow{y}) = \sum_{i=1}^{p} \alpha_i(\overrightarrow{y})$ on construit le polynôme $g : \bar{N}^k \to \bar{N}$ comme suit :

$$g(\overrightarrow{y}) = \alpha_0(\overrightarrow{y}) + \alpha_0(\overrightarrow{y})\alpha(\overrightarrow{y})\omega$$

on définit $F' : \bar{N}^k \to \bar{N}^k$ par la formule :

$$F'(\overrightarrow{y}) = F(g(\overrightarrow{y}), \overrightarrow{y})$$

On suppose qu'il y a une procédure de résolution du problème pour une dimension k
Soit $\overrightarrow{y_0}$ le plus petit point fixe de $\overrightarrow{F}$, soit $x_0 = g(\overrightarrow{y_0})$, la paire $(x_0, \overrightarrow{y_0})$ est le plus petit point fixe pour ce problème (Voir [13]).

3.2 Le nombre de termes fermés

Lemme 3.1. *Soit le terme fermé* T *de type* $\tau = \bigotimes_{j=1}^{n} \tau_j \to A$, *s'il existe* $i \leq arg(\tau)$ *tel que* $\tau[i] \sim \tau$ *et si* $k = arg(\tau[i]) > 0$ *alors il y'a des termes fermés* $G_1, \cdots, G_k$ *de types* $\bigotimes_{j=1}^{n} \tau_j \to \tau[i,1], \cdots, \bigotimes_{j=1}^{n} \tau_j \to \tau[i,k]$ *respectivement tel que :*

$$T = \lambda x_1 \cdots x_n . x_i (G_1 x_1 \cdots x_n) \cdots (G_k x_1 \cdots x_n)$$

Voir preuve [13] page 6.

Définition 3.6. *Pour tout type τ on définit :*

1. $S_\tau = \{i \leq arg(\tau) : \tau \sim \tau[i] \text{ et } rank(\tau[i] \geq 1)\}$
   et

2. $T_\tau = \{i \leq arg(\tau) : \tau \sim \tau[i] \text{ et } rank(\tau[i] = 0)\}$

Lemme 3.2. *Si $S_\tau \cup T_\tau = 0$ alors $|\tau| = 0$*

Théorème 3.3. *Si $S_\tau \cup T_\tau \neq 0$ alors*

1. $|\tau| = |T_\tau| + \sum_{i \in S_\tau} \prod_{p=1}^{arg(\tau[i])} | \bigotimes_{j=1}^{n} \tau[j] \to \tau[i,p]|$
2. $|\tau|_\alpha = |T_\tau| + \sum_{i \in S_\tau} \prod_{p=1}^{arg(\tau[i])} | \bigotimes_{j=1}^{n} \tau[j] \to \tau[i,p]|_{\alpha-1}$ *tel que* $\alpha > 0$.

Remarque 3.1. *le nombre total des termes de type τ de tête x_i est donné par le produit $\prod_{p=1}^{k} | \bigotimes_{j=1}^{n} \tau[j] \to [i,p]|$.*

Lemme 3.3. *Soit le type τ et soient $\tau_1, \ldots, \tau_n$ des prolongements de τ, Soient $x = |\tau|$ et $x_1 = |\tau_1|, \ldots, x_n = |\tau_n|$ alors il y' a un polynôme $f : \bar{N}^n \to N$ tel que $x = f(x_1, \ldots, x_n)$ (Voir preuve [13] page 7).*

Exemple 3.1. *Soit x, y, $z \in \bar{N}$ le nombre des termes fermés des types suivants respectivement :*
$(B, C \to C), C, C, (B, C \to B), (C, C \to A), (A, B \to A) \to A$
$(B, C \to C), C, C, (B, C \to B), (C, C \to A), (A, B \to A) \to B$
$(B, C \to C), C, C, (B, C \to B), (C, C \to A), (A, B \to A) \to C$

On cherche d'abord :
$S_\tau = \{i \leq arg(\tau) : \tau \sim \tau[i] \text{ et } rank(\tau[i] \geq 1)\}$
et
$T_\tau = \{i \leq arg(\tau) : \tau \sim \tau[i] \text{ et } rank(\tau[i] = 0)\}$
puis on applique le Théorème 3.3
$|\tau| = |T_\tau| + \sum_{i \in S_\tau} \prod_{p=1}^{arg(\tau[i])} | \bigotimes_{j=1}^{n} \tau[j] \to \tau[i,p]|$

- $rank(Arg(\tau[i])) \geq 1 \{i = 5, 6$ *tel que tail* $\tau[5] \sim$ *tail* $\tau[6] \sim A\}$ *alors* $S_\tau = \{5, 6\}$.
- $T_\tau = \Phi$ *pas d'arguments*

alors
$x = 0 + \sum_{i\in\{5,6\}} \prod_{p=1}^{arg(\tau[i])} |\bigotimes_{j=1}^{n} \tau[j] \to \tau[i,p]|$
tel que $x = \prod_{p=1}^{arg(\tau[5])} |\bigotimes_{j=1}^{n} \tau[j] \to \tau[5,p]| + \prod_{p=1}^{arg(\tau[6])} |\bigotimes_{j=1}^{n} \tau[j] \to \tau[6,p]|$
$x = |\bigotimes_{j=1}^{n} \tau[j] \to \tau[5,1]| * |\bigotimes_{j=1}^{n} \tau[j] \to \tau[5,2]| + \bigotimes_{j=1}^{n} \tau[j] \to \tau[6,1]| *$
$|\bigotimes_{j=1}^{n} \tau[j] \to \tau[6,2]|$
$|\bigotimes_{j=1}^{n} \tau[j] \to \tau[5,1]| = z$
$|\bigotimes_{j=1}^{n} \tau[j] \to \tau[5,2]| = z$
$|\bigotimes_{j=1}^{n} \tau[j] \to \tau[6,1]| = x$
$|\bigotimes_{j=1}^{h} \tau[j] \to \tau[6,2]| = y$ *alors*
$x = z^2 + xy$ *de la même manière on trouve*
$y = yz$
$z = 2 + yz$ *(rank(arg[2]) = rank(arg[3]) = 0) et (rank(arg [1]) ≥ 1).*

Lemme 3.4. *Soit $i \geq 1$ les types $\tau \to \mu$ et $\tau^i \to \mu$ ont la même taille (voir [13] page 7).*

Lemme 3.5. $\tau = \bigotimes_{j=1}^{n} \tau[j] \to A$ *et soit* $f : \{1, \cdots, n\} \to \{1, \cdots, n\}$ *est une permutation si* $\mu = \bigotimes_{j=1}^{n} \tau[f(j)] \to A$ *alors* $|\tau| = |\mu|$ *(voir [13]).*

Lemme 3.6. *Si le type τ est recouvert par μ alors τ et μ ont la même taille.*

Lemme 3.7. *Si le type τ est recouvert par μ alors $|\mu|_\alpha \leq |\tau|_\alpha$ $(\alpha \in N)$ et si $|\mu|_\alpha = 0$ alors $|\tau|_\alpha = 0$.*

3.3 Algorithme de construction polynomiale

Entrée : type τ
Sortie : Une liste de types $(\tau_1, \cdots, \tau_n)$ et le polynôme $F : \bar{N}^n \to \bar{N}^n$

Soit l'arbre fini d'extension de type τ (théorème 3.1)
Soient les types $\tau_1, \cdots, \tau_n$ dans l'arbre.

1. Mettre $\tau_1 = \tau$.
   Pour tout type τ_i on introduit une nouvelle variable x_i qui sera notée $x_i = |\tau_i|$.
   Soient toutes les extensions $\tau_1, \cdots, \tau_k$ de τ dans l'arbre, pour cela on va construire K polynômes $f_1, \cdots, f_k$ un pour chaque extension de type τ.
2. Si le noeud courant $\mu = \tau_i$ dans l'arbre est une extension de τ et μ a les prolongements directs $\mu_1, \cdots, \mu_p$, alors suivant le lemme 3.3 il y'a

un polynôme dit f_i tel que $f_i : \bar{N}^p \to \bar{N}$ (f_i peut être vue comme une fonction des arguments $x_1, \cdots, x_n$) tel que

$$x_i \quad = \quad f_i(x_1 \cdots x_n) \qquad (1)$$

3. Soit τ_i un noeud terminal dans l'arbre, τ_i est une extension de τ et τ_i ne possède pas des prolongements directs alors $s = |T_{\tau_i}|$ (Voir définition 3.6), mettre

$$x_i \quad = \quad s \qquad (2)$$

4. Si τ_i un noeud terminal tel que τ_i n'est pas une extension de τ (pour $i > k$), alors il existe un prédécesseur de τ_i appelé τ_s tel que τ_i est recouvert par τ_s et le type τ_s est une extension de τ, mettre

$$x_i \quad = \quad x_s \qquad (3)$$

Maintenant on a exactement n équations polynomiales entre variables
$x_1 = f_1(x_1 \cdots x_n)$
$\vdots$
$x_k = f_k(x_1 \cdots x_n)$
$x_{k+1} = x_{s_1}$ où τ_{k+1} est recouvert par $\tau_{s_1} \in \{\tau_1, \cdots, \tau_k\}$
$\vdots$
$x_n = x_{s_{n-k}}$ où τ_n est recouvert par $\tau_{s_{n-k}} \in \{\tau_1, \cdots, \tau_k\}$
Utilisant (n - k) équations, on peut remplacer $x_{k+1}, \cdots, x_n$ qui ne sont pas des extensions de τ par d'autres variables qui sont des extensions de τ, aussi on suppose pour tout i la variable x_i a une apparence dans le polynôme f_i , sinon on doit réduire le nombre d'équations par la substitution polynomiale $f_i\,(x_1 \cdots x_n)$ pour chaque x_i, on doit réduire le chemin pour toutes les équations de le forme $x_i = s$.
Si $(x_1, \cdots, x_n)$ est le plus petit point fixe de F alors

$$x_i = |\tau_i| \ \textit{pour tout}\ i \leq n$$

Suite de l'exemple précédent, le problème est de trouver des termes en forme normale pour le type $(B \to C \to C) \to C \to C \to (B \to C \to B) \to (C \to C \to A) \to (A \to B \to A) \to A$, on applique l'algorithme de construction polynomiale.
On introduit quelques types qui dénotent le nombre de termes en formes normales :
$x = |(B \to C \to C) \to C \to C \to (B \to C \to B) \to (C \to C \to A) \to (A \to$

$B \to A) \to A|$
$y = |(B \to C \to C) \to C \to C \to (B \to C \to B) \to (C \to C \to A) \to (A \to B \to A) \to B|$
$z = |(B \to C \to C) \to C \to C \to (B \to C \to B) \to (C \to C \to A) \to (A \to B \to A) \to C|$
Le résultat de l'algorithme de construction polynomiale sur les types est les équations suivantes :

$$\begin{aligned} x &= xy + z^2 \\ y &= yz \\ z &= yz + 2 \end{aligned}$$

On applique l'algorithme de plus petit point fixe on obtient le plus petit point fixe des équations précédentes

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix}$$

On applique le théorème 3 (knaster tarski)

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix} \xrightarrow{f}$$

il y'a exactement 4 termes fermés de type
$(B \to C \to C) \to C \to C \to (B \to C \to B) \to (C \to C \to A) \to (A \to B \to A) \to A$

Exemple 3.2. *(voir [13] page 11) Dans cet exemple on construit le système polynomial et on accorde le nombre de preuves (termes) pour cette formule (type) qui n'est pas prouvable nommé Pierce law* $((A \supset B) \supset A) \supset A$*, le symbole* $\supset$ *représente l'application classique* $\to$ *et le symbole* $\perp$ *le NOT logique.*
Alors $\sim A$ *est représenté par* $A \to \perp$ *et* $A \supset B$ *par* $A \to (\sim B \to \perp)$.
après la transformation de Priece law on aura la forme suivante :
$((A \to (B \to \perp) \to \perp) \to (A \to \perp) \to \perp) \to (A \to \perp) \to \perp$
Pour la performance de l'algorithme de construction polynomiale on introduit quelques types et des variables qui dénotent le nombre des formes normales pour ces types.
Soit :
$x = |((A \to (B \to \perp) \to \perp) \to (A \to \perp) \to \perp) \to (A \to \perp) \to \perp|$
$y = |((A \to (B \to \perp) \to \perp) \to (A \to \perp) \to \perp) \to (A \to \perp) \to A \to (B \to \perp) \to \perp|$
$z = |((A \to (B \to \perp) \to \perp) \to (A \to \perp) \to \perp) \to (A \to \perp) \to A \to \perp|$

$u = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A|$
$s = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to A \to (B \to \bot)|$
$t = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to A \to \bot\ |$
$w = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to A|$
$v = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to (B \to \bot) \to A \to (B \to \bot) \to \bot\ |$
$p = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to (B \to \bot) \to A \to \bot\ |$
$q = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to (B \to \bot) \to A|$
$r = |((A \to (B \to \bot) \to \bot) \to (A \to \bot) \to \bot) \to (A \to \bot) \to A \to (B \to \bot) \to B|$

Accorder l'algorithme construction polynomial on aura les équations polynomiales suivantes :
$x = yz + u$
$u = 0$
$y = vp + q + r$
$r = 0$
$q = 1$
$z = st + w$
$w = 1$

$v = y$ *v est recouvert par y*
$p = y$ *p est recouvert par y*
$s = y$ *s est recouvert par y*
$t = z$ *t est recouvert par z*
par substitution, le système peut être se réduit :
$x = yz$
$y = y^2 + 1$
$z = yz + 1$
Le résultat de l'algorithme de plus petit fixe point est la solution minimale pour le système $y = w, z = w$ *par conséquent* $x = w$*, il y'a un nombre infini de termes en formes normales (preuve normale) de type (formule)* $((A \supset B) \supset A) \supset A$.
Utilisant le théorème Knaster tarski on aura une diverge conséquence des tuples.

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 4 \\ 5 \\ 5 \end{pmatrix} \xrightarrow{f} \cdots$$

4 Grammaire de génération des habitants d'un type (Takahashi, Akama)

Dans [24] Takahashi et Akama ont présenté une grammaire à contexte libre pour générer l'ensemble des habitants M en forme β-normale et en forme normale longue pour un type A et un contexte Γ.

Définition 4.1. *(Grammaire à contexte libre G) Une grammaire à contexte libre est un quadruplet définit comme suit :*
G = (T,N,R,σ) tel que :
- *T :Alphabet de la grammaire ou symbole terminaux.*
- *N :Ensembles des symboles non-terminaux.*
- *R :Le sous-ensemble fini des règles de production* $\{\xi \rightsquigarrow w | \xi \in N, w \in (T \cup N)^*\}$
- *σ :Un élément de N appelé axiome.*

Définition 4.2. *(Langage Contexte libre) Le langage engendré par une grammaire à contexte libre G est défini comme suit :*
$L(G)=\{\tau \in T^* | \sigma \overset{*}{\rightsquigarrow} \tau\}$ *tel que :*
$\overset{*}{\rightsquigarrow}$ *est une fermeture transitive réflexive.*
$(w_1 \overset{*}{\rightsquigarrow} w_k, \exists k \in n | w_1 \overset{k}{\rightsquigarrow} w_k \equiv w_1 \rightsquigarrow w_2 \rightsquigarrow \cdots \rightsquigarrow w_k)$.

Définition 4.3. *(forme β-normale) $B_t(\Gamma, A)$ est une forme β-normale pour un contexte Γ et un type A et on définit :*

$$\begin{aligned} B_t(\Gamma, A) \quad = & \{\lambda x_{A_1} x_{A_2} \cdots x_{A_p}.x_B M_1 M_2 \cdots M_q | p, q \geq 0, \\ & \exists A', B_1, \cdots, B_q \text{ tel que} \\ & A \equiv A_1 \to A_2 \to \cdots \to A_p \to A', \\ & B \equiv B_1 \to B_2 \to \cdots \to B_q \to A' \in type(\Gamma'), \\ & M_j \in B_t(\Gamma', B_j)(j = 1, 2, \cdots q) \\ & \text{où } \Gamma' = \Gamma \cup \{x_1 : A_1, x_2 : A_2, \cdots, x_p : A_p\}\} \end{aligned}$$

Définition 4.4. *(Forme normale longue)*
La formule $\Gamma \vdash_t \lambda x_{A_1} x_{A_2} \cdots x_{A_m}.x_B M_1 M_2 \cdots M_n : A$ est dite en forme normale longue pour tout $a, B_1, \cdots, B_n$,
$A \equiv A_1 \to A_2 \to \cdots \to A_m \to a$,
$B \equiv B_1 \to B_2 \to \cdots \to B_n \to a \in type(\Gamma')$, *et*
$\Gamma' \vdash_t M_j : B_j$ *est en forme normale longue* $(j = 1, \cdots, n)$
où $\Gamma' = \Gamma \cup \{x_{A_1} : A_1, x_{A_2} : A_2 \cdots x_{A_m} : A_m\}$ *on écrit :*
$L_t(\Gamma, A) = \{M | \Gamma \vdash_t M : A$ *est en forme normale longue }.*

Rappels :
Soit le contexte $\Gamma = \{x_1 : A_1, x_2 : A_2, \cdots, x_n : A_n\}$ alors :
$Sujet(\Gamma) = \{x_1, x_2, \cdots, x_n\}$ et
$Type(\Gamma) = \{A_1, A_2, \cdots, A_n\}$

4.1 Grammaire de génération des habitants d'un type

Dans la suite nous allons présenter une grammaire $G(T, N, R, <\Gamma, A>)$ qui engendre les habitants d'un type A en forme normale longue $(L_t(\Gamma, A))$ dans un contexte Γ.

Théorème 4.1. *Soit le contexte* Γ *et le type* A *:*

- $\{\lambda, ., (,)\} \subseteq T$.
- $\langle \Gamma, A\rangle \in N$
- *Pour tout* $\langle \Delta, C\rangle \in N$ *ajouter à* R *la règle de production suivante :*
  $\langle \Delta, C\rangle \rightsquigarrow (\lambda x_{C_1} x_{C_2} \cdots x_{C_m}.x_B \langle \Delta', B_1\rangle \langle \Delta', B_2\rangle \cdots \langle \Delta', B_n\rangle)$
  $\Delta' = \Delta \cup \{x_{C_1} : C_1, \cdots x_{C_m} : C_m\}$,
  $\exists$ *c tel que :*

$$\begin{aligned} C &= C_1 \to C_2 \to \cdots \to C_m \to c. \\ B &= B_1 \to B_2 \to \cdots \to B_n \to c \in type(\Delta'). \\ x_{C_i} &\in T \quad (i = 1, \cdots m). \\ \langle \Delta', B_j\rangle &\in N \quad (j = 1, \cdots, n) \end{aligned}$$

alors l'ensemble $L_t(\Gamma, A)$ *est généré par la grammaire à contexte libre* $G_t(\Gamma, A) = (T, N, R, \langle \Gamma, A\rangle)$.

Exemple 4.1. *Soit* $\Gamma = \emptyset$ *et* $A = A_1 \to A_2 \to A_3 \to b$ *où* $A_1 = A_2 \to A_3 \to b$ $A_2 = A_3 \to b$, $A_3 = a$ *on applique le théorème 4.1 on obtient la grammaire à contexte libre G = (T,N,R,* $\langle \emptyset, A\rangle$*) pour générer l'ensemble* $L_t(\emptyset, A)$ *où :*
$T = \{x_{A_1}, x_{A_2}, x_{A_3}, \lambda, ., (,)\}$
$N = \{\langle \emptyset, A\rangle, \langle \Delta, A_2\rangle, \langle \Delta, A_3\rangle\}$

$$\begin{aligned} R = \{ \quad \langle \emptyset, A\rangle &\rightsquigarrow (\lambda x_{A_1} x_{A_2} x_{A_3}.x_{A_1} \langle \Delta, A_2\rangle \langle \Delta, A_3\rangle) \\ \langle \emptyset, A\rangle &\rightsquigarrow (\lambda x_{A_1} x_{A_2} x_{A_3}.x_{A_2} \langle \Delta, A_3\rangle) \\ \langle \Delta, A_2\rangle &\rightsquigarrow (\lambda x_{A_3}.x_{A_1} \langle \Delta, A_2\rangle \langle \Delta, A_3\rangle) \\ \langle \Delta, A_2\rangle &\rightsquigarrow (\lambda x_{A_3}.x_{A_2} \langle \Delta, A_3\rangle) \\ \langle \Delta, A_3\rangle &\rightsquigarrow (x_{A_3})\} \end{aligned}$$

Avec $\Delta = \{x_{A_1} : A_1, x_{A_2} : A_2, x_{A_3} : A_3\}$.

La génération d'une grammaire à contexte libre pour $B_t(\Gamma, A)$ est similaire à $L_t(\Gamma, A)$.
La grammaire pour générer les habitants en forme β-normale ($B_t(\Gamma, A)$) est donnée par le théorème suivant :

Théorème 4.2. *Soit le contexte* Γ *et le type* A *:*

- $\{\lambda, ., (,)\} \subseteq T$.
- $\langle \Gamma, A \rangle \in N$
- *Pour tout* $\langle \Delta, C \rangle \in N$ *ajouter à* R *la règle de production suivante :* $\langle \Delta, C \rangle \rightsquigarrow (\lambda x_{C_1} x_{C_2} \cdots x_{C_p}.x_B \langle \Delta', B_1 \rangle \langle \Delta', B_2 \rangle \cdots \langle \Delta', B_q \rangle) | \ p, q \geq, \exists C' tel$ *que :*

$$\begin{aligned} C &= C_1 \to C_2 \to \cdots \to C_p \to C' \\ B &= B_1 \to B_2 \to \cdots \to B_q \to C' \in type(\Delta') \\ \Delta' &= \Delta \cup \{x_{C_1} : C_1, \cdots x_{C_p} : C_p\} \end{aligned}$$

$\langle \Delta', B_j \rangle \in N (j = 1, \cdots, q)$.

Alors les ensembles T,N,R sont finis et $B_t(\Gamma, A)$ *est généré par une grammaire à contexte libre* $G_t(\Gamma, A) = (T, N, R, \langle \Gamma \rangle, A)$

$\Gamma \vdash_t M : A$ est en forme β-normale ssi $\exists \Gamma \vdash_t M' : A$ en forme normale longue tel que $M' \twoheadrightarrow_\eta M$.

Exemple 4.2. *On applique le théorème 4.2 pour l'exemple 4.1* *(*$\Gamma = \emptyset$ *et* $A = ((a \to b) \to a \to b) \to (a \to b) \to a \to b$*), on obtient la grammaire suivante :*

$T = \{x_{A_1}, x_{A_2}, x_{A_3}, \lambda, ., (,)\}$

$N = \{\langle \emptyset, A \rangle, \langle \Delta, A_2 \rangle, \langle \Delta, A_3 \rangle\}$

$$\begin{aligned} R = \{ \quad \langle \emptyset, A \rangle &\rightsquigarrow \lambda x_{A_1}.x_{A_1} \\ \langle \emptyset, A \rangle &\rightsquigarrow \lambda x_{A_1} x_{A_2}.x_{A_2} \\ \langle \emptyset, A \rangle &\rightsquigarrow (\lambda x_{A_1} x_{A_2} x_{A_3}.x_{A_1} \langle \Delta, A_2 \rangle \langle \Delta, A_3 \rangle) \\ \langle \emptyset, A \rangle &\rightsquigarrow (\lambda x_{A_1} x_{A_2} x_{A_3}.x_{A_2} \langle \Delta, A_3 \rangle) \\ \langle \Delta, A_2 \rangle &\rightsquigarrow (\lambda x_{A_3}.x_{A_1} \langle \Delta, A_2 \rangle \langle \Delta, A_3 \rangle) \\ \langle \Delta, A_2 \rangle &\rightsquigarrow (\lambda x_{A_3}.x_{A_2} \langle \Delta, A_3 \rangle) \\ \langle \Delta, A_2 \rangle &\rightsquigarrow (x_{A_1} \langle \Delta, A_2 \rangle) \\ \langle \Delta, A_2 \rangle &\rightsquigarrow (x_{A_2}) \\ \langle \Delta, A_3 \rangle &\rightsquigarrow (x_{A_3}) \} \end{aligned}$$

Avec $\Delta = \{x_{A_1} : A_1, x_{A_2} : A_2, x_{A_3} : A_3\}$.

Corollaire 4.1. $L_t(\Gamma, A)$ *peut être vide, fini, ou infini pour tout* Γ *et* A*. le même sera vrai pour* $B_t(\Gamma, A)$.

5 Génération aléatoires des termes en forme β-normale des types simples(J.Wang)

Dans [21] Jue Wang a donné un algorithme pour générer d'une manière uniforme aléatoire des termes en forme β-normale d'un type A et d' une taille s donnée. Dans la suite nous présentons cet algorithme après avoir rappelé quelques notations et définitions.

Définition 5.1.

- $size(M) = \begin{cases} 1 & si\ M \equiv x \\ 1 + size(N) + size(P) & si\ M \equiv (NP) \\ 1 + size(N) & si\ M \equiv (\lambda x.N) \end{cases}$

- $args(A) = \begin{cases} A_1 \cup A_2 \cup \cdots \cup A_n \cup args(A_1) \cup \cdots \cup args(A_n) & si\ A = A_1 \to \cdots \to A_n \to a \\ \{\} & si\ A = a \end{cases}$

- *Un environnement* Γ *est un ensemble constitué des types* A_i *et le nombre d'apparition* n *de ces types dans l'environnement* $(A_i : n)$.
  Soit l'environnement $\Gamma_0 = \{a : 1, a \to b : 3, a \to a : 0\}$, *cet environnement indique qu'il y'a une seule variable de type* a, *trois variables de type* $a \to b$ *et aucune variable de type* $a \to a$.
- $typeSet(\Gamma)$ *: est une fonction qui retourne les différents ensembles des types liés à un environnement* Γ *donné.*

  $typeSet(\Gamma_0) = \{a, a \to b, a \to a\}$.

- $typeCnt$ *est une fonction qui retourne le nombre de variables dans* Γ *de type* A

  $typeCnt(\Gamma_0, a) = 1$

- $typeCntInc(\Gamma, A)$ *: est une fonction qui incrémente le nombre de variables de type* A *dans* Γ *par 1.*
  $typeCntInc(\Gamma_0, a)$ *produit le nouveau environnement*

  $\Gamma_1 = \{a : 2, a \to b : 3, a \to a : 0\}$.

Jue Wang [21] a modifié la grammaire(infinie) développée par Takahashi [24] pour obtenir la grammaire présentée ci-dessous, dans cette grammaire il a ajouté les symboles non-terminaux $[\Gamma, A]$ qui dérivent toutes les variables dans l'environnement de type A.

5.1 Grammaire de génération des termes en forme β-normale d'un type A avec l'environnement Γ :

Soit l'environnement Γ et le type A,
La grammaire $G(\Gamma, A) = (T, N, R, <\Gamma, A>)$ génère l'ensemble
$B(\Gamma, A) = \{M | \Gamma \vdash M : A$ en forme β-normale $\}$
T :L'ensemble des symboles terminaux.
N :L'ensemble des symboles non-terminaux.
R :L'ensemble des règles de productions, définit comme suit :

- $\{\lambda, ., .(,)\} \subseteq T$.
- $<\Gamma, A> \in N, [\Gamma, A] \in N$.
- Pour tout $<\Gamma, A> \in N$, l'ensemble des règles de production R :

$$
\begin{array}{lll}
<\Gamma, A> & \rightsquigarrow & [\Gamma, A] \\
 & \rightsquigarrow & x_i^B <\Gamma, B_1> \cdots <\Gamma, B_m> \\
 & & \text{pour } i = 0, \cdots, (typeCnt(\Gamma, B) - 1), \\
 & & \text{pour tout } B \equiv B_1 \to \cdots \to B_m \to A \in typeSet(\Gamma) \\
 & & N \cup <\Gamma, B_j> \text{ pour } j = 1, \cdots, m \\
 & \rightsquigarrow & \lambda x^{A_1}_{typeCnt(\Gamma, A_1)} : A_1. <typeCntInc(\Gamma, A_1), A_2 \to \cdots \to A_n \to a> \\
 & & \text{si } A = A_1 \to A_2 \to \cdots \to A_n \to a, n \geq 1 \\
 & & T \cup x^{A_i}_{typeCnt(\Gamma, A_i)} \text{ pour } i = 0, \cdots, n \\
[\Gamma, A] & \rightsquigarrow & x_j^A, \text{ pour } j = 0 \cdots (numType(\Gamma, A) - 1)
\end{array}
$$

La grammaire génère tous les termes M de type A en forme β-normale par les règles de production de non-terminal $<\Gamma, A>$.
Si la première production $[\Gamma, A]$ est utilisée, alors la grammaire génère toutes les variables de type A dans Γ et les variables sont en forme β-normale.
Si la deuxième règle de production de $x_i^B <\Gamma, B_1> \cdots <\Gamma, B_m>$ est utilisée et si on suppose que $<\Gamma, B_1> \cdots <\Gamma, B_m>$ génèrent m termes $y_1, \cdots, y_m$ de type $B_1, \cdots, B_m$ en forme β-normale, alors le terme résultat est $M = (\cdots (x_i^B y_1) \cdots y_m)$.
Si $A = A_1 \to \cdots \to A_n \to a$ alors la troisième production peut être utilisée pour générer les termes (Voir[21] page 6).

Exemple 5.1. *Cet exemple donne une grammaire pour générer l'ensemble* $B(\Gamma, ((a \to b) \to a \to b) \to (a \to b) \to a \to b)$ *de tous les termes en forme* β*-normale de type* $A = A_1 \to A_2 \to a \to b$ *où* $A_1 = (a \to b) \to a \to b$, $A_2 = a \to b$.

- *L'environnement initial* $\Gamma_0 = \{A_1 : 0, A_2 : 0, a : 0\}$.

- *L'ensemble des terminaux T :*
  λ ,.,(,)
  $x_j^{A_2 \to a \to b}$, *pour* $j = 0 \cdots (typeCnt(\Gamma, A_2 \to a \to b) - 1)$,
  $x_j^{A_2}$ *pour* $j = 0 \cdots (typeCnt(\Gamma, A_2) - 1)$, *et* x_j^a *pour* $j = 0 \cdots (typeCnt(\Gamma, a) - 1)$.

- *L'ensemble des non-terminaux N :*
  $< \Gamma, A >, < \Gamma, A_2 \to a \to b >, < \Gamma, a \to b >, < \Gamma, a \to b >, < \Gamma, b >,$
  $< \Gamma, a >$, $[\Gamma, A_2 \to a \to b], [\Gamma, A_2], [\Gamma, a]$

- *L'ensemble des règles R :*

$< \Gamma, A >$	$\rightsquigarrow$	$\lambda x^{A_1}_{typeCnt(\Gamma,A_1)} : A_1. < typeCntInc(\Gamma, A_1), A_2 \to a \to b >$
$< \Gamma, A_2 \to a \to b >$	$\rightsquigarrow$	$[\Gamma, A_2 \to a \to b]$ $\lambda x^{A_2}_{typeCnt(\Gamma,A_2)} : A_2. < typeCntInc(\Gamma, A_2), a \to b >$
$< \Gamma, a \to b >$	$\rightsquigarrow$	$[\Gamma, a \to b]$ $\lambda x^{a}_{typeCnt(\Gamma,a)} : a. < typeCntInc(\Gamma, a), b >$ $[\Gamma, A_1] < \Gamma, a \to b >$
$< \Gamma, b >$	$\rightsquigarrow$	$[\Gamma, b]$ $x_i^{A_1} < \Gamma, A_2 >< \Gamma, a >$ *pour* $i = 0 \cdots (typeCnt(\Gamma, A_1) - 1)$ $x_i^{A_2} < \Gamma, a >$ *pour* $i = 0 \cdots (typeCnt(\Gamma, A_2) - 1)$
$< \Gamma, a >$	$\rightsquigarrow$	$[\Gamma, a]$
$[\Gamma, A_2 \to a \to b]$	$\rightsquigarrow$	$x_j^{A_2 \to a \to b}$ *pour* $j = 0 \cdots (typeCnt(\Gamma, A_2 \to a \to b) - 1)$
$[\Gamma, A_2]$	$\rightsquigarrow$	$x_j^{A_2}$ *pour* $j = 0 \cdots (typeCnt(\Gamma, A_2) - 1)$
$[\Gamma, a]$	$\rightsquigarrow$	x_j^{a} *pour* $j = 0 \cdots (typeCnt(\Gamma, a) - 1)$
$[\Gamma, b]$	$\rightsquigarrow$	*Aucun.*

5.2 Génération des termes d'un type donné

Dans cette section, nous présentons l'algorithme donné dans [21] pour générer les termes fermés en forme β-normale d'un type A et d'une taille s (size), d'une manière aléatoire uniforme .
Le type et la taille ne sont pas un critère suffisant pour connaitre la forme de

terme à générer, pour cela J.Wang a utilisé les fonctions de comptage pour déterminer la probabilité de chaque forme de terme qui va guider la génération.
Pour compter le nombre de termes d'un type A et d'une taille s, J.Wang a défini les fonctions $countTerm$, $countHeadVarTerm$ et $countHeadVarArgTerms$ d'une manière récursive.
La fonction *count* initialise $countTerm$ avec un environnement Γ vide, un type A et une taille s.
La fonction $countTerm$ retourne le nombre de termes en forme β-normale basée sur la taille s.

- Si s = 0 alors le nombre de terme =0.
- Si s = 1 alors le seul terme d'un type donné est une variable de terme, et on cherche le nombre de variable de termes dans l'environnement.
- Si s >1 on considère deux cas, le type soit une variable de type ou un type composé.
  - ⋆ Si le type = variable de type (atome) alors on appelle la fonction $countHeadVarTerm(A, \Gamma, s)$, cette fonction compte le nombre d'application terme de la forme $((\cdots(xM_1)\cdots M_{k-1})M_k)$ où $x : B_1 \to \cdots \to B_k \to A$ et $M_i : B_i$ pour $1 \leq i \leq k$.
    $countHeadVarTerm$ cherche toutes les variables x_j^B dans Γ qui retourne le type A et pour toute x_j^B on calcul tous les chemins possibles pour générer les termes de tête x_j^B à l'aide de la fonction $countHeadVarArgTerms$.
  - ⋆ Si le type $T = A_1 \to A_2$ (type composé) alors on peux générer soit une abstraction ou une application terme, on ajoute dans les deux cas le nombre de termes.
    Dans le cas d'une abstraction on déclenche la fonction $countTerm(A_2, s-1, typeCntInc(\Gamma, A_1))$.
    Dans le cas d'une application, on doit avoir comme opérateur une variable de terme de type T et on déclenche la fonction $countHeadVarTerm$ pour compter le nombre de ces termes.

La génération des termes se fait par les fonctions $genTerm$, $genVarterm$, $genLamTerm$ et $genAppTerm$ d'une manière récursive. On utilise $gen(A, s)$ pour générer les termes en forme β-normale cette dernière déclenche la fonction $genTerm(\Gamma = 0, s, A)$, on distingue les cas suivants sur la taille s.

- Si s = 0 alors pas de terme.
- Si s = 1 alors on appelle $genVarTerm$ qui retourne une variable de terme de type T choisit dans un environnement Γ d'une manière aléatoire uniforme.

- autrement
  - ⋆ Si type = variable de type (atome) soit A, la seule règle qu'on peut utiliser est $(\rightarrow E)$ et on doit appeler la fonction $genAppTerm$ pour générer les termes application.
    Dans la génération de l'application terme on appelle la fonction $chooseHeadVar$ pour générer le type de la variable tête choisit, avec ces informations utiliser $genVarTerm$ pour générer la variable tête, $genTerm$ pour générer les arguments de termes et construire le terme pour ces pièces.
  - ⋆ Si le type $T = A_1 \rightarrow A_2$ (type composé), on peut générer soit une abstraction ou une application, on appelle la fonction $countTerm$ pour compter le nombre total de termes d'un type T et d'une taille s donné et calculer le nombre total d'abstraction terme d'un type T et d'une taille donné s, si le nombre d'abstraction terme est supérieur à la probabilité correspondante au pourcentage de nombre total de termes alors on choisit la génération d'une abstraction terme, pour cela on génère une variable de type A_1, on appelle la fonction $genTerm(A_2, s-1, typeCntInt(\Gamma, A_1))$ pour générer le corps du terme sinon on choisit la génération d'application terme et on appelle la fonction $genAppTerm$.

Dans les figures (FIGURE 2.1 et FIGURE 2.2) l'auteur a présenté un pseudo-code pour les fonctions de comptage et de génération d'un terme suivant une spécification donnée, l'implémentation est donnée en Ocaml(voir [22] page 41-62).

Fonction auxiliaires partagées entre les fonctions de comptage et de génération

Entrée :type $A = A_1 \to A_2 \cdots \to A_n \to a$, taille s
Sortie :le nombre de termes en forme β-normale d'une taille s et de type A.

$count(A, s) = countTerm(A, 0, s)$
$countTerm(A, \Gamma, s) =$
IF $s = 0$ THEN
RETURN 0
ELSE IF $s = 1$ THEN
RETURN $typeCnt(\Gamma, A)$
ELSE
IF $varType?(A)$ THEN
RETURN $countHeadVarTerm(A, \Gamma, s)$
ELSE
RETURN $countTerm(resType(A), typeCntInc(\Gamma, argtype(A)), s-1)$
$+countHeadVarTerm(A, \Gamma, s)$

$countHeadVarTerm(A, \Gamma, s) =$
$numTerms \leftarrow 0$
FOR each $B = B_1 \to \cdots \to B_m \to A \in validHeadVarTypeSet(A, \Gamma)$
$numTerms \leftarrow numTerms + countHeadVarArgTerms(B, \Gamma, s)$
RETURN $numTerms$

$countHeadVarArgTerms(B, \Gamma, s) =$
$numTerms \leftarrow 0$
$numVarWithTypeInEnv \leftarrow typeCnt(\Gamma, B)$
IF $numVarWithTypeInEnv > 0$ THEN
FOR each $(s_1, \cdots, s_m) \in ndk(s-1-m, m)$
$numTerms \leftarrow numTerms + \prod_{i=1}^{m} countTerm(B_i, \Gamma, s_i)$
RETURN $numVarWithTypeInEnv \times numTerms$

FIGURE 2.1 – L'algorithme de comptage des termes en forme normale d'un type et d'une taille donné

$$varType?(A) = \begin{cases} true & \text{if } A = a \\ false & \text{if } A = A_1 \to A_2 \end{cases}$$

$$arrowType?(A) = \begin{cases} false & \text{if A = a} \\ true & \text{if } A = A_1 \to A_2 \end{cases}$$

$$argType(A) = \begin{cases} Error & \text{if A = a} \\ A_1 & \text{if } A = A_1 \to A_2 \end{cases}$$

$$resType(A) = \begin{cases} Error & \text{if A = a} \\ A_2 & \text{if } A = A_1 \to A_2 \end{cases}$$

$validHeadVarTypeSet(A, \Gamma) = \{B | B = B_1 \to \cdots B_m \to A, m \geq 0, B \in dom(\Gamma)\}$
$ndk(n, k) = \{(s_1, \cdots, s_k) | s_i \geq 1, \sum_{i=1}^{k} s_i = n\}$

random(bound)=générateur de nombre aléatoire qui retourne un nombre aléatoire entre 0 (inclusif) et une borne (exclusif).

Entrée :type $A = A_1 \to A_2 \to \cdots \to A_n \to a$, taille s
Sortie :un terme en forme β-normale de taille s et de type A, générer uniformement aléatoire.
gen(A,s)=genTerm(A,0,s)
IF $s < 1$ THEN
RETURN None
ELSE IF $s = 1$ THEN
IF $typeCnt(\Gamma, A) > 0$ THEN
RETURN $genVarTerm(\Gamma, A)$
ELSE
RETURN None
ELSE IF $arrowType?(A)$ THEN
$totalNumTerm \leftarrow countTerm(A, \Gamma, s)$
$numLamTerm \leftarrow countTerm(resType(A), typeCntInc(\Gamma, argType(A)), s-1)$
IF$(randomtotalNumTerm) < numLamTerm$ THEN
RETURN $genLamTerm(argType(A), resType(A), \Gamma, s)$
ELSE
RETURN $genAppTerm(A, \Gamma, s, totalNumTerms - numLamTerm)$
ELSE RETURN $genAppTerm(A, \Gamma, s, countTerm(A, \Gamma, s))$

$genVarTerm(A, \Gamma) =$
IF $typeCnt(\Gamma, A) = 0$ THEN
RETURN None
ELSE
RETURN $x^{index(\Gamma,A)}_{randomtypeCnt(\Gamma,A)}$
$genAppTerm(A, \Gamma, s, numAppTerms) =$
$(B, (s_1, s_2, \cdots, s_m)) \leftarrow chooseHeadVar(A, \Gamma, s, numAppTerms)$
$headVar \leftarrow genVarTerm(B, \Gamma)$
FOR $i = 1$ TO m
$term_i \leftarrow genTerm(B_i, \Gamma, s_i)$
RETURN $(\cdots((headVarterm_1)\cdots term_{m-1})term_m)$

$genLamTerm(argType, resType, \Gamma, s) =$
$var \leftarrow x^{index(\Gamma,argType)}_{typeCnt(\Gamma,argType)}$
$body \leftarrow genTerm(resType, typeCntInc(\Gamma, argType), s-1)$
RETURN $\lambda var.body$

$chooseHeadVar(A, \Gamma, s, numAppTerms) =$
$randNum \leftarrow randomnumAppTerms$
$numTerms \leftarrow 0$
FOR $textiteachB = B_1 \to \cdots B_m \to A \in validHeadVarTypeSet(A, \Gamma)$
$numTerms \leftarrow numTerms + countHeadVarArgTerms(B, \Gamma, s)$
IF $randNum < numTerms$ THEN
RETURN $(B, chooseArgSize(B, \Gamma, s, (countHeadVarArgTerms(B, \Gamma, s))/typeCnt(\Gamma, B)))$

$chooseArgSize(B, \Gamma, S, numArgTerms) =$
$randNum \leftarrow randomnumArgTerms$
$numTerms \leftarrow 0$
FOR $each(s_1, \cdots, s_m) \in ndk(s-1-m, m)$
$numTerms \leftarrow numTerms + \prod_{i=1}^{m} countTerm(B_i, \Gamma, s_i)$
IF $randNum < numTerms$ THEN
RETURN $(s_1, \cdots, s_m)$

FIGURE 2.2 – Algorithme de génération de terme d'un type A et de taille s.

6 Enumération des habitants des types simples (G.Dowek et Y.Jiang)

Dans cette section, nous exposerons l'approche de G.Dowek et Y.Jiang pour définir une grammaire de type 2 qui énumère les habitants en forme β-normale d'un type donné. pour arriver à ce résultat Dowek et Jiang dans [19, 20] introduisent la notion de schéma de termes qui consiste à identifier les variables ayant le même type.
Rappelons quelques définitions utiles pour ce qui suit :

Définition 6.1. *(Séquent) Un séquent est la paire* $\Gamma \vdash A$ *ou* Γ *est un ensemble fini de types appelé contexte du séquent, et* A *est un type appelé conclusion du séquent.*
On dit que le terme t *a le séquent type* $\Gamma \vdash A$ *s'il a le type* A*, et les types de ses variables libres sont dans* Γ *.*
Formellement, l'ensemble des termes du séquent types $\Gamma \vdash$ *est défini inductivement comme suit :*

- *Si* x *est une variable de type* A *et* A *est un élément de* Γ *alors* x *est un terme du séquent type* $\Gamma \vdash A$*.*
- *Si* x *est une variable de type* A *et* t *est un terme du séquent type* $\Gamma \cup \{A\} \vdash B$ *alors* λxt *est un terme du séquent type* $\Gamma \vdash A \to B$*.*
- *Si* t *est un terme du séquent type* $\Gamma \vdash A \to B$ *et* u *est un terme du séquent type* $\Gamma \vdash A$ *alors* (tu) *est un terme du séquent type* $\Gamma \vdash B$

Définition 6.2. *(sous-formule) La sous-formule d'un type est définie comme suit :*

1. *Sub(T)={T}, si T est une variable de type (atome).*
2. *Sub(*$A \to B$*)={*$A \to B$*}* $\cup$ *Sub(A)* $\cup$ *Sub(B).*

La sous-formule d'un séquent est définie comme suit :
Sub($\Gamma \vdash A$*)=Sub(*Γ*)* $\cup$*Sub*$\{A\}$ *tel que :*

$$Sub(\Gamma) = \begin{cases} \emptyset & Si\ \Gamma = \emptyset \\ Sub\{B_1\} \cup Sub\{B_2\} \cdots \cup Sub\{B_m\} & Si\ \Gamma = \{B_1, B_2, \cdots, B_m\} \end{cases}$$

L'ensemble des sous-formules d'un séquent $\Gamma \vdash A$ *ou d'un type est toujours fini.*

Définition 6.3. *(Variable canonique)Une variable typé est dite canonique dans un terme donné si elle est identique à toute variable de même type.*

Définition 6.4. *(Schémas de terme) Le schéma de terme est un terme en forme normale longue dont les variables sont toutes canoniques.*

Exemple 6.1. – *La projection d'un terme t est le schéma de terme obtenu par le remplacement des occurrences de variables par la variable canonique d'un même type.*

Parmi les termes normaux fermés de type $((T \to T) \to T) \to T$ *nous avons les termes :*
$\lambda F(F\lambda x(F\lambda yx))$ *et* $\lambda F(F\lambda x(F\lambda yy))$ *où*
F est de type $(T \to T) \to T$*, x et y sont de type T.*
Si on considère que x et y identique car de même type, on peut fusionner ces deux termes en un seul appelé schéma de terme :

$$\lambda F(F\lambda x(F\lambda xx))$$

Si x est une variable canonique de type T et F est une variable canonique de type $(T \to T) \to T$ *alors le terme*

$$\lambda F(F\lambda x(F\lambda x(F\lambda xx)))$$

est un schéma de terme et

$$\lambda F(F\lambda x(F\lambda y(F\lambda zy)))$$

n'est pas un schéma de terme

6.1 Grammaire pour énumérer les schémas de terme

Dans [19], les auteurs construisent une grammaire à contexte libre pour énumérer l'ensemble des schémas termes d'un type donné.

Définition 6.5. *(Schéma de la grammaire)*
Soit $\Gamma \vdash A$ *un séquent,* $\mathcal{T}$ *l'ensemble des sous-formules de* $\Gamma \vdash A$*.*
S est l'ensemble fini des séquents construit avec les types de $\mathcal{T}$*.*
On associe au séquent $\Gamma \vdash A$ *la grammaire G(T,N,R,* $S_{\Gamma\vdash A}$*) construite comme suit :*

- $\{(,), \lambda\} \in T$.
- $S_{\Gamma\vdash A} \in N$.
- *Les règles de productions R :*
  Pour tout symbole non-terminal $S_{\triangle\vdash A}$ *faire*
  -Si on a $S_{\triangle\vdash B\to C} \in N$ *alors ajouter la règle :*

$$S_{\triangle\vdash B\to C} \to \lambda x s_{\triangle\cup\{B\}\vdash C}$$

où x est la variable canonique de type B, $\{x\}\cup T$ et $s_{\triangle\cup\{B\}\vdash C}\in N$.
-Si $S_{\triangle\vdash a}\in N$ et $B_1\to\cdots\to B_n\to a\in\triangle$ alors ajouter la règle

$$S_{\triangle\vdash a} \to (x_{S_{\triangle\vdash B_1}\cdots S_{\triangle\vdash B_n}})$$

où x est une variable canonique de type $B_1\to\cdots\to B_n\to a$ et $\{S_{\triangle\vdash B_1}\}\cup N\cdots\{S_{\triangle\vdash B_n}\}\cup N$.
Le cas particulier de la deuxième règle est :
-Si $S_{\triangle\vdash a}\in N$ et $a\in\triangle$ alors ajouter la règle :

$$S_{\triangle\vdash a}\to x$$

x est une variable canonique de type a

Exemple 6.2. *Construisons la grammaire $G(T,N,R,S_{\Gamma\vdash A})$ qui engendre l'ensemble des schémas de termes habitants au type $A\equiv(a\to a)\to a\to a$*
Le contexte ici est vide $\Gamma=\emptyset$.

- *T contient initialement $\{(,),\lambda\}$.*
- *N contient initialement $S_{\emptyset\vdash A}$, l'axiome.*
- *Déterminons les règles de productions :*

$$\begin{array}{lcll}
S_{\Gamma\cup\vdash((a\to a)\to a\to a)} & \to & \lambda x\,\underbrace{S_{\emptyset\cup\{a\to a\}\vdash a\to a}}_{S_1} & |\{x\}\cup T \text{ et } \{S_1\}\cup N\\
\underbrace{S_{\emptyset\cup\{a\to a\}\vdash a\to a}}_{S_1} & \to & \lambda y\,\underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} & |\{y\}\cup T \text{ et } \{S_2\}\cup N\\
\underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} & \to & y & \\
\underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} & \to & x\,\underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} &
\end{array}$$

Alors :

$$\begin{array}{lcl}
S & \to & \lambda x S_1\\
S_1 & \to & \lambda y S_2\\
S_2 & \to & y\\
S_2 & \to & (xS_2)
\end{array}$$

Exemple 6.3. *de la même manière on construit les règles de la grammaire $G(T,N,R,S_{\Gamma\vdash A})$ du séquent $\vdash((T\to T)\to T)\to T$ sont :*

$$\begin{array}{lcl}
S & \to & \lambda F S_1\\
S_1 & \to & (FS_2)\\
S_2 & \to & \lambda x S_3\\
S_3 & \to & x\\
S_3 & \to & (FS_4)\\
S_4 & \to & \lambda x S_3
\end{array}$$

où x est une variable canonique de type T et F est la variable canonique de

type $(T \to T) \to T$

$$\begin{array}{rcl} S & = & S_{\vdash((T\to T)\to T)\to T} \\ S_1 & = & S_{(T\to T)\to T\vdash T} \\ S_2 & = & S_{(T\to T)\to T\vdash T\to T} \\ S_3 & = & S_{(T\to T)\to T,T\vdash T} \\ S_4 & = & S_{(T\to T)\to T,T\vdash T\to T} \end{array}$$

Proposition 6.1. *1. Un terme en forme normale longue du type séquent $\Delta \vdash B \to C$ a la forme λxt pour toute variable x de type B et d'un terme t de type séquent $\Delta \cup \{B\} \vdash C$.*

2. Un terme en forme normale longue de type séquent $\Delta \vdash T$ où T est un type atomique a la forme $(xt_1, \cdots, t_n)$ pour toute variable x de type $B_1 \to \cdots \to B_n \to T \in \Delta$ et les termes $t_1, \cdots, t_n$ des types séquent $\Delta \vdash B_1, \cdots, \Delta \vdash B_n$.

Proposition 6.2. *Soit le séquent $\Gamma \vdash A$, soit $\mathcal{T}$ l'ensemble des sous-formules de $\Gamma \vdash A$ et S ensemble fini des séquents construits des types de $\mathcal{T}$, soit le séquent $\Delta \vdash B$ de S et t le schéma de terme du type séquent $\Delta \vdash B$, alors t est généré dans $S_{\triangle\vdash B}$ par la grammaire décrite par la définition 6.5.*

***Preuve** (voir [19]) : par induction sur la structure de t*

- *Si $B = C \to D$ alors $t = \lambda xu$ où x est une variable canonique de type C et u est de type séquent $\Delta \cup \{C\} \vdash D$ (proposition 6.1), par hypothèse d'induction u est généré par $S_{\Delta\cup\{C\}\vdash D}$ et t est généré par $S_{\triangle\vdash B}$ utilisant la règle :*

$$S_{\Delta\vdash C\to D} \to \lambda x S_{\Delta\cup\{C\}\vdash D}$$

- *Si B est un type atome alors $t = (xt_1, \cdots, t_n)$ où x est une variable canonique de type $C_1 \to \cdots \to C_n \to B \in \Delta$ et $t_1, \cdots, t_n$ sont des termes des types séquent $\Delta \vdash C_1, \cdots \Delta \vdash C_n$ respectivement (proposition 6.1). Par hypothèse d'induction, tout type t_i est généré par $S_{\Delta\vdash C_i}$ et t est généré par $S_{\Delta\vdash B}$ utilisant la règle :*

$$S_{\Delta\vdash B} \to (xS_{\Delta\vdash C_1}, \cdots, S_{\Delta\vdash C_n})$$

6.2 Grammaire pour énumérer les termes fermés et ouverts

Pour simplifier la grammaire on suppose qu'on a une seule variable pour chaque type.

Pour construire cette grammaire on utilise la grammaire précédente et on ajoute un index pour chaque variable.

Définition 6.6. *Soit $\Gamma \vdash A$ un séquent*
Soit $\mathcal{T}$ l'ensemble des sous-formules de $\Gamma \vdash A$ et S un ensemble fini construit à partir de $\mathcal{T}$.
Pour chaque séquent $\Gamma \vdash B \in S$, on associe le symbole non-terminal $S_{\Delta \vdash B}$ et on prend les règles.

$$S_{\Delta \vdash B \to C} \to \lambda x_i S_{\Delta \cup \{B\} \vdash C}$$

x est une variable de type B
-Si Δ contient le type $B_1 \to \cdots \to B_n \to T$ on prend la règle

$$x_i S_{\Delta \vdash B_1} \cdots S_{\Delta \vdash B_n}$$

où x est une variable de type $B_1 \to \cdots \to B_n \to T$ on ajoute les deux règles suivante :

$$i \to 0$$

$$i \to S_i$$

Preuve par induction sur la structure de t

- Si $B = C \to D$ alors $t = \lambda x_i u$ où x_i est une variable lettre de type C et u est de type séquent $\Delta \cup \{C\} \vdash D$ (proposition 6.1), par hypothèse d'induction u est généré par $S_{\Delta \cup \{C\} \vdash D}$ et t est généré par $S_{\triangle \vdash B}$ utilisant la règle :

$$S_{\Delta \vdash C \to D} \to \lambda x_i S_{\Delta \cup \{C\} \vdash D}$$

- Si B est un type atome alors $t = (x_i t_1, \cdots, t_n)$ où x_i est une variable lettre de type $C_1 \to \cdots \to C_n \to B \in \Delta$ et $t_1, \cdots, t_n$ sont des termes des types séquent $\Delta \vdash C_1, \cdots \Delta \vdash C_n$ respectivement. Par hypothèse d'induction, tout type t_i est généré par $S_{\Delta \vdash C_i}$ et t est généré par $S_{\Delta \vdash B}$ utilisant la règle :

$$S_{\Delta \vdash B} \to (x_i S_{\Delta \vdash C_1}, \cdots, S_{\Delta \vdash C_n})$$

(voir l'exemple dans [19])

6.3 Grammaire infinie pour énumérer les termes normaux fermés

La grammaire donnée pour énumérer les termes ouverts et fermés peut être transformée à une grammaire infinie qui génère les termes normaux fermés d'un type donné, on contrôle les indices des variables libres des termes générés par le symbole non-terminale $S_{B_1,\cdots,B_n\vdash C}$, Ajouter un paramètre pour chaque B_i.
Le symbole non-terminal $B_1^{k_1},\cdots,B_n^{k_n}$ génère tous les termes dont les variables libres de type B_i avec indice $\leqslant k_i$

Définition 6.7. – *Si* $C \notin \{B_1,\cdots,B_n\}$ *on prend la règle*

$$S_{B_1^{k_1},\cdots,B_n^{kn}\vdash C\to D} \to \lambda x_0 S_{B_1^{k_1},\cdots,B_n^{kn},C^0\vdash D}$$

ou x *est une variable lettre de type* C

– *Si un élément de* $\{B_1,\cdots,B_n\}$, *soit* B_1 *on prends la règle*

$$S_{B_1^{k_1},\cdots B_n^{kn}\vdash C\to D} \to \lambda x_{Sk_1} S_{B_1^{Sk_1},B_2^{k_2}\cdots,B_n^{kn}\vdash D}$$

ou x *est la variable lettre de* C.

– *Si un élément de* $\{B_1,\cdots,B_n\}$, *soit* B_1 *a la forme*

$$C_1\to\cdots\to C_p\to T$$

$$S_{B_1^{k_1},\cdots,B_n^{kn}\vdash T} \to (x_{i_{k_1}} S_{B_1^{k_1},\cdots,B_n^{kn}\vdash C_1}{}^{\cdots} S_{B_1^{k_1},\cdots,B_n^{kn}\vdash C_p})$$

où x *est une variable de lettre de type* B_1

L'indice de la variable x *est un nouveau non-terminal* i_q *on ajoute des règles pour ce symbole pour générer tous les indices* $\leqslant q$

$$i_n \to n$$
$$i_{s_n} \to i_n$$

Chapitre 3

Proposition d'algorithmes de génération des habitants d'un type

1 Machine *Mat* (α, Γ) pour l'énumération des habitants d'un type

Dans cette partie, nous présentons un algorithme d'énumération des habitants d'un type α de degré quelconque, cet algorithme est le fruit de l'étude des algorithmes précédents il se base sur la notion de contexte.

1.1 Machine d'énumération des habitants en forme β-normale longue

Entrée : $Mat(\alpha_1 \to \cdots \to \alpha_n \to o, \Gamma = \emptyset)$.
Sortie : Ensemble des habitants en forme normale longue.

1. **Si** l'état de la machine est $Mat(\alpha_1 \to \alpha_2, \Gamma)$ **alors** remplacer $Mat(\alpha_1 \to \alpha_2, \Gamma)$ par $\lambda x^{\alpha_1}.Mat(\alpha_2, \Gamma \cup \{x : \alpha_1\})$.
2. **Si** l'état de la machine est $Mat(o, x : \alpha_1 \to \cdots \to \alpha_n \to o \in \Gamma)$ **alors** remplacer :
   $Mat(o, x \in \Gamma)$ par $xMat(\alpha_1, \Gamma)Mat(\alpha_2, \Gamma) \cdots Mat(\alpha_n, \Gamma)$
3. **Si** $Mat(o, \Gamma = \emptyset)$ **alors** arrêter la machine, pas d'habitants.
4. **Si** $Mat(o, x : \alpha_1 \to \cdots \to \alpha_n \to o \notin \Gamma)$ (pour $i \geq o$) **alors** arrêter la machine, pas d'habitants.

5. **Si** $Mat(\emptyset, \Gamma)$ **alors** Stop et lire le terme.

Exemple 1.1. *Soit le type* $(a \to a) \to a \to a$

$$\underbrace{Mat((a \to a) \to a \to a, \emptyset)}_{S} \to \lambda x^{a \to a}. \underbrace{Mat(a \to a; \{x : a \to a\})}_{S_1}$$
$$\to \lambda x^{a \to a} \lambda y^{a}. \underbrace{Mat(a; \{x : a \to a, y : a\})}_{S_2}$$
$$\to \lambda x^{a \to a} \lambda y^{a}. x \underbrace{Mat(a; \{x : a \to a, y : a\})}_{S_2}$$
$$\to \lambda x^{a \to a} \lambda y^{a}. xy Mat(\emptyset; \{x : a \to a, y : a\})$$

ou bien

$$\to \lambda x^{a \to a} \lambda y^{a}. x(x(\underbrace{Mat(a; \{x : a \to a, y : a\})}_{S_2}))$$

Les autres termes sont générés par application de cette règle

$$\underbrace{Mat(a; \{x : a \to a, y : a\})}_{S_2} \to x \underbrace{Mat(a; \{x : a \to a, y : a\})}_{S_2} | y$$

récursivement.

Définition 1.1. *(Grammaire* $Mat(\alpha, \Gamma)$*) soit la machine* $Mat(\alpha, \Gamma)$ *d'un type* α *et d'un contexte* Γ, T *est l'ensemble des sous types de* α, S *est l'ensemble fini des machines construites à partir des types de* T, *on associe à* $Mat(\alpha, \Gamma)$ *la grammaire* $G(V_T, V_N, P, Mat(\alpha, \Gamma))$ *est définie comme suit :*

- $\{(,), \lambda\} \in V_T$.
- $Mat(\alpha, \Gamma) \in V_N$.
- *Les règles de productions* P *:*

  Pour tout $Mat(\alpha, \Gamma) \in N$ *faire*

  Si *on a* $Mat(\alpha_1 \to \alpha_2, \Gamma) \in V_N$ ***alors*** *on ajoute*

  si $x : \alpha_1 \in \Gamma$ ***alors*** *ajouter la règle*
  $Mat(\alpha_1 \to \alpha_2, \Gamma) \to \lambda x^{\alpha_1}. Mat(\alpha_2, \Gamma)$

  sinon $Mat(\alpha_1 \to \alpha_2, \Gamma) \to \lambda x^{\alpha 1}. Mat(\alpha_2, \Gamma \cup \{x : \alpha_1\})$.

  Si *on a* $Mat(o, \Gamma) \in V_N$ *et* $x : \alpha_1 \to \cdots \to \alpha_n \to o \in \Gamma$ ***alors*** *ajouter la règle :*
  $Mat(o, \Gamma) \to x Mat(\alpha_1, \Gamma) \cdots Mat(\alpha_n, \Gamma) \qquad V_N \cup \{Mat(\alpha_i, \Gamma)\}$ *(pour* $i = 1, \cdots, n$*), le cas particulier de cette règle est :*

  Si *on a* $Mat(o, \Gamma) \in V_N$ *et* $x : o \in \Gamma$ ***alors*** *ajouter la règle :*
  $Mat(o, \Gamma) \to x$

2 Les types finement engendrés

Plusieurs auteurs ont étudié le problème d'énumération des habitants d'un type (Voir chapitre 2), l'ensemble de ces habitants est généralement infini.
Dans ce chapitre nous intéressons à étudier ce problème d'énumération par la définition d'un ensemble $E = \{G_1, G_2, \cdots, G_n, M_{01}, M_{02}, \cdots, M_{0m}\}$ fini de termes (termes générateurs et termes initiaux) pour un type α, cet ensemble fini permet de représenter les habitants de ce type (par exemple si on prend le type des entiers de Church $(a \to a) \to a \to a$, l'ensemble de ses habitants est finement engendré c'est à dire, il est généré à partir des deux termes {0, S}, Zéro et la fonction successeur).
L'idée est déjà donnée dans [23] pour les types de degré ≤ 2, la méthode proposée ne caractérise pas complètement les habitants d'un type puisque elle suppose déjà connu l'un de ces habitants (terme initial). Nous enrichissons cette dernière par une procédure de calcul des termes initiaux et nous étudions ce problème pour une classe plus grande des types (les types finement engendrés de degré ≤ 3).
Donnons d'abord quelques définitions.

Définition 2.1. *(termes générateurs) Soit un type α, on appelle un terme générateur **G** tout terme de type $\alpha \to \alpha$ tel que si on applique ce terme **G** à un autre terme M de type α on obtient un autre terme de type α $(GM : \alpha)$.*

Notation 2.1. *Pour tout ensemble C de λ-termes, $[C]$ est l'ensemble des λ-termes construit à partir de C en utilisant seulement les applications.*

Définition 2.2. *On dit qu'un ensemble S de λ-termes fermés est finement engendré s'il existe un ensemble fini C de λ-termes fermés tel que chaque λ-terme de S est $\beta\eta$-equivalent d'un λ-terme de $[C]$.*

Définition 2.3.

- *Rank d'un type (degré) :*
  rk (α) est défini récursivement par :
  $rk(o) = 0$ *et* $rk\,(A \to B) = Max\,(rk\,(A) + 1, rk\,(B))$

  Exemple 2.1.

  1. $rk\,(((o \to o) \to (o \to ((o \to o) \to o)))) = 2$
  2. $rk\,(((o \to (o \to (o \to o))) \to o) \to (o \to (((o \to (o \to o)) \to o) \to o))) = 3$

- *Argument d'un type α :*
  $\alpha = A_1 \to (A_2 \to \cdots (A_n \to o))$
  Arg $(\alpha) = \{A_1, A_2, \cdots, A_n\}$

2.1 Génération des habitants des types de degré 2 à une seule variable

Hirokawa a donné l'idée (voir [31]) et il a montré qu'on pouvait utiliser la notion de grammaire à contexte libre pour générer les habitants en forme β-normale dans le cas particulier des types de degré deux :
Un type de degré deux est de la forme :
$\alpha = (a_1^1 \to \cdots \to a_{n1}^1 \to a^1) \to \cdots \to (a_1^m \to \cdots \to a_{nm}^m \to a^m) \to a$ où
$a_1^1, \cdots, a_{n1}^1, a^1, \cdots, a_1^m \cdots, a_{nm}^m, a^m$ et a sont des variables de types,
$(n_1, \cdots, n_m \geq 0)$ et $m \geq 1$.
Etant donné un type α de degré deux, l'auteur a montré comment on peut construire une grammaire à contexte libre G(α) tel que :
Nhabs (α) = $\{N | N \longleftarrow \lambda x_1 \cdots x_m.M\}$ pour un certain M $\in$ L(G(α)).
ou L(G(α)) est le langage engendré par G(α).

Dans [23] Hemdani s'est limité aux types ayant un seul symbole non-terminal ($V_N = \{a\}$) apparaissant au plus une fois dans chaque membre droit des règles de production, les règles de production étudiées sont de la forme :

$a \to x_i x_{i1} \cdots x_{ij} a x_{ij+1} \cdots x_{in}$ (1)
ou bien
$a \to x_j x_{j1} \cdots x_{jn}$ (2)

Les grammaires traitées ici, l'itération est assurée par les règles de production de la forme (1). Le générateur associé aux grammaires ayant un seul symbole non-terminal apparaissant une seule fois dans chaque membre droit des règles de production est de la forme :

$G_i = \lambda y x_1 \cdots x_m.x_i x_{i1} \cdots x_{ij}(y x_1 \cdots x_m) x_{ij+1} \cdots x_{in}.$

Dans la suite l'auteur a traité les grammaires ayant un seul symbole non-terminal qui peut apparaitre plusieurs fois dans le membre droit des règles de production, les règles de production sont de la forme :
$a \to x_j x_{j1} \cdots x_{jn1} a_1 \cdots x_{jp} \cdots x_{jnp} a_p \cdots x_{jn} \cdots x_{jnm}$ (3)

Pour chaque règle de production de la forme (3) il peut définir $p * q^{p-1}$ (q : règle de production triviales, p : le nombre de a) générateurs selon l'algorithme suivant : ([23] page 77)

pour i=1 à p faire
Début :

1. Prendre une combinaison de (q-1) dérivation triviales (parmi les q^{p-1} combinaison distinctes).
2. Remplacer les (p-1) occurrence du symbole a (exceptée la i^{eme}) par les dérivations triviales correspondantes.
3. Construire l'unique itérateur associé à la règle de production obtenue (ayant une seule occurrence du symbole a dans son membre droit).
4. Prendre la combinaison de dérivations triviales suivante et allez à 2.

Fin :

Appliquant cet algorithme à l'exemple suivant :

Exemple 2.2. *Soit le type* $\alpha = (o \to o \to o) \to o \to o \to o$ *de degré 2 :*
La grammaire associée à ce type (relativement à la base $\{x : o \to o \to o, y : o, z : o\}$*) a pour règles de production :*
$o \to xoo|y|z$
Dans ce cas, l'itération est assurée par la règle $o \to xoo$*, Hemdani a extrait 4 générateurs :*
$G_1 = \lambda oxyz.\,(x\,(oxyz)\,y)$
$G_2 = \lambda oxyz.\,(x\,(oxyz)\,z)$
$G_3 = \lambda oxyz.\,(xy\,(oxyz))$
$G_4 = \lambda oxyz.\,(xz\,(oxyz))$

Pour calculer les autres termes l'auteur a supposé un terme initial
$M = \lambda xyz.x\,(xyz)\,y$, appliquant ces générateurs au terme M :
$G_1(G_2(G_3(G_4M))) \twoheadrightarrow \lambda xyz.x(x(xy(xz(x(xyz)y)))z)y$
les autres termes sont obtenus par application des G_i à M $(i = 1, \cdots, 4)$.

Simplification et amélioration de la procédure $(G2)$

La méthode présentée ci-dessus ne traite que le cas des grammaires de degré 2 elle ne caractérise pas complètement les habitants d'un type α donné puisqu'elle suppose déjà connu l'un de ces habitants (terme initial).
Dans la suite nous enrichissons la méthode proposée $(G2)$ par une procédure de génération $(G3)$ pour calculer les termes initiaux et les générateurs des types finement engendrés de degré ≤ 3 sous la forme étudiée par Thierry joly (voir [25, 12]).

Dans la section suivante nous donnons une procédure qui permet de générer un ensemble fini de termes générateurs et de termes initiaux pour les types α de degré quelconque on se base sur la grammaire de Dowek [20].

Définition 2.4. *(Grammaire associée à un type de degré 2) Soit le type* $\alpha = (o_1^1 \to \cdots \to o_{n1}^1 \to o^1) \to \cdots \to (o_1^m \to \cdots \to o_{nm}^m \to o^m) \to o$ *Soit* $B = \{x_i : o_1^i \to \cdots \to o_{ni}^i \to o^i | i = 1, \cdots, m\}$ *une base de type avec* $o_1^i, \cdots, o_{ni}^i, o^i$ *et* o *des variables de type atomique* $(i = 1, \cdots, m)$
on définit une grammaire à contexte libre $G(B) =< V_T, V_N, o, P >$ *comme suit :*

- *Ensemble des symboles terminaux* $V_T = \{x_1, \cdots, x_m, (,), ., \lambda\}$.
- *Ensemble des symboles non-terminaux* $V_N = \{o\}$.
- *Ensemble des règles de production,* $P = \{o^i \to x_i o_1^i \cdots o_{ni}^i | i = 1, \cdots, m\}$.

Remarque 2.1. $P_i \in o \to V_T V_N^*$.

Définition 2.5. *Procédure de génération* $(G2)$
Entrée :*Un type* $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to o /$
$(\alpha_i \equiv o_1^i \to \cdots \to o_{ni}^i \to o^i | i = 1, \cdots, m)$
Sortie : *deux ensembles* $E_{M_0} = \{M_{01}, \cdots, M_{0n}\}$ *et* $E_G = \{G_1, \cdots, G_m\}$ *appelés respectivement ensembles des termes initiaux et ensembles des termes générateurs :*

Etape 0 :

		$E_{M_0} = E_{G_0} = \emptyset$
	Si	$o \to x_j \in P$ ***alors*** *ajouter le terme initial* M_{0j}
		$M_{0j} = \lambda x_1 \cdots x_m . x_j$
		$E_{M_0} \cup \{M_{0j}\}$.
	Si	$o \to x_i o_1 \cdots o_m \in P$ *et* $m \geq 1$ ***alors*** *ajouter le générateur* G_i
		$G_i = \lambda f_1 \cdots f_m x_1 \cdots x_n . x_i (f_1 x_1 \cdots x_n) \cdots (f_m x_1 \cdots x_n)$
		$E_G \cup \{G_i\}$

Etape1 :

Si	$E_{M_0} = \emptyset$	***alors*** *pas d'habitants pour ce type*
Sinon	***Si***	$E_G = \emptyset$ ***alors*** *l'ensemble des habitants est* E_{M_0}
	Sinon	*l'ensemble des habitants est E tel que*
		$E = E_{M_0} \cup \{G_1 M_{01}, \cdots, (G_1 \cdots (G_1 M_{01})) \cdots, G_m M_{0n}, \cdots,$
		$(G_m \cdots (G_m M_{0n}))\} \cup \{G_k \cdots (G_p M_{0j})\}$
		tel que $j = 1, \cdots n$ *et* $G_k, \cdots, G_p \in \{G_1, \cdots, G_m\}$

Preuve : Démontrons que si $M_{0j} : \alpha$ alors $(((G_i M_{0j}) M_{0j+1}) \cdots M_{0m}) : \alpha$
Donc l'assignation de type au $(((G_i M_{0j}) M_{0j+1}) \cdots M_{0m}) : \alpha$ se fait selon la déduction :

$$\frac{f_1 : \alpha_1 \to \cdots \to \alpha_n \to o \quad x_1 : \alpha_1 \cdots x_n : \alpha_n}{f_1 x_1 \cdots x_n : o}(\to E) \cdots \frac{f_m : \alpha_1 \to \cdots \to \alpha_n \to o \quad x_1 : \alpha_1 \cdots x_n : \alpha_n}{f_m x_1 \cdots x_n : o}(\to E)$$

On a $x_i : \alpha_i$ et d'après la construction de la grammaire la variable x_i se trouve en tête d'une règle o, α_i a donc la forme :
$\alpha_i \equiv \underbrace{o \to \cdots \to o}_{m \text{ fois o}} \to o$
la suite de l'assignation de type au terme $(((G_i M_{0j}) M_{0j+1}) \cdots M_{0m})$ se fera donc comme suit :

$$\frac{x_i : \overbrace{o \to \cdots \to o}^{m \text{ fois o}} \to o \quad (f_1 x_1 \cdots x_n : o) \cdots (f_m x_1 \cdots x_n : o)}{x_i (f_1 x_1 \cdots x_n) \cdots (f_m x_1 \cdots x_n) : o}$$

En déchargeant les hypothèses $x_1 : \alpha_1, \cdots, x_n : \alpha_n$ dans l'ordre nous obtenons :

$\lambda x_1 \cdots x_n . x_i (f_1 x_1 \cdots x_n) \cdots (f_m x_1 \cdots x_n) : \alpha_1 \to \cdots \to \alpha_n \to o$
$G_i = \lambda f_1 \cdots f_m \lambda x_1 \cdots x_n . x_i (f_1 x_1 \cdots x_n) \cdots (f_m x_1 \cdots x_n) : \beta_1 \to \cdots \to \beta_m \to \alpha_1 \to \cdots \to \alpha_n \to o$

et nous terminons l'assignation de type pour $(((G_i M_{0j}) M_{0j+1}) \cdots M_{0m})$ comme suit :

$$\frac{G_i : \beta_1 \to \cdots \to \beta_m \to \alpha_1 \to \cdots \to \alpha_n \to o \quad M_{0j} : \beta_1, \cdots, M_{0m} : \beta_m}{((G_i M_{0j}) \cdots M_{0m}) : \alpha_1 \to \cdots \to \alpha_n \to o \equiv \beta_1 \equiv \cdots \equiv \beta_m \equiv \alpha}$$

□

Nous reprenons l'exemple 2.2 ($\alpha = (o \to o \to o) \to o \to o \to o$) étudié dans ([23] page 79) présenté ci-dessus, calculons les générateurs et les termes initiaux pour extraire l'ensemble des habitants de ce type.
Suivant notre méthode on traite le type comme suit :
Les termes initiaux :
on a $o \to y$ et $o \to z \in P$ alors on ajoute les deux termes initiaux :

$M_{00} = \lambda xyz.y \quad M_{01} = \lambda xyz.z$ donc $E_{M_0} = \{M_{00}, M_{01}\}$

Les générateurs :

on a $o \to xoo \in P$ alors on ajoute le générateur :
$G = \lambda f_1 f_2 xyz.x\,(f_1 xyz)\,(f_2 xyz)$ donc $E_G = \{G\}$
le terme ($\lambda xyz.x\,(x\,(xy\,(xz\,(x\,(xyz)\,y)))\,z)\,y$) trouvé dans ([23] page 79)

peut être calculé comme suit :
Il suffit d'appliquer
$(G\,(M_{00}M_{01})) \equiv \lambda xyz.xyz = M_1$
on applique une autre fois (G (M_1M_{00})) on obtient le terme $\lambda xyz.x\,(x\,(xyz))\,y = M_2$ puis G $(M_{01}M_2)$ pour obtenir le terme $\lambda xyz.xz(x(xyz)y) = M_3$ après $G(M_3M_{01}) \equiv x(xz(x(xyz)y))z = M_4$ enfin
$G(M_4M_{00}) \equiv \lambda xyz.x\,(x\,(xy\,(xz\,(x\,(xyz)\,y)))\,z)\,y$.

2.2 Génération des habitants des types de degré ≤ 3 ($G3$)

Thierry joly a démontré (voir [25,26,27,28,29]) que les types finement engendrés sont les types de degré ≤ 2 et les types de degré 3 sous la forme $A_1 \to (A_2 \to \cdots \to (A_n \to o))$ tel que $A_i = o$, $A_i = (o \to o)$ ou
$A_i = (o \to (o \to \cdots \to (o \to o))) \to o \quad (\ pour\ i = 1, \cdots, n)$
Le type résultat par le remplacement des A_i par $(o \to (o \to \cdots (o \to o))) \to o$ est un type de degré 3.

Dans cette partie nous essayons d'étudier ces types cas par cas, extraire les grammaires, les termes initiaux, les générateurs pour définir l'ensemble des habitants de ces types α.

Prenons l'idée de Thierry joly (voir [25]) et la notion de générateur **G** (ou itérateur) calculons les termes initiaux, appliquons les générateurs aux termes initiaux nous obtenons par la suite les habitants des types de degré ≤ 3.
Notre méthode consiste à :

1. Extraire la grammaire de ces types.
2. Calculer les termes initiaux pour le type α.
3. Calculer les générateurs $G_i : \alpha \to \alpha$ pour le type α.
4. Par application de ces itérateurs $G_1, G_2, \cdots$ de type $\alpha \to \alpha$ aux termes initiaux $M_{00}, M_{01}, \cdots$ de type α on définit l'ensemble de ces habitants qui sont aussi de type α.

Définition 2.6. *(Grammaire associée aux types finement engendré de degré ≤ 3)*
Soit le type $A_1 \to (A_2 \cdots \to (A_n \to o))$
$B = \{x_i : A_i \ /\ pour\ i = 1, \cdots, n\} \cup \{y_j : o\ pour\ j = 1, \cdots, m\}$ *un contexte ou une base de type avec :*
$A_i = o$ *variable de type,* $A_i = o \to o$ *ou* $A_i = (o_1 \to (o_2 \to \cdots (o_m \to$

$o))) \rightarrow$ *o types composés* $(i = 1, \cdots, n)$*, on définit une grammaire à contexte libre* $G(B) =< V_T, V_N, o, P >$ *comme suit :*

- *L'ensemble des symboles terminaux* $V_T = \{(,), ., \lambda, x_1, \cdots, x_n, y_1, \cdots, y_m\}$
- *L'ensemble des symboles non-terminaux* $V_N = \{o, b_1, \cdots, b_p\} | b_i = (o_1 \rightarrow (o_2 \rightarrow \cdots (o_m \rightarrow o)))(m \geq 1)$ *et* $1 \leq i \leq p$
- *L'ensemble des règles de production* $P = \{o \rightarrow x_i | x_j o | x_n b\} \cup \{b \rightarrow \lambda y_1 \cdots y_m.o\}$ *est construit comme suit :*

Soit $\alpha = A_1 \rightarrow (A_2 \rightarrow \cdots (A_n \rightarrow o))$

1. *s'il existe* $A_i = o \in Arg(\alpha)$ *alors ajouter la règle de production* $o \rightarrow x_i$
2. *s'il existe* $A_i = (o \rightarrow o) \in Arg(\alpha)$ *alors ajouter la règle de production* $o \rightarrow x_i o$
3. *s'il existe* $A_i = (o_1 \rightarrow (o_2 \rightarrow \cdots \rightarrow (o_m \rightarrow o))) \rightarrow o \in Arg(\alpha)$*, ajouter les deux règles suivantes* $o \rightarrow x_i b$ *et* $b \rightarrow \lambda y_1 \cdots y_m.o$ *tel que* $b = o_1 \rightarrow (o_2 \rightarrow \cdots \rightarrow (o_m \rightarrow o))$

Quand le symbole initial (l'axiome) o est spécifié on note la grammaire G(B,o).

Exemple 2.3. $\alpha = o \rightarrow (o \rightarrow o) \rightarrow ((o \rightarrow (o \rightarrow o)) \rightarrow o) \rightarrow o$ *la grammaire à contexte libre* $G(\alpha)$ *est définie comme suit :*
$G(\alpha) = < V_N, V_T, o, P >$ *avec* $V_N = \{o, b\}$, $V_T = \{(,), ., \lambda, x_1, x_2, x_3, y_1, y_2\}$
$P = \{o \rightarrow x_1 | x_2 o | x_3 b \qquad b \rightarrow \lambda y_1 y_2.o\}$ $b : o \rightarrow (o \rightarrow o)$
La base ici est $B : \{x_1 : o, x_2 : o \rightarrow o, x_3 : (o \rightarrow (o \rightarrow o)) \rightarrow o\} \cup \{y_1 : o, y_2 : o\}$.

Définition 2.7. *(Procédure de génération (G3))*
Entrée : *Un type* $\alpha \equiv \alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow o | \alpha_i = o, (o \rightarrow o)$ *ou bien* $(o \rightarrow (o \rightarrow \cdots \rightarrow (o \rightarrow o))) \rightarrow o$.
Sortie : *Un ensemble fini de termes initiaux* $E_{M_0} = \{M_{01}, \cdots M_{0m}\}$ *et de générateurs* $E_G = \{G_1, \cdots, G_n\}$.

Etape 0 : $E_{M_0} = E_G = \emptyset$.
Si $o \to x_i \in P$ ***alors*** *construire le terme initial suivant :*
$M_{0i} = \lambda x_1 \cdots x_n.x_i$.
$E_{M_0} \cup \{M_{0i}\}$
Si $o \to x_k b$ *et* $b \to \lambda y_1 \cdots y_m.o \in P$***alors***
Si $o \to x_i o \in P$ ***alors*** *ajouter les termes initiaux suivants*
$M_{0it} = \lambda x_1 \cdots x_n.x_k(\lambda y_1 \cdots y_m.x_i y_t)$ *et* $M_{0t} = \lambda x_1 \cdots x_n.x_k\,(\lambda y_1 \cdots y_m.y_t)$
$E_{M_0} \cup \{M_{0it}, M_{0t}\}$ *et ajouter les deux générateurs* G_i *et* G_k
$G_i = \lambda f x_1 \cdots x_n.x_i(f x_1 \cdots x_n)$ *et* $G_k = \lambda f x_1 \cdots x_n.x_k(\lambda y_1 \cdots y_m.f x_1 \cdots x_n)$
$E_G \cup \{G_i, G_k\}$
Sinon *construire seulement les termes initiaux de cette forme :*
$M_{0t} = \lambda x_1 \cdots x_n.x_k(\lambda y_1 \cdots y_m.y_t)$
$E_{M_0} \cup \{M_{0t}\}$ *et les générateurs*
$G_k = \lambda f x_1 \cdots x_n.x_k(\lambda y_1 \cdots y_m.f x_1 \cdots x_n)$
$E_G \cup \{G_k\}$

Etape 1 :
Si $E_{M_0} = \emptyset$ ***alors*** *pas d'habitants pour ce type*
Sinon Si $E_G = \emptyset$ ***alors*** *l'ensemble des habitants est* E_{M_0}
Sinon *l'ensemble des habitants est E tel que :*
$E = E_{M_0} \cup \{G_1 M_{01}, \cdots, G_1(\cdots(G_1 M_{01})), \cdots, G_n(M_{0m}), \cdots, G_n(\cdots(G_n M_{0m}))\}$
$\cup \{G_p(\cdots(G_o M_{0k}))\} | k = 1, \cdots, m$ *et* $G_p, \cdots, G_o \in \{G_1, \cdots, G_n\}$

Etude des types par cas :

Cas 1 :
Tous les $A_i = o$ alors le type générale est le suivant :
$\alpha = o \to (o \cdots \to (o \to o))$
$Rank(\alpha) = 1$
la grammaire associée est :
$o \to x_1 | \cdots | x_n$
les habitants de ces types sont exactement les termes initiaux suivants :
$\prod_1^m = \lambda x_1 \cdots x_m.x_1, \cdots, \prod_n^m = \lambda x_1 \cdots x_m.x_n$
le terme général est le terme projecteur suivant : $\prod_{i=1}^m = \lambda x_1 \cdots x_m.x_i$

Exemple 2.4. *soit le type* $\alpha = o \to (o \to o)$
La grammaire à contexte libre G(α*) est définie comme suit :*
$G(\alpha) = < V_N, V_T, o, P >$ *avec* $V_N = \{o\}$, $V_T = \{x_1, x_2\}$
$P = \{o \to x_1 | x_2$ *On a la règle* $o \to x_1 \in P$ *alors on ajoute le terme initial*

$$\lambda x_1 x_2.x_1$$

de même pour la règle $o \to x_2$ *on ajoute le terme initial*

$$\lambda x_1 x_2 . x_2$$

Cas 2 :
Tous les $A_i = (o \to o)$ alors le type général est le suivant :
$\alpha = (o \to o) \to ((o \to o) \cdots ((o \to o) \to o))$
$Rank(\alpha) = 2$
La grammaire pour ce type est :
$o \to x_1 o | \cdots | x_n o$ c'est une grammaire infinie, ce type ne possède pas d'habitants(aucun terme initial).
Cas 3 :
Tous les $A_i = (o \to (o \to \cdots (o \to o))) \to o$ alors le type général est le suivant :
$\alpha = ((o \to (o \to \cdots (o \to o))) \to o) \to \cdots \to (((o \to (o \to \cdots (o \to o))) \to o) \to o)$
tel que :
$Rank(\alpha) = 3$
La grammaire pour ce type est :
$o \to x_1 b_1 | \cdots | x_n b_n$
$b_i \to \lambda y_1^o \cdots y_m^o . o \qquad i = 1, \cdots n$ (projection), alors pour cette grammaire on définit les termes initiaux suivants :

$$\begin{cases} M_{011} = \lambda x_1 \cdots x_n . x_1 (\lambda y_1 \cdots y_m . y_1) \\ \vdots \\ M_{01m} = \lambda x_1 \cdots x_n . x_1 \, (\lambda y_1 \cdots y_m . y_m) \end{cases}$$

Le générateur pour ces termes initiaux $(M_{011} \cdots M_{01m})$ est de la forme :
$G_1 = \lambda f x_1 \cdots x_n . x_1 \, (\lambda y_1 \cdots y_m . f x_1 \cdots x_n)$

$$\begin{cases} M_{021} = \lambda x_1 \cdots x_n . x_2 \, (\lambda y_1 \cdots y_m . y_1) \\ \vdots \\ M_{02m} = \lambda x_1 \cdots x_n . x_2 (\lambda y_1 \cdots y_m . y_m) \end{cases}$$

Le générateur pour ces termes initiaux $(M_{021} \cdots M_{02m})$ est de la forme :
$G_2 = \lambda f x_1 \cdots x_n . x_2 (\lambda y_1 \cdots y_m . f x_1 \cdots x_n)$

$$\vdots$$
$$\begin{cases} M_{0n1} = \lambda x_1 \cdots x_n . x_n (\lambda y_1 \cdots y_m . y_1) \\ \vdots \\ M_{0nm} = \lambda x_1 \cdots x_n . x_n (\lambda y_1 \cdots y_m . y_m)) \end{cases}$$

Le générateur pour ces termes initiaux $(M_{0n1} \cdots M_{0nm})$ est de la forme :
$G_n = \lambda f x_1 \cdots x_n . x_n \, (\lambda y_1 \cdots y_m . f x_1 \cdots x_n)$ tel que :
$y_i \in \{y_1 : o, \cdots, y_m : o\}$ ($i = 1, \cdots, m$) $\qquad x_n : (o \to (o \to \cdots (o \to o))) \to o$
$i = 1, \cdots, n$

On obtient m × n terme initial et n générateur
La classe des habitants peut être obtenue par application des $G_1, G_2, \cdots, G_n$ aux $M_{011}, \cdots, M_{01m}, M_{021}, \cdots, M_{02m}, \cdots, M_{01n}, \cdots, M_{0nm}$

Preuve :
Démontrons que si $M : \alpha$ alors $(G_i M) : \alpha$
Donc l'assignation de type à $(G_i M) : \alpha$ se fait selon la déduction

$$\frac{f : \alpha_1 \to \cdots \alpha_n \to o \quad x_1 : \alpha_1, \cdots, x_n : \alpha_n}{f x_1 \cdots x_n : o}(\to E)$$

On introduit les variables $y_1 : o, \cdots, y_m : o$ dans l'ordre on obtient :

$$\frac{y_1 : o, \cdots, y_m : o \quad f x_1 \cdots x_n : o}{\lambda y_1 \cdots y_m . f x_1 \cdots x_n : \underbrace{o \to \cdots \to o}_{m \text{ fois o}} \to o}(\to I)$$

on a $x_i : \alpha_i$ et d'après la construction de la grammaire, la variable x_i se trouve en tête d'une règle o, α_i a donc la forme :
$\alpha_i : \underbrace{o \to o \cdots \to o}_{m \text{ fois o}} \to o \to o$
Dans la suite de l'assignation de type au terme $(G_i M)$ se fera comme suit :

$$\frac{x_i : \overbrace{o \to \cdots \to o}^{\text{m fois o}} \to o \to o \quad \lambda y_1 \cdots y_m . f x_1 \cdots x_n : \overbrace{o \to \cdots \to o}^{\text{m fois o}} \to o}{x_i(\lambda y_1 \cdots y_m . f x_1 \cdots x_n) : o}$$

En déchargeant les hypothèses $x_1 : \alpha_i, \cdots, x_n : \alpha_n$ dans l'ordre nous obtenons :
$\lambda x_1, \cdots x_n . x_i(\lambda y_1 \cdots y_m . f x_1 \cdots x_n) : \alpha_1 \to \cdots \alpha_n \to o$
$G_i = \lambda f x_1, \cdots x_n . x_i(\lambda y_1 \cdots y_m . f x_1 \cdots x_n) : \alpha \to \alpha_1 \to \cdots \alpha_n \to o$
Et nous terminons l'assignation de type pour $(G_i M)$ comme suit :

$$\frac{G_i : \alpha \to \alpha_1 \to \cdots \alpha_n \to o \qquad M : \alpha}{(G_i M) : \alpha_1 \to \cdots \alpha_n \to o \equiv \alpha}(\to E) \qquad \square$$

Exemple 2.5. $\alpha = ((o \to (o \to o)) \to o) \to (((o \to (o \to (o \to o))) \to o) \to o)$
La grammaire associée a ce type de degré 3 est :
$o \to x_1 b_1 | x_2 b_2$
$b_1 \to \lambda y_1 y_2 . o$
$b_2 \to \lambda y_1 y_2 y_3 . o$
La liste des termes initiaux :
$M_{01i} = \lambda x_1 x_2 . x_1(\lambda y_1 y_2 . y_i)\ i = 1, 2$

$M_{02i} = \lambda x_1 x_2.x_2\,(\lambda y_1 y_2 y_3.y_i)$ $i = 1,2,3$ *termes initiaux*

$G_1 = \lambda f x_1 x_2.x_1\,(\lambda y_1 y_2.f x_1 x_2)$
$G_2 = \lambda f x_1 x_2.x_1\,(\lambda y_1 y_2 y_3.f x_1 x_2)$, *deux générateurs*

$$\begin{aligned} G_1 M_{011} &= (\lambda f x_1 x_2.x_1\,(\lambda y_1 y_2.f x_1 x_2))\,M_{011} \\ &= \lambda x_1 x_2.x_1\,(\lambda y_1 y_2.M_{011} x_1 x_2) \\ &= \lambda x_1 x_2.x_1\,(\lambda y_1 y_2.\,(\lambda x_1 x_2.x_1\,(\lambda y_1 y_2.y_1))\,x_1 x_2) \\ &= \lambda x_1 x_2.x_1\,(\lambda y_1 y_2.x_1\,(\lambda y_1 y_2.y_1)) \ \textit{ce terme est de type } \alpha \end{aligned}$$

Appliquons G_1 *au* M_{021} *:*
$G_1 M_{021} = (\lambda f x_1 x_2.x_1\,(\lambda y_1 y_2.f x_1 x_2))\,M_{021}$

$$\begin{aligned} &= \lambda x_1 x_2.x_1\,(\lambda y_1 y_2.M_{021} x_1 x_2) \\ &= \lambda x_1 x_2.x_1\,(\lambda y_1 y_2.\,(\lambda x_1 x_2.x_2\,(\lambda y_1 y_2 y_3.y_1))\,x_1 x_2) \\ &= \lambda x_1 x_2.x_1\,(\lambda y_1 y_2.x_2\,(\lambda y_1 y_2 y_3.y_1)) \end{aligned}$$

Le terme résultat est aussi de type α, les autres termes peuvent être calculés soit par application de G_1 et G_2 aux termes initiaux, soit par application au résultat de chaque application séparément ou composé, dans l'ordre ou d'une manière aléatoire comme suit :

$G_1\,(G_1 \cdots G_1\,(M_{01i}))$ ou i = 1,2

$G_2\,(G_2 \cdots G_2\,(M_{01i}))$ ou

$G_1\,(G_1 \cdots G_1\,(M_{02j}))$ j= 1,2,3 ou

$G_2\,(G_2 \cdots G_2\,(M_{02j}))$ ou

$G_i\,(G_i \cdots (M_{01t}))$ $i = 1,2$ et $t = 1,2$
$G_i\,(G_i \cdots (M_{02k}))$ $i = 1,2$ et $k = 1,2,3$

Cas 4 :
$A_i = o$ ou $A_i = (o \to (o \to \cdots \to (o \to o))) \to o$
alors $x_i : o$ ou $x_i : (o \to (o \to \cdots \to (o \to o))) \to o$
$\alpha = (o \to \cdots (((o \to (o \to \cdots \to (o \to o))) \to o) \to \cdots \to (((o \to (o \to \cdots \to (o \to o))) \to o) \to\ o)))$

La grammaire correspondante est :
$o \to x_i | \cdots | x_j b | \cdots$

$b \to \lambda y_1 \cdots y_m.o$

Les termes initiaux :
$M_{0i} = \lambda x_1, \cdots x_n.x_i$ tel que $x_i : o$
$M_{0jt} = \lambda x_1, \cdots x_n.x_j(\lambda y_1 \cdots y_m.y_t)$
tel que $y_t \in \{y_1 : o, \cdots, y_m : o\}$ $x_j : (o \to (o \to \cdots (o \to o))) \to o$

le générateur correspondant :
$G_j = \lambda f x_1, \cdots x_n.x_j\,(\lambda y_1 \cdots y_m.f x_1, \cdots x_n)$ tel que $x_j \in \{x : (o \to (o \to \cdots (o \to o))) \to o\}$

(même preuve que précédemment)

Exemple 2.6. $\alpha = (o \to (((o \to (o \to (o \to o))) \to o) \to (o \to (((o \to (o \to o)) \to o) \to o))))$

La base $B = \{x_1 : o, x_2 : (o \to (o \to (o \to o))) \to o, x_3 : o, x_4 : (o \to (o \to o)) \to o\}$
La grammaire :

$o \to x_1|x_2b_1|x_3|x_4b_2$
$b_1 \to \lambda y_1y_2y_3.o$
$b_2 \to \lambda y_1y_2.o$

Les termes initiaux :
$M_{01} = \lambda x_1x_2x_3x_4.x_1$
$M_{03} = \lambda x_1x_2x_3x_4.x_3$
$M_{02i} = \lambda x_1x_2x_3x_4.x_2\,(\lambda y_1y_2y_3.y_i)$ *tel que* $y_i : o \in \{y_1, y_2, y_3\}$
$M_{04j} = \lambda x_1x_2x_3x_4.x_4\,(\lambda y_1y_2.y_j)$ *tel que* $y_j : o \in \{y_1, y_2\}$

Les générateurs :

$$\begin{aligned} G_1 &= \lambda f x_1x_2x_3x_4.x_2\,(\lambda y_1y_2y_3.f x_1x_2x_3x_4) \\ G_2 &= \lambda f x_1x_2x_3x_4.x_4\,(\lambda y_1y_2.f x_1x_2x_3x_4) \end{aligned}$$

L'ensemble des habitants peuvent être obtenus par application de G_1, G_2 *aux* $M_{01}, M_{03}, M_{02i}, M_{04j}$
$G_1\,(G_1 \cdots (G_1M_{01}))$ *ou* $G_2\,(\cdots (G_2M_{01}))$
$G_t\,(G_t \cdots (G_tM_{02i}))$ $t = 1, 2$ *et i=1, 2, 3*
$G_t\,(G_t \cdots (G_tM_{03j}))$ $t = 1, 2$ *et* $j = 1, 2$

$$\begin{aligned}
(G_1 M_{01}) &= (\lambda f x_1 x_2 x_3 x_4.x_2(\lambda y_1 y_2 y_3.f x_1 x_2 x_3 x_4))M_{01} \\
&= \lambda x_1 x_2 x_3 x_4.x_2(\lambda y_1 y_2 y_3.x_1) \\
G_2(G_1 M_{01}) &= \lambda f x_1 x_2 x_3 x_4.x_4(\lambda y_1 y_2.f x_1 x_2 x_3 x_4)(G_1 M_{01}) \\
&= \lambda f x_1 x_2 x_3 x_4.x_4(\lambda y_1 y_2.(G_1 M_{01}) x_1 x_2 x_3 x_4) \\
&= \lambda x_1 x_2 x_3 x_4.x_4(\lambda y_1 y_2.(\lambda x_1 x_2 x_3 x_4.x_2(\lambda y_1 y_2 y_3.x_1)) x_1 x_2 x_3 x_4) \\
&= \lambda x_1 x_2 x_3 x_4.x_4(\lambda y_1 y_2.x_2(\lambda y_1 y_2 y_3.x_1))
\end{aligned}$$

$\vdots$

Cas 5 :
$A_i = (o \to o)$ ou $A_i = (o \to (o \to \cdots (o \to o))) \to o$
Le type général :
$\alpha = (o \to o) \to \cdots \to (((o \to (o \to \cdots (o \to o))) \to o) \to o)$

La grammaire est :
$o \to x_i o | \cdots | x_j b$
$b \to \lambda y_1 \cdots y_m.o$

Les termes initiaux :
$M_{0jt} = \lambda x_1, \cdots x_n.x_j(\lambda y_1 \cdots y_m.y_t)$ tel que $t = 1, \cdots, m$
$M_{0jit} = \lambda x_1, \cdots x_n.x_j(\lambda y_1 \cdots y_m.x_i y_t)$

Les générateurs correspondants :
$G_{1j} = \lambda f x_1 \cdots x_n.x_j(\lambda y_1 \cdots y_m.f x_1 \cdots x_n)$
$G_{1i} = \lambda f x_1 \cdots x_n.x_i(f x_1 \cdots x_n)$
$x_i : (o \to o)$ et $y_t : o \in \{y_1 : o, \cdots, y_m : o\}$ et $x_j : (o \to (o \to \cdots (o \to o))) \to o$ (même preuve que précédent)

Exemple 2.7. $((o \to o) \to ((o \to o) \to (((o \to (o \to (o \to o))) \to o) \to o)))$

La grammaire :
$o \to x_1 o | x_2 o | x_3 b$
$b \to \lambda y_1 y_2 y_3.o$
$M_{03t} = \lambda x_1 x_2 x_3.x_3(\lambda y_1 y_2 y_3.y_t)$
$M_{3it} = \lambda x_1 x_2 x_3.x_3(\lambda y_1 y_2 y_3.x_i y_t)$
tel que $x_i : (o \to o)/i = 1, 2$ *et* $y_t : o \in \{y_1 \cdots y_m\}$

Les générateurs :
$G_1 = \lambda f x_1 x_2 x_3.x_1\,(f x_1 x_2 x_3)$
$G_2 = \lambda f x_1 x_2 x_3.x_2\,(f x_1 x_2 x_3)$
$G_3 = \lambda f x_1 x_2 x_3.x_3\,(\lambda y_1 y_2 y_3.f x_1 x_2 x_3)$

Cas 6 :
$A_i = o$ ou $A_i = (o \to (o \to \cdots \to (o \to o))) \to o$
$\alpha = o \to \cdots \to (((o \to (o \to \cdots (o \to o))) \to o) \to o)$

La grammaire associée :
$o \to x_i | x_j b$
$b \to \lambda y_1 \cdots y_m.o$

Les termes initiaux :
$M_0 = \lambda x_1 \cdots x_n.x_i$
$M_{0jt} = \lambda x_1 \cdots x_n.x_j\,(\lambda y_1 \cdots y_m.y_t)$
$G_j = \lambda f x_1 \cdots x_n.x_j\,(\lambda y_1 \cdots y_m.f x_1 \cdots x_n)$
tel que $y_t : o \in \{y_1, \cdots, y_m\}$ et $x_j : (o \to (o \to \cdots \to (o \to o))) \to o$

Exemple 2.8. $\alpha = (o \to (((o \to (o \to (o \to o))) \to o) \to (o \to o)))$

La grammaire :
$o \to x_1 | x_2 b | x_3$
$b \to \lambda y_1 y_2 y_3.o$

Les termes initiaux :
$M_{01} = \lambda x_1 x_2 x_3.x_1$
$M_{03} = \lambda x_1 x_2 x_3.x_3$
$M_{02t} = \lambda x_1 x_2 x_3.x_2\,(\lambda y_1 y_2 y_3.y_t)$

Le seul générateur est :
$G = \lambda f x_1 x_2 x_3.x_2\,(\lambda y_1 y_2 y_3.f x_1 x_2 x_3)$.
Par la suite appliquons ce générateur aux termes initiaux pour définir l'ensemble des habitants

$$\begin{aligned} GM_{01} &= \lambda x_1 x_2 x_3.x_2\,(\lambda y_1 y_2 y_3.x_1)\,. \\ G\,(GM_{01}) &= \lambda x_1 x_2 x_3.x_2\,(\lambda y_1 y_2 y_3.x_2\,(\lambda y_1 y_2 y_3.x_1))\,. \\ GM_{02t} &= \lambda x_1 x_2 x_3.x_2\,(\lambda y_1 y_2 y_3.x_2\,(\lambda y_1 y_2 y_3.y_t)) \end{aligned}$$

$\vdots$

Cas 7 (englobe tous les cas précédents) :
$A_i = o$ ou $A_i = (o \to o)$ et $A_i = (o \to (o \to \cdots \to (o \to o))) \to o$

Le type général est :
$\alpha = o \to \cdots \to (o \to o) \to \cdots \to (((o \to (o \to \cdots (o \to o))) \to o) \to o)$
$o \to x_i | \cdots | x_j o | \cdots | x_n b$
$b \to \lambda y_1 \cdots y_m.o$
$M_{0i} = \lambda x_1 \cdots x_n.x_i$
$M_{0nt} = \lambda x_1 \cdots x_n.x_n\,(\lambda y_1 \cdots y_m.y_t)$
$M_{0njt} = \lambda x_1 \cdots x_n.x_n\,(\lambda y_1 \cdots y_m.x_j y_t)$
$G_n = \lambda f x_1 \cdots x_n.x_n\,(\lambda y_1 \cdots y_m.f x_1 \cdots x_n)$
$G_j = \lambda f x_1 \cdots x_n.x_j\,(f x_1 \cdots x_n)$ tel que :
$y_t : o \in \{y_1, \cdots, y_m\}$ et $x_j : (o \to o)$ et $x_n : (o \to (o \to \cdots \to (o \to o))) \to o$ et $x_i : o$

Exemple 2.9. $\alpha = (o \to ((o \to o) \to (((o \to (o \to (o \to o))) \to o) \to o)))$

la grammaire correspondante est :
$o \to x_1 | x_2 o | x_3 b$
$b \to \lambda y_1 y_2 y_3.o$

Les termes initiaux et les générateurs sont respectivement :
$M_{01} = \lambda x_1 x_2 x_3.x_1$
$M_{03t} = \lambda x_1 x_2 x_3.x_3\,(\lambda y_1 y_2 y_3.y_t)$
$M_{032t} = \lambda x_1 x_2 x_3.x_3\,(\lambda y_1 y_2 y_3.x_2 y_t)$

$G_1 = \lambda f x_1 x_2 x_3.x_3\,(\lambda y_1 y_2 y_3.f x_1 x_2 x_3)$
$G_2 = \lambda f x_1 x_2 x_3.x_2\,(f x_1 x_2 x_3)$

$$\begin{aligned} G_1 M_{02t} &= \lambda x_1 x_2 x_3.x_3\,(\lambda y_1 y_2 y_3.M_{032t} x_1 x_2 x_3) \\ &= \lambda x_1 x_2 x_3.x_3\,(\lambda y_1 y_2 y_3.x_3\,(\lambda y_1 y_2 y_3.x_2 y_t)) \\ G_2\,(G_1 M_{032t}) &= \lambda x_1 x_2 x_3.x_2\,(G_1 M_{032t})\,x_1 x_2 x_3 \\ &= \lambda x_1 x_2 x_3.x_2\,(x_3\,(\lambda y_1 y_2 y_3.x_3\,(\lambda y_1 y_2 y_3.x_2 y_t))) \end{aligned}$$

$\vdots$

Dans ([23] page 85) l'auteur a donné les $\lambda - termes\ P_n$ qui sont définis par :
$P_n = \lambda x.x(\lambda y_1.x(\cdots x(\lambda y_n.xI)\cdots))$ de type $\alpha = ((o \to o) \to o) \to o$, et le terme générateur $C_i = \lambda xy.y(\lambda z.xy)$ qui est un habitant en forme normale sans avoir comment les calculer.
Notre méthode (présenter précédemment $(G3)$) permet d'extraire la grammaire, les termes initiaux et les générateurs de ce type de degré 3 comme suit :

La grammaire associée est :
$o \to x_1 b$
$b \to \lambda y_1.o$
On a $A_1 = ((o \to o) \to o)$ alors on ajoute :
Le terme initial $M_{00} = \lambda x_1.x_1\,(\lambda y_1.y_1)$ et
Le générateur $G = \lambda f x_1.x_1\,(\lambda y_1.f x_1)$.
Par la suite l'ensemble des termes est défini par application de $G(\cdots(GM_{00}))$ (selon la procédure de génération)
Pour démontrer le cas 7 (général) on a besoin de ce lemme :

Lemme 2.1. *Soit* $B = \{x_1 : \alpha_1, \cdots, x_n : \alpha_n, y_1 : o, \cdots, y_m : o\}$ *avec degré* $\alpha_i \leq 2 \quad (i = 1, \cdots, n)$. *Pour tout* λ*-terme* M *en forme* β*-normale, et pour toute variable de type* $a \in \cup_{i=1}^{n} Vars(\alpha_i)$, *nous avons :*
$\grave{B} \vdash M : o \Longleftrightarrow M \in L(G(B, o))$ *où* $\grave{B} = \{x_i : \alpha_i \in B | x_i \in FV(M)\}$.

Preuve :
($\Leftarrow$)
Cas1 :

* $M = x_i$, $M \in L(G(B, o))$ pour $i \in \{1, \cdots, n\}$ donc $\alpha_i = o$ et G(B) contient la règle $o \to x_i$ d'où $x_i : o \in B$ et donc $\{x_i : o\} \vdash \{x_i : o\}$
* $M = x_k(\lambda y_1 \cdots y_m.x_j y_t)$ pour $k \in \{1, \cdots, n\}$ et $M \in L(G(B, o))$, $\alpha_k \equiv (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o$ et G(B) contient les règles : $o \to x_k b_k | x_j o$ et $b_k \to \lambda y_1 \cdots y_m.o$, B contient $x_k : (o_1 \to (o_2 \to \cdots \to (o_3 \to o))) \to o$ et $x_j : o \to o$ et $y_t : o \in \{y_1, \cdots, y_m\}$
  Nous avons alors $\{x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o\} \cup \{x_j : o \to o\} \cup \{y_1 : o, \cdots, y_t : o, \cdots y_m : o\} \vdash x_k(\lambda y_1 \cdots y_m.x_j y_t) : o$
  Suivant le schéma d'assignation de type :

$$\frac{x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o \quad y_1 : o_1, \cdots, y_m : o_m \quad x_j : o \to o}{x_k(\lambda y_1 \cdots y_m.x_j y_t) : o}$$

* $M = x_k(\lambda y_1 \cdots y_m.y_t)$ pour un certain $k = 1, \cdots, n$ $M \in L(G(B,o))$
  $\alpha_k \equiv (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o$ alors G(B) contient la règle $o \to x_k b_k$ et $b_k \to \lambda y_1 \cdots y_m.o$ et par conséquent nous avons :
  $x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o \in B$ et $y_1 : o_1, \cdots, y_t : o_t, \cdots, y_m : o_m \in B$ nous avons alors :

$$\frac{\{x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o\} \quad y_1 : o_1, \cdots y_t : o_t \cdots, y_m : o_m}{x_k(\lambda y_1 \cdots y_m.y_t) : o}$$

cas2 :

* $M = x_k(\lambda y_1. \cdots .y_m.M_k)$ puisque $M \in L(G(B, o))$ nous avons la dérivation à partir de o comme suit :

$\begin{cases} o \to x_k b_k \\ et \qquad \stackrel{*}{\Rightarrow} x_k(\lambda y_1 \cdots .y_m.M^{'}) \text{ alors } o \stackrel{*}{\Rightarrow} M^{'} \text{ donc G(B)} \\ b_k \to \lambda y_1 \cdots y_m.o \end{cases}$
contient les deux règles $o \to x_k b_k$ et $b_k \to \lambda y_1 \cdots y_m.o$ par conséquent $x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o_i))) \to o \in B$
$\{y_1 : o\}, \cdots, \{y_m : o\} \in B$
D'un autre coté nous avons $M^{'} \in L(G(B, o))$ par hypothèse d'induction nous avons

$$B^{'} \vdash M^{'} : o$$

Nous avons alors :
$M^{'} : o \cup \{y_1 : o_1\} \cup \cdots \cup \{y_m : o_m\} \cup \{x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to\} \vdash x_k(\lambda y_1 \cdots y_m.M^{'}) : o$ suivant le schéma d'assignation de type suivant :

$$\frac{B^{'} \vdash M^{'} : o \quad y_1 : o_1 \quad \cdots \quad y_m : o_m \quad x_k : (o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o}{x_k(\lambda y_1 \cdots y_m.M^{'}) : o}$$

* $M = x_j M^{'}$ pour un certain $j \in \{1, \cdots, n\} M \in L(G(B, o))$, $\alpha_j : o \to o$ alors G(B) contient la règle $o \to x_j o$ alors nous avons une dérivation de o comme suit :
$o \to x_j o \stackrel{*}{\Rightarrow} x_j M^{'}$ et $o \stackrel{*}{\Rightarrow} M^{'}$ par conséquent $x_j : o \to o \in B$
D'un autre coté $M^{'} \in M(L(G(B, o))$ par hypothèse d'induction nous avons

$$B^{'} \vdash M^{'} : o$$

Nous avons alors :
$M^{'} : o \qquad x_j : o \to o \vdash x_j M^{'} : o$
Suivant le schéma d'assignation de type suivant :

$$\frac{x_j : o \to o \qquad M^{'} : o}{x_j M^{'} : o}(\to E)$$

($\Rightarrow$) on démontre si $B^{'} \vdash M : o \Rightarrow L(G(B, o))$
cas1 :

* $M = x_i$ est une variable nous avons $B^{'} = \{x_i : o\}$ alors $\alpha_i = o$ donc G(B) contient la règle $o \to x_i$ et nous avons $x_i \in L(G(B, o))$
* $M = x_k(\lambda y_1 \cdots y_m.y_t)$ alors $B^{'} \vdash M : o$ donc $B^{'} \vdash x_k(\lambda y_1 \cdots y_m.y_t) : o \quad \alpha_k \equiv o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o$ suivant le schéma d'assignation de type

$$\frac{x_k : o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o \quad y_1 : o_1 \cdots y_m : o_m}{x_k(\lambda y_1 \cdots y_m.y_t) : o}$$

Donc nous avons
$x_k : o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o \in B$
et $\{y_1 : o\}, \cdots, \{y_m : o\} \in B$
G(B) contient les règles :

$$\begin{cases} o \to x_k b_k \\ et \\ b_k \to \lambda y_1 \cdots y_m . y_t \end{cases}$$

nous avons donc $x_k(\lambda y_1 \cdots y_m . y_t) \in L(G(B, o))$

* $M = x_k(\lambda y_1 \cdots y_m . x_j y_t)$ alors $B' \vdash M : o$ donc $B' \vdash x_k(\lambda y_1 \cdots y_m . x_j y_t)$
suivant le schéma d'assignation de type :

$$\frac{x_k : o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o \quad y_1 : o_1 \cdots y_t : o_t \cdots y_m : o_m \quad x_j : o \to o}{x_k(\lambda y_1 \cdots y_m . x_j y_t) : o}$$

Donc nous avons $x_k : o_1 \to (o_2 \to \cdots \to (o_m \to o))) \to o \in B$ et
$\{y_1 : o\} \cup \cdots \cup \{y_t : o\} \cup \cdots \cup \{y_m : o\} \in B$ et $x_j : o \to o \in B$
G(B) contient les règles

$$\begin{cases} o \to x_k b_k | x_j o \\ et \\ b_k \to \lambda y_1 \cdots y_m . o \end{cases}$$

et nous avons donc
$x_k(\lambda y_1 \cdots y_m . y_t) \in L(G(B, o))$

Cas 2 :

* $M = x_k(\lambda y_1 \cdots y_m . M')$ alors $B' \vdash M : o$ donc $B' \vdash x_k(\lambda y_1 \cdots y_m . M') :$
o suivant le schéma d'assignation de type nous avons :

$$\frac{x_k : o_1 \to (o_2 \to \cdots \to (o_m \to o_i))) \to o \quad y_1 : o_1 \cdots y_m : o_m \quad M' : o_i}{x_k(\lambda y_1 \cdots y_m . M') : o}$$

Donc nous avons $B_i \vdash M' : o_i$ d'un autre coté nous avons $x_k : o_1 \to (o_2 \to \cdots \to (o_m \to o_i)))$
et $y_1 : o_1 \cdots y_m : o_m \in B$
Par hypothèse d'induction, nous avons $M' \in L(G(B, o))$, G(B) contient
les règles :

$$\begin{cases} o \to x_k b_k | x_j o \\ et \\ b_k \to \lambda y_1 \cdots y_m . o \end{cases}$$

nous avons donc
$M = x_k(\lambda y_1 \cdots y_m.M') \in L(G(B,o))$

* $M_j = x_j M_j$ nous avons $\grave{B} \vdash M_j : o$ alors $\grave{B} \vdash x_j M_j$ suivant le schéma d'assignation de type nous avons

$$\frac{x_j : o \to o \quad M_j : o}{x_j M_j : o}(\to E)$$

Donc nous avons $B_j \vdash M_j : o_j$ d'un autre coté nous avons $x_j : o \to o \in B$, par hypothèse d'induction, nous avons $M_j \in L(G(B,o))$ G(B) contient la règle $o \to x_j o$ nous avons donc $M = x_j M_j \in (L(G(B,o))$

Théorème 2.1. *(voir [31] page 6) : Soit $\alpha = \alpha_1 \to \cdots \to \alpha_n \to o$ un type de degré 2, nous avons alors $Nhabs(\alpha) = \{N \leftarrow_\eta \lambda x_1 \cdots x_n.M\}$ pour un certain $M \in L(G(\{x_1 : \alpha_1, \cdots, x_m : \alpha_m, o\}))\}$.*

Preuve : ($\subseteq$) Sachant que la $\eta-$réduction préserve les types, il suffit de montrer $\vdash \lambda x_1 \cdots x_n.M : \alpha$ pour tout $M \in L(G(\alpha))$ d'après le lemme 2.1, nous avons.
$\{x_{i1} : \alpha_{i1}, \cdots, x_{ik} : \alpha_{ik}\} \vdash M : o$
où $FV(M) = \{x_{i1}, \cdots, x_{ik}\} \subseteq \{x_1, \cdots, x_n\}$
Par abstraction de $x_1 : \alpha_1, \cdots, x_n : \alpha_n$ dans cet ordre, on obtient
$\vdash \lambda x_1 \cdots x_n.M : \alpha_1 \to \cdots \to \alpha_n \to o$.
($\supseteq$)
Soit $N = \lambda x_1 \cdots x_p.xN_1 \cdots N_q \in Nhabs(\alpha)$, la déduction de $\vdash N : \alpha$ a la forme suivante :

$$\frac{\dfrac{xN_1 \cdots N_q : B}{x : o_1 \to \cdots o_q \to B \quad N_1 : o_1 \cdots N_q : o_q}(\to E)}{\lambda x_1 \cdots x_p.xN_1 \cdots N_q : \alpha_1 \to \cdots \alpha_p \to B}(\to E)$$

où $B = a_{q+1} \to \cdots \to a_{p+r} \to o$
$n = p + r$
$\alpha_{p+1} = a_{q+1}$
$\vdots$
$\alpha_{p+1} = a_{q+r} = \alpha_n$
Soit maintenant $x_{p+1}, \cdots, x_{p+r}$ de nouvelles variables de termes et $B = \{x_{i1} : \alpha_{i1}, \cdots, x_{in} : \alpha_{in}\}$ pour $FV(xN_1 \cdots N_q) = \{x_{i1}, \cdots, x_{in}\}$

$$\frac{\dfrac{xN_1 \cdots N_q : B \quad x_{p+1} = a_{q+1}, \cdots, x_{p+r} = a_{q+r}}{x : a_1 \to \cdots \to a_p \to B \qquad N_1 a_1 \cdots N_q : a_q}(\to E)}{xN_1 \cdots N_q x_{p+1} \cdots x_{p+r} : o}(\to E)$$

On a donc
$B \cup \{x_{p+1} = a_{q+1}, \cdots, x_{p+r} = a_{q+r}\} \vdash xN_1 \cdots N_q x_{p+1} \cdots x_{p+r} : o$ et d'après le lemme 2.1, $xN_1 \cdots x_q x_{p+1} \cdots x_{p+r} \in L(G(B, o))$ et puisque $x_{p+1}, \cdots, x_{p+r} \notin FV(xN_1 \cdots N_q)$, alors $\lambda x_1 \cdots x_p x_{p+1} \cdots x_{p+r}.xN_1 \cdots N_q x_{p+1} \cdots x_{p+r} \rightarrow_\eta \lambda x_1 \cdots x_p.xN_1 \cdots N_q$

2.3 Conclusion

La méthode $(G3)$ présentée ci-dessus donne une caractérisation des types finement engendrés de $deg \leq 3$ sous la forme $A_1 \rightarrow (A_2 \rightarrow \cdots \rightarrow (A_n \rightarrow o))$ tel que $A_i = o$, $A_i = (o \rightarrow o)$ ou
$A_i = (o \rightarrow (o \rightarrow \cdots (o \rightarrow o))) \rightarrow o \, (\mathrm{i} = 1, \cdots, n)$ (étudié par Thierry joly[25]).

Notre méthode est une extension de la méthode donnée dans [23], on a amélioré cette dernière par une procédure de calcul des termes initiaux et on a simplifié la procédure de calcul des générateurs pour les types de $deg \leq 2$, par la suite on a proposé une procédure $(G3)$ de calcul des termes initiaux et de générateurs pour une classe plus grande des types, les types finement engendrés de degré ≤ 3, la méthode proposée consiste à :

1. Extraire la grammaire de ces types.
2. Calculer les termes initiaux pour le type α.
3. Calculer les générateurs G_i pour le type α qui sont de type $\alpha \rightarrow \alpha$.

Par application de ces itérateurs $G_1, G_2, \cdots$ de type $\alpha \rightarrow \alpha$ aux termes initiaux $M_{00}, M_{01}, \cdots$ de type α on définit l'ensemble de ces habitants M_i qui sont aussi de type α .

3 Génération des habitants d'un type de degré quelconque

Dowek et Jiang (voir [19]) ont défini une grammaire à contexte libre pour énumérer les habitants d'un type simple en schéma de terme.
De notre part nous avons exploité cette grammaire pour construire un ensemble fini de termes (termes générateurs et termes initiaux) qui représente les habitants d'un type de degré quelconque.
La méthode proposée consiste à :

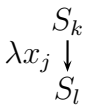
1. Ecrire la grammaire de Dowek à contexte libre (voir chapitre 3 page 49) pour un type donné $\alpha \equiv \alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow o$

2. Représenter cette grammaire sous forme d'un arbre de preuve.
3. Appliquer ensuite l'algorithme termes générateurs et termes initiaux.

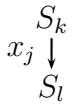
3.1 Procédure de construction d'un ensemble fini de termes

Arbre preuve de la grammaire de Dowek

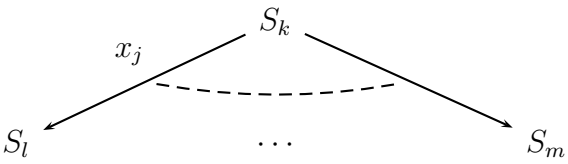
Soit le type $\alpha = \alpha_1 \to \cdots \to \alpha_n \to o$.
Entrée : Grammaire schéma de terme (Dowek).
Sortie : Arbre preuve T.
La règle de production $S_k \longrightarrow \lambda x_j S_l$ sera représentée par :



La règle de production $S_k \longrightarrow x_j S_l$ sera représentée par :

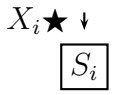


La règle de production $S_k \longrightarrow x_j S_l \cdots S_m$ sera représentée par :



La règle de production $S_k \longrightarrow x_j$ sera représentée par :

Après la construction de l'arbre, repérer les S_i de la grammaire de Dowek qui sont $S_{\Gamma \vdash o}$ et mettre ces noeuds dans un cadre, ajouter l'étiquette $\bigstar$ à gauche de la variable de l'arc entrant au noeud encadré.



tel que $\bigstar = (f x_1 \cdots x_n)$ (x_i variables de type α_i).

Algorithme termes générateurs et termes initiaux

Entrée : Arbre preuve T.
Sortie : Un ensemble $E = \{G_1^{l_1}, \cdots, G_m^{l_m}, M_1^{l'_1}, \cdots, M_k^{l'_k}\}$ fini de termes.

Si $T \equiv$

$$\begin{array}{c} S_k \\ \downarrow \\ x_j \end{array}$$

Alors "pas de termes à générer".
Etape 0 :

Tant que $\exists T' \in T$ tel que :

$T' \equiv$

$$\begin{array}{rl} & S \\ \lambda x_1 & \downarrow \\ & S_1 \\ & \vdots \\ \lambda x_n & \downarrow \\ & S_n \\ Y_1 & \downarrow \\ & \vdots \\ & S_j \\ Y_k \bigstar & \downarrow \\ & \boxed{S_{n+k}} \end{array}$$

Ajouter le générateur $G_1^{0 \in l}$:

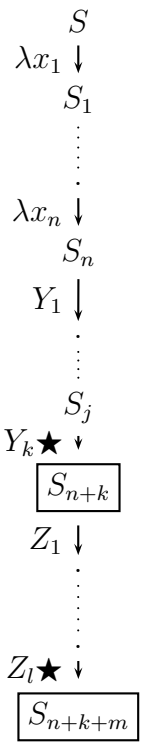
$$E \cup \{G_1^{0 \in l} = \lambda f x_1 \cdots x_n . Y_1 \cdots Y_k \bigstar\}$$

Tel que :
S_{n+k} est le premier noeud encadré trouvé en parcourant l'arbre de la racine S.
l : Ensemble des indices des générateurs crées en parcourant S_{n+k} vers le prochain noeud encadré immédiat suivant l'Etape I et I+1 .
$\bigstar = (f x_1 \cdots x_n)$.
Y_i : variables libres ou liées (pour $2 \leq i \leq k$) et Y_1 variable libre.

x_i : Sont des variables de type α_i pour $1 \leq i \leq n$.
f : est la fonction qui permet de générer les autres termes, elle représente l'arc entrant au le noeud encadré.
Etape I :

Tant que $\exists T' \in T$ tel que :

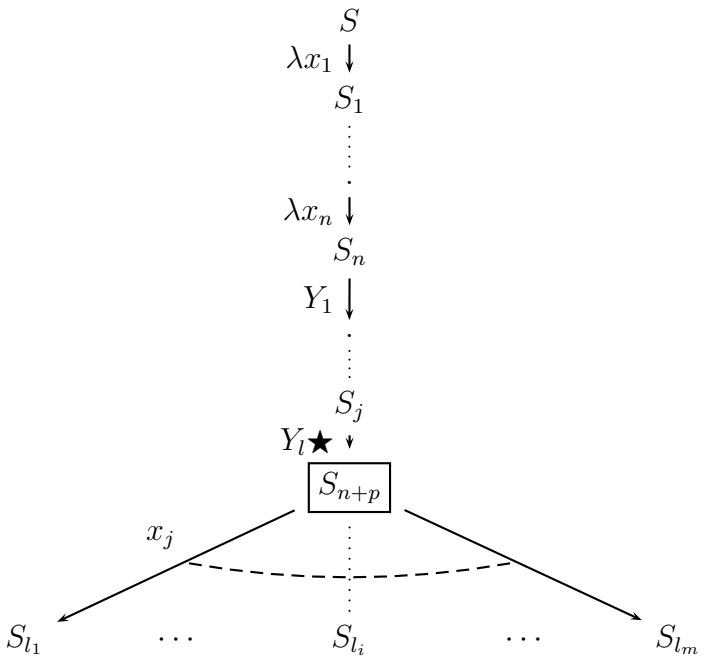
$T' \equiv$



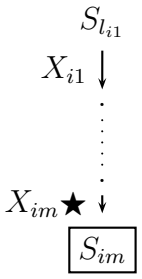
Ajouter le générateur $G_j^{l'}$:

$$E \cup \{G_j^{l'} = \lambda f x_1 \cdots x_n . Z_1 \cdots Z_l \bigstar\}$$

l' :Ensemble des indices des générateurs crées en parcourant S_{n+k+m} vers le prochain noeud encadré suivant l'Etape I et I+1.
Etape I+1 :
Tant que $\exists T' \in T$ tel que :
$T' \equiv$



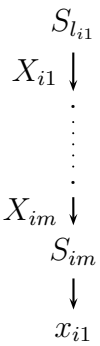
et $\exists S_{li}(i = 1, \cdots, m)$ tel que
Si $S_{li} \equiv$



Ajouter les générateurs $G_j^{l''}$ sous la forme suivante :

$$E \cup \{G_j^{l''} = \lambda f_1 \cdots f_r x_1 \cdots x_n . x_j \delta_{l_1} \cdots \overbrace{(X_{i1} \cdots X_{im}(f_i x_1 \cdots x_n))}^{\delta_{l_i}} \cdots \delta_{l_m}\}$$

tel que $f_i \in \{f_1, \cdots, f_r\}$.
Si $S_{li} \equiv$



Alors remplacer δ_{li} par la séquence finie suivante

$$X_{i1} \cdots X_{im} x_{i1}$$

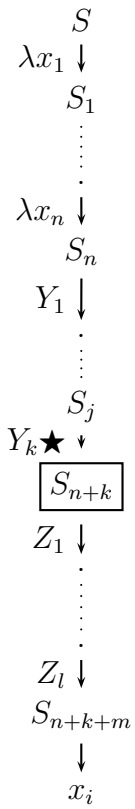
l'' : Ensemble des indices des générateurs crées en parcourant le noeud encadré S_{im} du sous-arbre S_{li} $(i = 1, \cdots, m)$ vers le prochain noeud encadré suivant les Etapes I et I+1, les indices sont de la forme k_i (k indice du générateur G_k et i représente le sous-arbre S_{li}).

Remarque 3.1. *Si le sous arbre S_{n+p} ne contient aucun noeud encadré le terme générateur $G_j^{l''}$ va se transformer en terme initial $M_j^{l''}$ tel que :*

$$E \cup \{M_j^{l''} = \lambda x_1 \cdots x_n . x_j \delta_{l_1} \cdots \overbrace{(X_{i1} \cdots X_{im})}^{\delta_{l_i}} \cdots \delta_{l_m}\}$$

l'' : ensemble des indices des générateurs crées pour le noeud encadré S_{n+p}

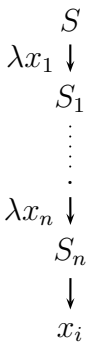
Etape I+2 :
Etape (a) tant que $\exists T' \in T$ tel que :
$T' \equiv$



Ajouter le terme initial M_i^l :

$$E \cup \{M_i^l = \lambda x_1 \cdots x_n . Z_1 \cdots Z_l x_i\}$$

l : ensemble des indices des générateurs crées pour le noeud encadré S_{n+k} qui existe immédiatement en dessus de x_i.
Etape (b) Tant que $\exists T' \in T$ tel que :
$T' \equiv$



Ajouter le terme initial M_i^0 :

$$E \cup \{M_i^0 = \lambda x_1 \cdots x_n . x_i\}$$

Ordre d'application des générateurs et de termes initiaux

- Si G_i^l un générateur crée suivant l'algorithme précédent, alors pour chaque séquence d'application de générateurs et de termes doit commencer par au moins un G_i^l tel que $0 \in l$.
- Si G_i^l et $G_j^{l'}$ deux générateurs crées, G_i^l peut être appliqué au résultat de l'application du générateur $G_j^{l'}$ (par exemple $G_i(G_j \cdots)$) ssi $j \in l$.
- Si $G_j^{l'}$ et M_i^l un générateur et un terme initial crées suivant l'algorithme et $j \in l$ alors pour chaque séquence d'application de générateurs au terme M_i^l, $G_j^{l'}$ doit exister au moins une seule fois avant M_i^l.

Remarque 3.2. *Toute séquence d'application de terme générateurs et de termes initiaux doit commencer par un* $G_i^{0 \in l}$ *et se termine par* $M_j^{l'}$.
Si M_i^l *est un terme initial crée suivant l'algorithme et* $0 \in l$ *alors ce terme sera ajouté a l'ensemble des habitants générés.*

On démontre que :

l'arbre-preuve $L(A) \Leftrightarrow L(G)$ tel que :

$L(G) = \{M | S \xrightarrow{*} M \ et \ M \in V_T^*\}$
$L(A) = \{M | M \equiv etiq_0 \cdots etiq_{in}\}$
L'arbre-preuve est représenté par

$$S \underset{etiq0}{\rightarrow} S_1 \underset{etiq1}{\rightarrow} \cdots \underset{etiqin-1}{\rightarrow} S_{in-1} \rightarrow etiq_{in}$$

S la racine de l'arbre.
$S_1 \cdots S_{in-1}$ les noeuds intérieurs de l'arbre.
$\rightarrow$ représente l'arc de l'arbre.
$etiq_0$ variable liée (λx_1).
$etiq_1 \cdots etiq_{in-1}$ variables libres ou liées $(\lambda x_i, x_i)$.
$etiq_{in}$ variable libre (x_i).

Preuve On démontre par induction sur la longueur de dérivation $S \xrightarrow{*} M$.
Base d'induction $l = 2$
$S \xrightarrow{2} M$, $M \equiv \lambda xx$ alors G contient les deux règles :
$S \rightarrow \lambda xS$
$S \rightarrow x$
$M \equiv etiq_0 etiq_1$,
$etiq_0 \equiv \lambda x$
$etiq_1 \equiv x$

$$\begin{array}{c} S \\ \lambda x \downarrow \\ S_1 \\ \downarrow \\ x \end{array}$$

par conséquence $M \in L(A)$
Hypothèse d'induction $S \xrightarrow{*} M$ est $l = n$, tous les termes engendrés par application de $(n-1)$ règles de production sont tel que $M \in L(G), M \in L(A)$
Cas 1 :
$S \xrightarrow{*} \lambda x_1 \cdots x_m S_j$ est de longueur $n-1$, $M \in L(G)$ *et* $M \in L(A)$, G contient les règles

$$\text{(n-1) règles} \quad \begin{cases} S & \rightarrow & \lambda x_1 S_1 \\ \vdots & & \\ S_{n-2} & \rightarrow & \lambda x_{n-1} S_{n-1} \end{cases}$$
$$S_{n-1} \rightarrow x_i \in L(A)$$

$M \equiv etiq_0 \cdots etiq_{n-1}$.
$etiq_0 \equiv \lambda x_1$
$\vdots$
$etiq_{n-2} \equiv \lambda x_{n-1}$
$etiq_{n-1} \equiv x_i$
$S \xrightarrow{*} \lambda x_1 \cdots x_m S_j \in L(A)$ pour une longueur $l = n-1$ (hypothèse d'induction) et $S_{n-1} \rightarrow x_i \in L(A)$ par conséquence $M \equiv \lambda x_1 \cdots x_{n-1}.x_i \in L(A)$ pour $l = n$
Cas 2 : $S \xrightarrow{*} \lambda x_1 \cdots x_m x_i M_{i1} \cdots S_{ij} \cdots M_{im}$ est de longueur $n-1$, $M \in L(G)$ *et* $\in L(A)$ (hypothèse d'induction), et on démontre pour une longueur $l = n$, G contient les règles

$$\text{p règles} \quad \begin{cases} S & \rightarrow & \lambda x_1 S_1 \\ \vdots & & \\ S_{m-1} & \rightarrow & \lambda x_m S_m \\ S_m & \rightarrow & x_i S_{i1} \cdots S_{im} \\ \vdots & & \end{cases}$$
$$S_{ij} \rightarrow y$$
$$\text{(n-p-1)règles} \quad \begin{cases} \vdots & & \\ S_{im+l} & \rightarrow & x_{im} \end{cases}$$

$M \equiv etiq_0 \cdots etiq_{n-1}$

$$
\begin{array}{lcl}
etiq_0 & \equiv & \lambda x_1 \\
\vdots & & \\
etiq_{m-1} & \equiv & \lambda x_m \\
etiq_m & \equiv & x_i \\
\vdots & & \\
etiq_{p-1} & \equiv & y \\
\vdots & & \\
etiq_{n-1} & \equiv & x_{im}
\end{array}
$$

$S \xrightarrow{*} \lambda x_1 \cdots x_m x_i M_{i1} \cdots S_{ij} \cdots M_{im} \in L(A)$ pour une longueur $n-1$ (hypothèse d'induction) et $S_{ij} \to y \in L(A)$ alors $\lambda x_1 \cdots x_m x_i M_{i1} \cdots y \cdots M_{im} \in L(A)$ de longueur $l = n$.

Cas 3

$S \xrightarrow{*} M$ tel que $M \in L(A)$ *et* $M \in L(G)$ pour $l = n-1$ (hypothèse d'induction)

$S \xrightarrow{*} \lambda x_1 \cdots x_m x_i M_{i1} \cdots (\lambda y S_p) \cdots M_{im}$, G contient les règles

$$
\begin{array}{ll}
\text{p règles} & \left\{ \begin{array}{lcl}
S & \to & \lambda x_1 S_1 \\
\vdots & & \\
S_{m-1} & \to & \lambda x_m S_m \\
S_m & \to & x_i S_{i1} \cdots S_{im} \\
\vdots & & \\
S_{ij-1} & \to & \lambda y S_{ij}
\end{array} \right. \\
 & S_{ij} \to y \\
\text{(n-p-1) règles} & \left\{ \begin{array}{lcl}
\vdots & & \\
S_{im+l} & \to & x_{im}
\end{array} \right.
\end{array}
$$

$M \equiv etiq_0 \cdots etiq_{n-1}$

$$
\begin{array}{lcl}
etiq_0 & \equiv & \lambda x_1 \\
\vdots & & \\
etiq_{m-1} & \equiv & \lambda x_m \\
etiq_m & \equiv & x_i \\
\vdots & & \\
etiq_{p-1} & \equiv & y \\
\vdots & & \\
etiq_{n-1} & \equiv & x_{im}
\end{array}
$$

$M \equiv \lambda x_1 \cdots x_m x_i M_{i1} \cdots (\lambda y S_p) \cdots M_{im} \in L(A)$ $l = n-1$ et $S_p \equiv S_{ij} \to y \in L(A)$ alors $M \equiv \lambda x_1 \cdots x_m x_i M_{i1} \cdots (\lambda y y) \cdots M_{im} \in L(A)$ pour une longueur

$l = n$.
Cas 4 $S \overset{*}{\rightarrow} M$ tel que $M \in L(A)$ *et* $M \in L(G)$ pour une longueur $l = n-1$ (hypothèse d'induction).
$S \overset{*}{\rightarrow} \lambda x_1 \cdots x_m x_i M_{i1} \cdots \lambda y_1 \cdots y_r S_p \cdots M_{im} \in L(A), L(G)$, G contient les règles

$$\text{p règles} \begin{cases} S & \rightarrow & \lambda x_1 S_1 \\ \vdots & & \\ S_{m-1} & \rightarrow & \lambda x_m S_m \\ S_m & \rightarrow & x_i S_{i1} \cdots S_{im} \\ \vdots & & \\ S_{ij-r} & \rightarrow & \lambda y_1 S_{ij-r+1} \\ \vdots & & \\ S_{ij-2} & \rightarrow & \lambda y_{r-1} S_{ij-1} \\ S_{ij-1} & \rightarrow & \lambda y_r S_{ij} \end{cases}$$

$$S_{ij} \rightarrow y$$

$$\text{(n-p-1) règles} \begin{cases} \vdots & \\ S_{im+l} & \rightarrow & x_{im} \end{cases}$$

$M \equiv etiq_0 \cdots etiq_p \cdots etiq_{n-1}$.

$$\begin{array}{lcl} etiq_0 & \equiv & \lambda x_1 \\ \vdots & & \\ etiq_{m-1} & \equiv & \lambda x_m \\ etiq_m & \equiv & x_i \\ \vdots & & \\ etiq_{p-r} & \equiv & \lambda y_1 \\ \vdots & & \\ etiq_{p-1} & \equiv & \lambda y_r \\ etiq_p & \equiv & y \\ \vdots & & \\ etiq_{n-1} & \equiv & x_{im} \end{array}$$

$S \overset{*}{\rightarrow} M$, $M \equiv \lambda x_1 \cdots x_m x_i M_{i1} \cdots (\lambda y_1 \cdots y_r S_p) \cdots M_{im} \in L(A)$ pour une longueur $l = n-1$
$S_p \rightarrow y \in L(A)$(*tel que* $ij = p$) alors $M \equiv \lambda x_1 \cdots x_m x_i M_{i1} \cdots (\lambda y_1 \cdots y_r y) \cdots M_{im}$, par conséquence $M \in L(A)$ pour $l = n$.

Exemple 3.1. *Soit le type des entiers* $A = (a \rightarrow a) \rightarrow a \rightarrow a$

1. ***Grammaire de Dowek :***
   La grammaire de Dowek G(T,N,R, $S_{\Gamma\vdash A}$) pour ce type est :

   - $\{(,),\lambda\} \in T$.
   - $S_{\Gamma\vdash A} \in N$.
   - *Les règles de productions :*

   $$S_{\Gamma\cup\vdash((a\to a)\to a\to a)} \to \lambda x \underbrace{S_{\emptyset\cup\{a\to a\}\vdash a\to a}}_{S_1} \quad |\{x\}\cup T \text{ et } \{S_1\}\cup N$$

   $$\underbrace{S_{\emptyset\cup\{a\to a\}\vdash a\to a}}_{S_1} \to \lambda y \underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} \quad |\{y\}\cup T \text{ et } \{S_2\}\cup N$$

   $$\underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} \to y$$

   $$\underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2} \to x \underbrace{S_{\emptyset\cup\{a\to a\}\cup\{a\}\vdash a}}_{S_2}$$

   Alors :

   $$\begin{aligned} S &\to \lambda x S_1 \\ S_1 &\to \lambda y S_2 \\ S_2 &\to y \\ S_2 &\to (x S_2) \end{aligned}$$

2. ***Construction de l'arbre preuve :***
   Nous construisons d'abord l'arbre de cette grammaire.
   Nous avons $S \to \lambda x S_1$ alors on ajoute

   $$\begin{array}{c} S \\ \lambda x_1 \downarrow \\ S_1 \end{array}$$

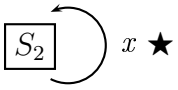
   Nous avons $S_1 \to \lambda y S_2$ alors on ajoute le sous arbre

   $$\begin{array}{c} S_1 \\ \lambda y \downarrow \\ S_2 \end{array}$$

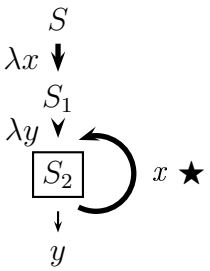
   Et la règle $S_2 \to y$ sera représentée par :

   $$\begin{array}{c} S_2 \\ \downarrow \\ y \end{array}$$

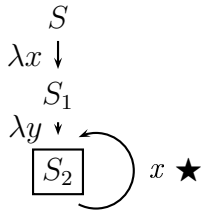
   On représente la règle $S_2 \to (x S_2)$ par



tel que $\bigstar = (fxy)$
Le noeud S_2 *sera encadré car il représente la règle de production* $\underbrace{S_{\emptyset \cup \{a \to a\} \cup \{a\} \vdash a}}_{S_2}$
qui génère des atomes.
L'arbre final est :



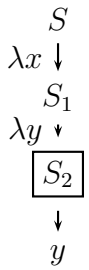
3. ***Génération des termes (termes initiaux et générateurs) :***
Il y'a un seul chemin de la racine S *vers le noeud encadré* S_2 *qui est représenté par :*



Cet sous arbre permet de créer le générateur $G_1^{\{0,1\}}$ *tel que :*

$$G_1^{\{0,1\}} = \lambda \boldsymbol{f} xy.x(\boldsymbol{f} xy)$$

Le parcours de l'arbre



de la racine vers la variable y *permet de générer le terme initial* $M_1^{\{0,1\}}$ *:*

$$M_1^{\{0,1\}} = \lambda xy.y$$

l'ensemble fini de termes est $E = \{G_1^{\{0,1\}}, M_1^{\{0,1\}}\}$

Pour générer l'ensemble des habitants, la séquence de l'application doit commencer par G_1^l *puisque* $0 \in l$ *et on applique* G_1 *à* G_1 *autant de fois (n fois) puisque* $1 \in l$*, pour* $M_1^{l'}$ *est un terme initial qui sera ajouté à l'ensemble des habitants puisque* $0 \in l'$ *et ce terme initial sera précédé par au moins un* G_1^l *puisque* $1 \in l'$*, la séquence de l'application des générateurs aux termes initiaux est donnée comme suit :*

$$(\underbrace{G_1 \cdots (G_1}_{nfois} M_1))$$

Si on applique G_1 *une seule fois à* M_1
$\underbrace{(\lambda fxy.(fxy))}_{G_1}\underbrace{(\lambda xy.y)}_{M_1}$ *on aura* $\lambda xy.xy$.
Et si on applique G_1 *à* M_1 *n fois on aura le terme* $\lambda xy.\underbrace{(x \cdots (x}_{nfois} y))$

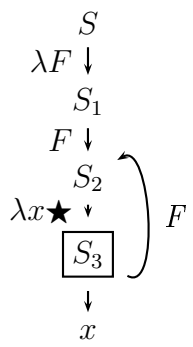
Exemple 3.2. *les règles de production du type* $A \equiv ((T \to T) \to T) \to T$ *(la grammaire de Dowek) sont :*

$$\begin{array}{rcl} S & \to & \lambda F S_1 \\ S_1 & \to & (F S_2) \\ S_2 & \to & \lambda x S_3 \\ S_3 & \to & x \\ S_3 & \to & (F S_4) \\ S_4 & \to & \lambda x S_3 \end{array}$$

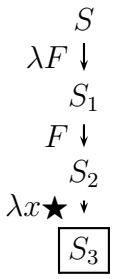
Tel que :

$$\begin{array}{rcl} S & = & S_{\vdash((T\to T)\to T)\to T} \\ S_1 & = & S_{(T\to T)\to T \vdash T} \\ S_2 & = & S_{(T\to T)\to T \vdash T\to T} \\ S_3 & = & S_{(T\to T)\to T, T \vdash T} \\ S_4 & = & S_{(T\to T)\to T, T \vdash T\to T} \end{array}$$

La représentation de ces règles de production sous forme d'un arbre est :



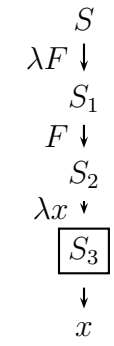
Le premier chemin de la racine S vers le premier noeud encadré S_3



Permet de créer le générateur $G_1^{\{0,1\}}$

$$G_1^{\{0,1\}} = \lambda \boldsymbol{f} F.F(\lambda x.\boldsymbol{f} F)$$

Tel que $\bigstar = (fF)$
Et le chemin de la racine vers la variable feuille x



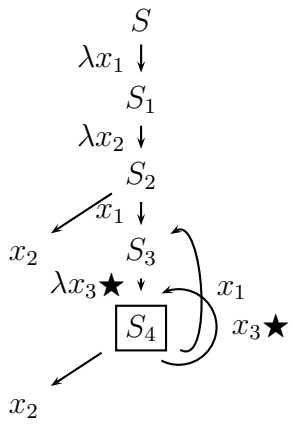
permet de créer le terme initial $M_1^{\{1\}}$ *:*

$$M_1^{\{1\}} = \lambda F.x$$

Exemple 3.3. *Soit le type Monster* $M \equiv (((o \to o) \to o) \to o) \to o \to o$.
La grammaire de Dowek permet de générer les règles de productions suivantes :

$S \to \lambda x_1 S_1$
$S_1 \to \lambda x_2 S_2$
$S_2 \to x_2 \mid x_1 S_3$
$S_3 \to \lambda x_3 S_4$
$S_4 \to x_2 \mid x_1 S_3 \mid x_3 S_4$

L'arbre de preuve correspondant à ces règles de production est :



tel que $\bigstar = (f x_1 x_2)$

1. *Création des générateurs :*

   Le parcours de l'arbre de la racine S vers le premier noeud encadré immédiat S_4 *(*S_4 *le noeud encadré représente la règle de production qui permet de générer des atomes) permet de construire le générateur* $G_1^{\{0,1,2\}}$ *tel que :*

   $$G_1^{\{0,1,2\}} = \lambda\ \boldsymbol{f} x_1 x_2 . x_1 (\lambda x_3 . (\boldsymbol{f} x_1 x_2))$$

   Le chemin du noeud encadré S_4 *vers* S_4 *permet de créer le générateur* $G_2^{\{1,2\}}$ *tel que :*

   $$G_2^{\{1,2\}} = \lambda\ \boldsymbol{f} x_1 x_2 . x_1 . x_3 (\boldsymbol{f} x_1 x_2)$$

2. *Création de termes initiaux :*

   Le parcours de l'arbre de la racine vers la variable feuille x_2 *possédant comme noeud père* S_2 *permet de créer le terme initial* $M_1^{\{0\}}$*.*

   $$M_1^{\{0\}} = \lambda x_1 x_2 . x_2$$

   et de la racine vers la feuille variable x_2 *possédant comme noeud père* S_4 *permet de créer le terme initial* $M_1^{\{1,2\}}$ *suivant l'Etape I+2 de l'algorithme termes générateurs et termes initiaux tel que :*

   $$M_1^{\{1,2\}} = \lambda x_1 x_2 . x_2$$

   puisque c'est le même terme initial alors on écrit :

   $$M_1^{\{0,1,2\}} = \lambda x_1 x_2 . x_2$$

L'ensemble fini des termes $\{G_1^{\{0,1,2\}}, G_2^{\{1,2\}}, M_1^{\{0,1,2\}}\}$ *permet de générer les habitants de type* $M \equiv (((o \to o) \to o) \to o) \to o \to o$*.*

Conclusion et perspectives

Nous avons présenté dans ce mémoire un état de l'art concernant les algorithmes qui énumèrent les habitants d'un type, comme Hirokawa, Takahashi, Zaionc, Broda et Damas, Ben-yelles et Hindely, Dowek et Jiang.
Nous avons proposé en premier lieu une grammaire à contexte libre et un algorithme qui se base sur la gestion de contexte appelé $Mat(\alpha, 0)$ (voir page 55).
Nous avons simplifié et enrichi la procédure donnée par Hemdani (voir [23]) puis nous avons proposé une méthode qui traite une classe plus grande des types, les types finement engendrés de degré ≤ 3. Cette méthode se base sur la notion des générateurs (itérateurs) et les termes initiaux, elle consiste à :

1. Définir une grammaire pour les types finement engendré de degré ≤ 3.
2. Définir une procédure qui permet de calculer l'ensemble les termes initiaux $M_{00}, M_{01}, \cdots$ et de générateurs $G_1, G_2, \cdots$.
3. Générer l'ensemble des habitants d'un type par application des termes générateurs aux termes initiaux et aux termes résultat de cette application.

De notre part nous avons été amené à développer :

- Une application qui répond à nos objectifs, génération des habitants des types finement engendrés de degré ≤ 3 (voir page 93).

Enfin nous avons exploité la grammaire à contexte libre de Dowek et Giang (voir [20]) et on a proposé un algorithme qui permet de définir un ensemble fini de termes (termes générateurs et termes initiaux) pour énumérer l'ensemble des habitants pour les types simple de degré quelconque, suivant l'importance de ce sujet dans le domaine de la recherche nous proposons d'appliquer les résultats obtenus au problème de la décidabilité de Ticket-logic.

Annexe A

Programme de génération des termes initiaux, générateurs pour les types finement engendrés de degré ≤ 3 "gen.ml"

```
open Genlex;;
exception  Error;;
exception END;;

type typ =
Atom of string
|Comp of typ * typ;;

let lista=[];;
let listv=[];;
let list_brut=[];;

(*génération des variables*)
let (genvar, resetvar) =
 let index = ref (-1) in
 (
  (function () -> index:=!index+1;

                  "x" ^ (string_of_int (!index))),

  (function () -> index:=-1)
 )
```

```
;;
let (genvary, resetvary) =
 let index = ref (-1) in
 (
  (function () -> index:=!index+1;

                  "y" ^ (string_of_int (!index))),

  (function () -> index:=-1)
 )
;;

let (genvarb, resetvarb) =
 let index = ref (-1) in
 (
  (function () -> index:=!index+1;

                  "b" ^ (string_of_int (!index))),

  (function () -> index:=-1)
 )
;;

let (genvarf, resetvarf) =
 let index = ref (-1) in
 (
  (function () -> index:=!index+1;

                  "f" ^ (string_of_int (!index))),

  (function () -> index:=-1)
 )
;;

let lexer  s =
         let l1 = make_lexer [ "("; ")"; "->"]
             in l1 (Stream.of_string s) ;;

let  rec lire_prop  f  =  prop2  f
and prop0 = parser
[< 'Ident s >]  ->  Atom s
```

```
| [< 'Kwd "(" ; p = lire_prop ;  'Kwd ")" >]  -> p

and   prop1  =  parser


|[< p = prop0  >] -> p
and prop2  = parser
[< p = prop1 ;  q = (reste p)  >]  -> q
and reste p  =  parser
[< 'Kwd "->"; q = prop1; r = (reste q) >]   -> Comp(p,r)
| [< >] -> p
;;
(*présentation du type sous forme d'une liste*)
let rec tta typ lista =
match typ with
|Atom a->List.rev((genvar (),Atom a)::lista)
|Comp (a1, a2) ->tta a2 ((genvar(),a1)::lista);;

let rec elimin_der lista list_brut=
   match lista with
|(x,a)::[]->List.rev(list_brut)
|(x,a)::l-> elimin_der l ((x,a)::list_brut);;

(*création des variables pour chaque argument du type *)
let rec var_typ lista =
match lista with
|[]->""
|(x,a)::l-> print_string x ;var_typ l;;

(*génération des variables y pour les types finement engendrés de degré <= 3*)
let rec affichy ty=
match ty with
|Atom a -> print_string "."
|Comp(Atom a1,Atom a2)-> print_string (genvary())
|Comp(b,Atom c)->affichy b
|Comp(Atom a1,Comp(a2,a3))->print_string (genvary());affichy (Comp(a2,a3))
|_->raise Error;;

(*génération des règles de production pour les types degré <=3*)
let rec affich_rg_pr_oo lista =
match lista with
```

```
|[]-> print_string ""
|(x,Comp (Comp (a,b),Atom c))::l-> print_string c; print_string "->";
print_string x;let s=(genvarb()) in print_string (s) ; print_string "|";
print_string s;print_string "->";print_string "\\";affichy(Comp (a,b));
resetvary();print_string ".";print_string c;print_newline(); affich_rg_pr_oo l
|(x,Atom a)::l-> print_string a; print_string "->"; print_string x;
print_string "|"; affich_rg_pr_oo l
|(x,Comp (Atom a,Atom b))::l-> print_string b; print_string "->";
 print_string x;print_string a;print_string "|";
affich_rg_pr_oo l
|_->raise Error;;

let rec print_rest typ=
match typ with
|Comp(Atom a,Comp(b,c))->print_string a;print_rest (Comp(b,c))
|Comp(Atom a,Atom b)->print_string a;
|_->raise Error;;

(*génération des règles de production aux types de degré 2*)
let rec affich_rg_pr_2 lista =
match lista with
[]-> print_string ""
|(x,Comp(Atom a ,Comp(b,c)))::l->print_string "o";
print_string "->";print_string x;print_rest (Comp(Atom
a,Comp(b,c)));print_string "|";affich_rg_pr_2 l
|(x,Atom a)::l->print_string a; print_string "->";
print_string x;print_string "|";affich_rg_pr_2 l
|(x,Comp(Atom a,Atom b))::l->print_string b;
 print_string "->"; print_string x;print_string a;
print_string "|"; affich_rg_pr_2 l
|_-> raise Error;;

(*est ce que le type est pour les termes projecteurs?*)
let rec test_proj lista=
match lista with
|[]->true
|(x,Atom a)::l->test_proj l
|_->false;;

(*génération des termes projecteurs*)
let rec print_term_proj listv=
```

```
match listv with
|[]-> print_string ""
|(x,Atom a)::l ->print_string "\\"; print_string(var_typ listv);
print_string ".";print_string "xi"; print_string "|";print_string "\n xi";
print_string "E";print_string "{";print_string (var_typ listv);print_string "}"

(*calcul rang d'un type*)
let rec rank  p  =
         match p with
      Atom a  ->  0
    | Comp (a,b) ->  (max ( (rank a) +1)  (rank b))
;;

let rec print_term p =

   match p with

 Atom t   -> print_string t


| Comp (a, b)   ->    (print_string "(" ;
      print_term  a ;
      print_string  "->";
      print_term b;
     print_string ")"
      )
;;

(*génération des termes initiaux degré 2*)
let affich_term2 lista1=
let rec affich_term_ini2=function
|[]->print_string ""
|(x,Atom a)::l->print_string "\\"; print_string(var_typ lista1);
print_string ".";print_string x;print_string "|";affich_term_ini2  l
|_::l->affich_term_ini2  l in affich_term_ini2 lista1;;

(*génération des termes initiaux des types finement engendrés degré <= 3*)
let affich_term lista1=
let rec affich_term_ini  =function
|[]-> print_string ""
|(x,Atom a)::l1->print_string "\\"; print_string(var_typ lista1);
```

```
print_string ".";print_string x;print_string "|";affich_term_ini  l1
|(x2,Comp (Atom a1,Atom a2))::l2->affich_term_ini  l2
|(x1,Comp(Comp(a,b),Atom c))::l-> resetvary();
recherch lista1 lista1((x1,Comp(Comp(a,b),Atom c))::l)  and
recherch lista lista1 ((x,t)::l) = match lista with
|[]->print_string "\\"; print_string  (var_typ lista1);print_string".";
print_string x;print_string"(";print_string "\\";affichy t;print_string".";
print_string "yi";print_string ")";print_string "|";affich_term_ini  l
|(x1,Atom a1)::l1 -> recherch l1  lista1((x,t)::l)
|(x2,Comp (Atom a1,Atom a2))::l2->resetvary(); print_string "\\";
print_string (var_typ lista1);print_string".";print_string x;
print_string "(";print_string "\\";affichy t;resetvary();
print_string ".";print_string x2; print_string "yi";print_string ")";
print_string "|";recherch l2 lista1 ((x,t)::l)
|(x4,Comp (Comp(a1,b1),Atom c1))::l3->recherch l3 lista1 ((x,t)::l) in
  affich_term_ini lista1;;

(*génération de générateurs pour les types finement engendrés degré <= 3*)
let affich_g lista1  =
let rec affich_termg  = function
|[]->print_string ""
|(x,Atom a)::l->affich_termg l
|(x,Comp(Atom a,Atom b))::l->resetvary();print_string "\\";
let f=(genvarf()) in print_string f ;print_string (var_typ lista1);
print_string ".";print_string x;print_string "(";print_string f;
print_string (var_typ lista1);print_string ")";print_string
"|"; affich_termg l  |(x,Comp (Comp(a1,b1),Atom c1))
::l3->resetvary();print_string "\\";let f1=(genvarf())
in print_string f1;print_string(var_typ lista1);print_string ".";
print_string x;print_string "(";print_string "\\";
 affichy (Comp (Comp(a1,b1),Atom c1));print_string ".";
resetvary();print_string f1;print_string (var_typ lista1);print_string ")";
print_string "|";affich_termg l3  in affich_termg lista1 ;;

(*calcul des générateurs  pour les types de degré 2*)
let rec genf2 typ=
match typ with
|Comp(Atom a,Comp(b,c))->print_string(genvarf());
genf2 (Comp(b,c))
|Comp(Atom a,Atom b)->print_string (genvarf())
|_->raise Error;;
```

```
let rec  genf_vartyp typ lis=
match typ with
|Comp(Atom a,Comp(b,c))-> print_string "(";print_string (genvarf());
print_string (var_typ lis);print_string ")";genf_vartyp (Comp (b,c)) lis
|Comp(Atom a,Atom b)->print_string "(";print_string (genvarf());
print_string (var_typ lis); print_string ")";;

let affich_gen2 lista1=
let rec affich_termg2=function
|[]->print_string""
|(x,Atom a)::l->affich_termg2 l
|(x,Comp(Atom a,Atom b))::l->print_string "\\";let f=(genvarf())
in print_string f ;print_string (var_typ lista1);print_string ".";
print_string x;print_string "(";print_string f;print_string(var_typ lista1);
print_string ")";affich_termg2 l
|(x,Comp (Atom a,Comp(b,c)))::l->print_string "\\";
genf2 (Comp(Atom a,Comp(b,c)));resetvarf();print_string (var_typ lista1);
print_string ".";print_string x; genf_vartyp (Comp (Atom a,Comp(b,c)))lista1;
affich_termg2 l in affich_termg2 lista1;;

(*est 'il le type finement engendré?*)
let rec testjol lis =
match lis with
|[]->true
|(x,Atom a)::l->testjol l
|(x,Comp(Atom a,Atom b))::l->testjol l
|(x,Comp(Comp (a,b),Atom c))::l-> testjol l
|_::l-> false;;

(*est ce qu'il y'a argument de type Atom?*)
let rec test_atom lis=
match lis with
|[]-> false
|(x,Atom a)::l->true
|_::l-> test_atom l;;

try
  flush stdout;
  for i=0 to  5 do print_newline()done;
  print_string "** Enumération des habitants d'un type finement engendré **\n";
```

```
  print_string "**************************************************\n";
  print_newline(); resetvar(); resetvary();resetvarf(); resetvarb();
  print_string " Type := ";

  while true do
    let t =  read_line()  in
      try
                  let typ = (lire_prop  (lexer t)) in
          begin resetvar(); resetvary();resetvarf(); resetvarb();
                      (print_newline(); print_string "The rank of  " ;
print_term ( typ); print_string "  est : ";  print_int(rank(typ));
                              print_newline();

                    let list_typ_var =tta typ lista in
                    let list_brute= elimin_der list_typ_var list_brut in
if rank(typ) =2 then begin if test_atom (list_brute)=true then begin
 print_string "\n La grammaire de ce type \n"; print_string t;
 print_string "  est :"; print_newline();affich_rg_pr_2(list_brute);
print_newline();print_string "\n Les termes initiaux du type \n";
 print_string t; print_string "  sont :"; print_newline();
  affich_term2 (list_brute);print_newline();
print_string "\n Les termes générateurs du type \n"; print_string t;
 print_string "  sont :"; print_newline(); affich_gen2 (list_brute);
print_newline();
end else print_string "Ce type de degré 2 ne possède pas d'habitant";
 end else

          if (testjol list_brute)= true then
           begin
           if test_proj list_brute= false then
           begin
          print_newline(); print_newline ();
print_string "\n La grammaire de ce type \n"; print_string t;
print_string "  est :"; print_newline();  print_newline ();
affich_rg_pr_oo (list_brute); print_newline();
print_string "\n Les termes initiaux du type \n";
print_string t; print_string "  sont :"; print_newline();
print_newline ();affich_term (list_brute); print_newline();
print_string "\n Les termes générateurs du type \n"; print_string t;
print_string "  sont :"; print_newline();  print_newline ();
affich_g (list_brute);
```

```
end else
( print_string "Les Termes Projecteurs :"; print_newline();
print_newline();print_term_proj(list_brute));
end  else print_string "Ce type n'est pas finement engendré ";

       print_newline();
               print_string "************************************\n";
                   print_newline();
      resetvar();

print_string " Type  : "

                            )

          end
      with
          Error ->print_string ("\n" ^ t); raise END;

  done

with END -> ();
exit 0
;;
```

Bibliographie

[1] Benjamin C.Pierce, *Types and programming Languages*. The MIT Press, Massachusetts Institute of Technology Cambridge, Massachusetts 02142, ISBN 0-262-16228-8, 2002.

[2] W. A. Howard, *The formula-as-type notion of construction*. in J. P. Seldinand J. R. Hindely, Editors, To H. B Curry, essys on Combinatory Logic, Lambda Calculus and Formalism, pp 479-490. Academic Press, 1980.

[3] J. Roger Hindely, *Basic Simple type theory*, Cambridge University press 1997.

[4] H.Barendregt, W.Dekkers,R.Statman Typed Lambda Culculus Volume I :$\lambda^{A\rightarrow}$ Aout, 2006

[5] H. P Barendregt, *the lambda calculus, its syntax and semantic*, north-Holland Co., Amesterdam, s^{nd} edition 1984.

[6] C.Ben-YELLES, *Type assignement in the lambda-calculus*, Doctoral Thesis, Septembre 1979.

[7] S. Broda, L. Damas, *The Formula-Tree Proof Method*, Depatamento de Ciência de Computadores-Faculdade de Ciências and Laboratoria de Inteligência Artificial e Ciência de Computadores(Universidade do Porto Portugal), Technical Report Series DCC-04-01.

[8] S. Broda and L.Damas, *Counting a type's principal inhabitants*, Lecture Notes in Computer Science 1581 (1999), pp 69-82.

[9] S. Broda and L.Damas, *Generating normal inhabitants of types with a common structure*, Depatamento de Ciência de Computadores-Faculdade de Ciências and Laboratoria de Inteligência Artificial e Ciência de Computadores (Universidade do Porto Portugal), Technical Report Series, DCC-01-01.

[10] S. Broda and L.Damas, *On Long Normal inhabitants of a type*, J. of Logic and Computation 15, pp 353-390, 2005.

[11] S. Broda and L.Damas, M. Finger, P. Silva, *The decidability of a fragment of BB'IW-logic*, Theoretical Computer Science Vol 318 pp 373-408, 2004.

[12] M.Zaionc, *Lambda definability is decidable for second order types and for regular third order type*,Technical Report 95-24 of Department of Computer Science at State University of New York at Buffalo (http ://tcs.uj.edu.pl/ zaionc/papers.html), May 1995.

[13] M.Zaionc, *Fixpoint technique for counting terms in typed λ-calculus*, Technical Report 95-20 of Department of Computer Science at State University of New York at Buffalo (citeseer.nj.nec.com/158046.html), Avril 1995.

[14] M.Zaionc, *probabilistic approach to the lambda definability for fourth order types*, Electronic Notes in Theoretical Computer Science, Vol 140, pp 41-54, 2005.

[15] M.Zaionc, R. David, *Counting proofs in propositional logic*, Archive for Mathematical Logic, Vol 48 N°2, pp 185-199, 2009

[16] M. Moczurad, M. Zaionc, *second order λ-Representability by Linear Terms. The case study of type $(N \to N) \to N$*, Computer science Depatement Jagiellonian University Nawojki Poland, 1997.

[17] M.Zaionc, *Lambda definability is decidable for second order types and for regular third order types*, Report 95-24 of Department of Computer Science at State University of New York at Buffalo, May 1995,(citeseer.nj.nec.com/88135.html).

[18] M.Zaionc, *Statistical propreties of types*, Mathematical Structures in Computer Science, 10, pp 575-594 June 18, 1998

[19] Gilles Dowek and Ying Jiang, *Enumerating the inhabitants of a simple type*, The Computer Journal, 2008.

[20] Gilles Dowek and Ying Jiang, *Enumerating proofs of positive formulae*, The computer Journal Vol. 00 No. 0, 2008.

[21] Jue Wang, *Générating Random Terms in Beta Normal Form of the Simply-Typed Lambda Calculus*, Boston University july 10, 2005.

[22] Jue Wang, *The Efficient Generation of Random Programs and Their Applications*, Submitted in Partial Fulfillment of the Prerequisite for Honors in Computer Science, 2004.

[23] Hemdani Chaabane, *Caractérisation des termes ayant le même type principal dans le $\lambda-$calcul simplement typé* Mémoire de Magister 2002.

[24] M.asako Takahashi, Yohji Akama, *Normal Proofs and their Grammar* Lecture Notes in Computer Science vol 789 pp 465-493, 1996.

[25] Thierry Joly, *Types finitely generated* TLCA 2001, LNCS 2044, pp.240-252, 2001 Springer-Verlag Berlin Heidelberg.

[26] Thierry Joly, *Non finitely generated types & λ-terms combinatoric representation cost*, C. R. Acad. Sci. Paris, t. 331, Série I, pp 1-6, 2000.

[27] Thierry joly, *On λ-Definability II :the Fixed Type Problem and Finite Generation of Types*, pp 1-19, 2002.

[28] Thierry joly, *Constant time parallel computations in λ-culculus*, ISSN 0304-3975 CODEN TCSCDI Vol-226 N°1-2, pp 975-985, 2001.

[29] Thierry joly, *On λ-Definability I :the Fixed Type Problem and Generalizations of matching problem.* Fundam. inform, 65(1-2), pp 135-151, 2005.

[30] Ralph Loader, *The undecidability of λ-Definability* Logic, Meaning and Computation : Essays in Memory of Alonzo Church, Kluwer, pp 331-342, 2001

[31] Sachio Hirokawa, *Degree-Twoo Formulas and Thier Proofs*, Research institute of Fundamental Information Science(RIFIS) Technical Report November 7, 1991.

[32] J. B. Wells and Boris Yakobowski, *Graph-Based Counting and enumeration with Applications for Program Fragment Synthesis*, Lecture notes in Computer Science, Vol 3573 ISSN 0302-9743, pp 262-277, 2005.