REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE M'HAMED BOUGARA – BOUMERDES
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUEE

Résumé de Mémoire de Magistère

THEME

Confluence et Préservation de la Propriété de Normalisation forte du système λ@

Présenté par

M^{r} SALMI Cheikh

Soutenu publiquement le ……

Membres du jury:

Mme. A. Aissani	(Professeur U.S.T.H.B.)	Président
M. Mezghiche	(Professeur U.M.B.B.)	Rapporteur
A. Belkhir	(Maître de conférence U.S.T.H.B.)	Examinateur
M. Khalifa	(Docteur U.M.B.B.)	Examinateur

ANNEE -2006-

Table des matières

Introduction Générale

Dans le λ-calcul classique, l'opération de la β-réduction utilise une définition de la substitution externe à la théorie de ce calcul. La présence des notions de variables libres et de variables liées rend l'implémentation de l'opération de substitution très complexe et par conséquent rend difficile la mise en œuvre des évaluations basées sur la λβ-réduction.

Une solution à ce problème consiste à formaliser et internaliser le processus de substitution en introduisant des extensions du λ-calcul. De tels systèmes sont appelés calculs avec substitutions explicites.

Plusieurs systèmes de calculs avec substitutions explicites ont été définis. (Voir : [KR 95], [ACCL91], [BBLR 96], [DG 99],…).

Le $\lambda\sigma$ calcul introduit dans [ACCL91] résout efficacement les substitutions. Mais il ne préserve pas la normalisation forte du λ-calcul et n'est pas confluent. Plusieurs alternatives de ce calcul ont été proposées mais aucune d'entre elles ne satisfait les deux propriétés à la fois.

Dans notre mémoire, nous traitons le $\lambda@$, un nouveau calcul de substitution explicite introduit dans [GMMS02]. Ce calcul a une présentation très simple et un nombre réduit de règles. Contrairement aux autres calculs, le $\lambda@$ n'utilise aucun opérateur de substitution et résout automatiquement les conflits de noms de variables.

Dans le cadre de la théorie de la réécriture nous étudions ce calcul et nous donnons les preuves de ces propriétés principales.

Le mémoire est organisé comme suit :
Le chapitre I aborde les systèmes de réécriture. Dans la partie 1 nous donnons un aperçu des systèmes de réductions abstraits, Nous donnons les définitions des notions de base des systèmes

de réduction abstraits tels que la réduction, la confluence, la normalisation forte et l'interaction entre ces propriétés.

Nous avons consacré la partie 2 aux systèmes de réécriture proprement dits. Après une présentation des notions préliminaires sur les relations binaires et les ordres qui sont très utiles dans les démonstrations, nous donnons les définitions de toutes les notions relatives aux systèmes de réécriture tels que les signatures, les règles de réécriture, les paires critiques. Nous étudions ensuite les systèmes de réécriture orthogonaux, nous en donnons les théorèmes fondamentaux : le théorème de confluence et le théorème d' O'donnell.

Dans la partie 3, nous abordons la terminaison des systèmes de réécriture. Nous étudions toutes les techniques qui existent pour prouver cette propriété dans le cadre de la réécriture.

On a consacré le chapitre II pour l'étude des λ-calculs et les substitutions explicite. Après une présentations des notions de base tels que les variables libres, liées, la β-réduction, la α-conversion et le problème de la substitution implicite nous étudions la substitution explicite avec ses deux formes : indices de De Bruijin et les variables nommées. Nous concluons le chapitre par un panorama des calculs avec substitution explicite.

Dans le chapitre III nous présentons une étude détaillée du système λ@ et nous montrons que ce système de réécriture est confluent et préserve la forte normalisation du λ-calcul classique.

Chapitre I

Les Systèmes de Réécritures

I.1 Systèmes de Réduction Abstraits

Nous commençons par introduire les systèmes de réduction abstraits ou systèmes de réécriture abstraits, composés uniquement d'un ensemble d'objets liés par une ou plusieurs relations de réductions ou de réécritures (fig. I.1.1), et qui seront perçues comme des relations de transformation ou de calcul.

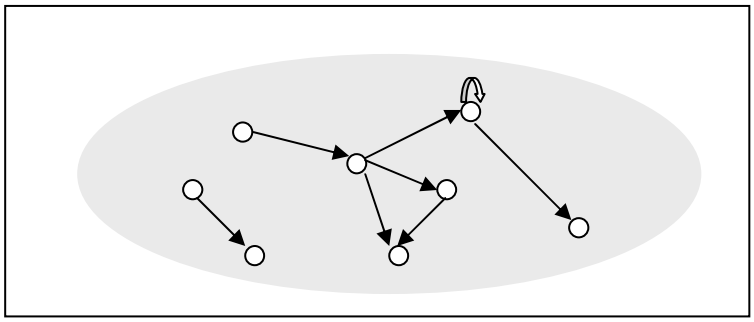


Fig. I.1.1

I.1.1 Notions de base

1) Un système de réduction abstrait (ARS) est une structure $A = \langle A, (\rightarrow_\alpha)_{\alpha \in I} \rangle$ composée d'un ensemble A et d'une suite finie dénombrable de relations binaires $(\rightarrow_\alpha)$ sur A, appelées aussi relations de réduction ou de réécriture.

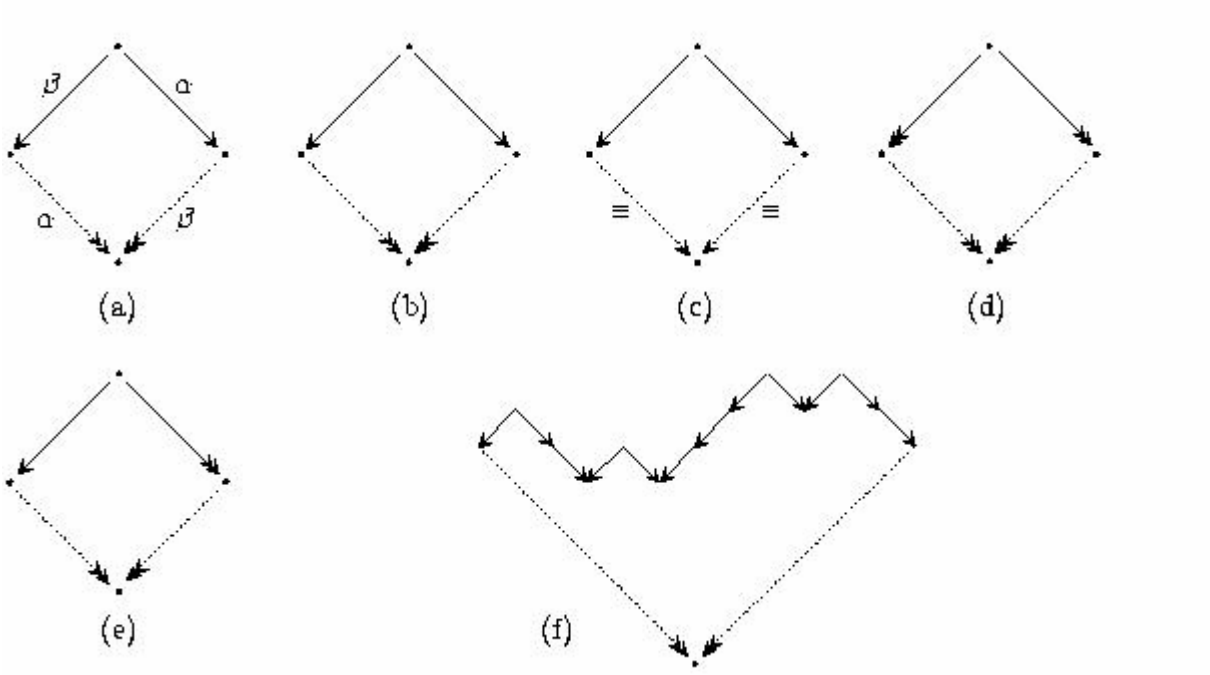
Quelque fois nous notons $(\rightarrow_\alpha)$ comme α, dans le cas d'une seule relation de réduction nous utilisons aussi $(\rightarrow)$ sans plus. (Un ARS avec une seule relation de réduction est appelé 'système de remplacement' dans [STA 75] et un 'système de réduction' dans [JAN 88]. Si pour $a, b \in A$ nous avons $(a,b) \in (\rightarrow_\alpha)$ nous écrivons $a \rightarrow_\alpha b$ et nous disons que b est le (α-) réduit de a en une seule étape.

2) La fermeture transitive réflexive de $(\rightarrow_\alpha)$ s'écrit $\twoheadrightarrow_\alpha$ (La notation $\rightarrow_\alpha^*$ est plus habituelle, mais la notation à flèche double est préférée dans le cas des schémas graphiques.)
Donc on écrit $a \twoheadrightarrow_\alpha b$ s'il est possible de trouver une séquence éventuellement vide et finie 'd'étapes de réduction' $a \equiv a_0 \rightarrow_\alpha a_1 \rightarrow_\alpha ... \rightarrow_\alpha a_n \equiv b$. Ici $\equiv$ Indique l'identité dans A. l'élément b est appelé un (α-) réduit de a. La relation d'équivalence générée par $(\rightarrow_\alpha)$ est notée $=_\alpha$ appelée aussi la relation de convertibilité générée par $(\rightarrow_\alpha)$. La fermeture réflexive de $(\rightarrow_\alpha)$ est notée $(\rightarrow^{=}{}_\alpha)$. La fermeture transitive de $(\rightarrow_\alpha)$ est notée $(\rightarrow^{+}{}_\alpha)$. La relation inverse de $\rightarrow_\alpha$ est notée $({}_\alpha\leftarrow)$ ou $(\rightarrow^{-1}{}_\alpha)$. L'union $(\rightarrow_\alpha \cup \rightarrow_\beta)$ est dénotée par $(\rightarrow_{\alpha\beta})$. La composition $(\rightarrow_\alpha \circ \rightarrow_\beta)$ est définie par $(a \rightarrow_\alpha \circ \rightarrow_\beta)$ si $a \rightarrow_\alpha c \rightarrow_\beta b$ pour un certain $c \in A$.

3) Si α, β sont deux relations de réduction sur A, nous disons que α commute faiblement avec β si le diagramme suivant (Fig. I.1.2a) est valable, c'est-à-dire si $\forall a,b,c \in A\ \exists d \in A$ $(c\ {}_\beta\leftarrow a \rightarrow_\alpha b \Rightarrow c \twoheadrightarrow_\alpha d\ {}_\beta\prec\leftarrow\ b)$ ou d'une notation plus courte : ${}_\beta\leftarrow$ o $\rightarrow_\alpha \subseteq \twoheadrightarrow_\alpha$ o ${}_\beta\prec\leftarrow$.
Plus encore, α commute avec β si $\twoheadrightarrow_\alpha$ et $\twoheadrightarrow_\beta$ commute faiblement. (Cette terminologie diffère de celle de Bachmair & Dershowitz (voir [BD 86], où, α commute avec β si α^{-1} o $\beta \subseteq \beta^{-1}$ o α.)

4) La relation de réduction $\rightarrow$ est dite faiblement confluente ou WCR (weakly Church Rosser), si elle est faiblement auto commutative (voir Fig. I.1.2b), c'est-à-dire si $\forall a,b,c \in A\ \exists d \in A$ $(c \leftarrow a \rightarrow b \Rightarrow c \prec\leftarrow d \twoheadrightarrow b)$.
(La propriété WCR est aussi appelée 'confluence locale' [JAN 88].)

5) $\rightarrow$ est subcommutative (notation $WCR^{\leq 1}$) si le diagramme dans le schéma I.1.2c est valable, c'est-à-dire $\forall a,b,c \in A\ \exists d \in A$ $(c \leftarrow a \rightarrow b \Rightarrow c \rightarrow^{=} d \leftarrow^{=} b)$.
6) $\rightarrow$ est dite confluente, Church-Rosser ou possède la propriété de Church-Rosser (CR) si elle est autocommutatif (voir d) c'est-à-dire $\forall a,b,c \in A\ \exists d \in A$ $(c \prec\leftarrow a \twoheadrightarrow b \Rightarrow c \twoheadrightarrow d \prec\leftarrow b)$



FigI.1.2

Dans ce qui suit nous utilisons les termes ‘confluence’ et ‘Church-Rosser’ ou ‘CR’ sans préférence. Pareil pour la confluence faible et WCR. La proposition suivante découle immédiatement des définitions.

Proposition I.1.1 Les cas suivants sont équivalents :

(i) $\rightarrow$ est confluente.

(ii) $\rightarrow\!\!\succ$ est faiblement confluente.

(iii) $\rightarrow\!\!\succ$ est auto commutative.

(iv) $\rightarrow\!\!\succ$ est sub-commutative.

(v) le diagramme dans le schéma I.1.2e est valable, c'est-à-dire

$\forall a,b,c \in A \ \exists d \in A \ \ (c \leftarrow a \rightarrow\!\!\succ b \Rightarrow c \rightarrow\!\!\succ d \prec\!\!\leftarrow b)$.

(vi) $\forall a,b \in A \ (a = b \Rightarrow \exists c \in A \ a \rightarrow\!\!\succ c \prec\!\!\leftarrow b \ )$.

(Dans ce cas : ‘=’ est la relation de convertibilité générée par $\rightarrow$ voir le diagramme dans Fig. I.1.2f.)

Définition I.1.1 Soit $A = \langle A, \rightarrow \rangle$ un système de réduction abstrait.

(i) Nous disons que $a \in A$ est une forme normale s'il n'existe aucun b dans A de tel que $a \rightarrow b$. De plus, $b \in A$ a une forme normale si $b \rightarrow\!\!\succ a$ et a est une forme normale.

(ii) La relation de réduction $\rightarrow$ est faiblement normalisante (Weakly Normalizing (WN)) si $\forall a \in A$, a possède une forme normale. Dans ce cas nous disons aussi que A est WN.

(iii) A (ou $\rightarrow$) est fortement normalisant (Strongly Normalizing (SN)) si chaque séquence de réductions $a_0 \rightarrow a_1 \rightarrow ... \rightarrow a_n$; est finie. (Une autre terminologie : $\rightarrow$ termine ou elle Noethérienne.) Si la relation de réduction inverse $\leftarrow$ est SN nous disons que A (ou $\rightarrow$) est SN^{-1}.

(iv) A (ou $\rightarrow$) a la propriété de la forme normale unique (UN) si

($a <\!\!\leftarrow . \rightarrow\!\!> b \Rightarrow a \equiv b$ pour toutes les formes normales $a, b \in A$.

(v) A (ou $\rightarrow$) a la propriété de la forme normale (NF) si

$\forall a, b \in A$ (a Est une forme normale et $a = b \Rightarrow b \rightarrow\!\!\succ b$).

(vi) A (ou $\rightarrow$) est inductif (Ind) si pour toute séquence de réduction (probablement infinie) $a_0 \rightarrow a_1 \rightarrow ...$ il y a un $a \in A$ tel que $a_n \rightarrow a$ pour tout n.

(vii) A (ou $\rightarrow$) est croissante (Inc) s'il existe une application $|\ |: A \rightarrow N$ tel que

$\forall a, b \in A$ ($a \rightarrow b \Rightarrow |a| < |b|$). Ici N est l'ensemble des nombres naturels avec l'ordre habituel sur les entiers<.

(viii) A (Ou $\rightarrow$) est à branchement fini (FB) si pour tout $a \in A$ à l'ensemble des réductions à une étape de a, $\{b \in A / a \rightarrow b\}$, est fini. Si la relation de réduction inverse $\leftarrow$ est FB, nous disons que A (Ou $\rightarrow$) est FB^{-1}. Dans [HUE 80], FB est appelé 'localement fini'.

Un ARS qui est confluent et terminant (CR & SN) est aussi appelé complet (Une autre terminologie : 'canonique' ou 'uniquement terminant').

Avant d'étudier les relations entre ces propriétés, nous introduisons d'abord quelques concepts de plus.

Définition I.1.2 Soit $A = \langle A, \rightarrow_\alpha \rangle$ et $B = \langle B, \rightarrow_\beta \rangle$ deux ARS. A est un sous ARS de B, noté $A \subseteq B$, si :

(i) $A \subseteq B$.

(ii) α est la restriction de β sur A, c'est-à-dire $\forall a, a' \in A (a \rightarrow_\beta a' \Leftrightarrow a \rightarrow_\alpha a')$.

(iii) A est fermé sous β, c'est-à-dire $\forall a \in A (a \rightarrow_\beta b \Rightarrow b \in A)$.

L'ARS B est aussi appelé une extension de A.

Notez que toutes les propriétés vues jusque là (CR, WCR, $WCR^{\leq 1}$, WN, SN, UN, NF, Ind, Inc, FB) sont stables sous l'inclusion : par exemple si $A \subseteq B$ et B est CR donc A l'est aussi. Le sous ARS déterminé par un élément a d'un ARS A est particulièrement intéressant.

Définition I.1.3 Soit $A = \langle A, \rightarrow_\alpha \rangle$ un ARS et $a \in A$. Le graphe de réduction de a noté $G(a)$ est le plus petit sous ARS de A contenant a. Donc G(a) a pour éléments toutes les réductions de a (y compris a lui-même) et est structuré par la restriction de la relation $\rightarrow$ sur cette ensemble de réduits.

Nous allons maintenant rassembler dans un théorème plusieurs implications entre les différentes propriétés des ARS. La première partie (i) est en réalité la motivation principale pour le concept de confluence. Elle garantit des formes normales uniques, qui est bien sur une propriété souhaitable dans l'implémentation des spécifications des types de données algébriques. A part l'implication fondamentale CR $\Rightarrow$ UN, le concept plus important est (ii), connu aussi comme le Lemme de Newman.

Théorème I.1.2

i. CR $\Rightarrow$ NF $\Rightarrow$ UN

ii. SN & WCR $\Rightarrow$ CR (lemme de Newmman)

iii. UN & WN $\Rightarrow$ CR

iv. UN & WN $\Rightarrow$ Ind

v. Ind & Inc $\Rightarrow$ SN

vi. WCR & WN & Inc $\Rightarrow$ SN

vii. CR $\Leftrightarrow$ CP pour les ARSs dénombrables.

Les propositions dans l'énoncé du théorème (et quelques autres) sont montrées aussi dans le schéma I.1.3 ; ici il est important de voir si la flèche d'implication pointe vers le signe de la conjonction & ou vers une des conjonctions, Idem pour la queue de la flèche d'implication. Il est impossible de renverser une des flèches dans le diagramme des implications. Un contre exemple

instructif pour WCR ⇒CR est le TRS[1] dans le schéma I.1.4, (donné par R. Hindley, voir aussi [Hue80]).

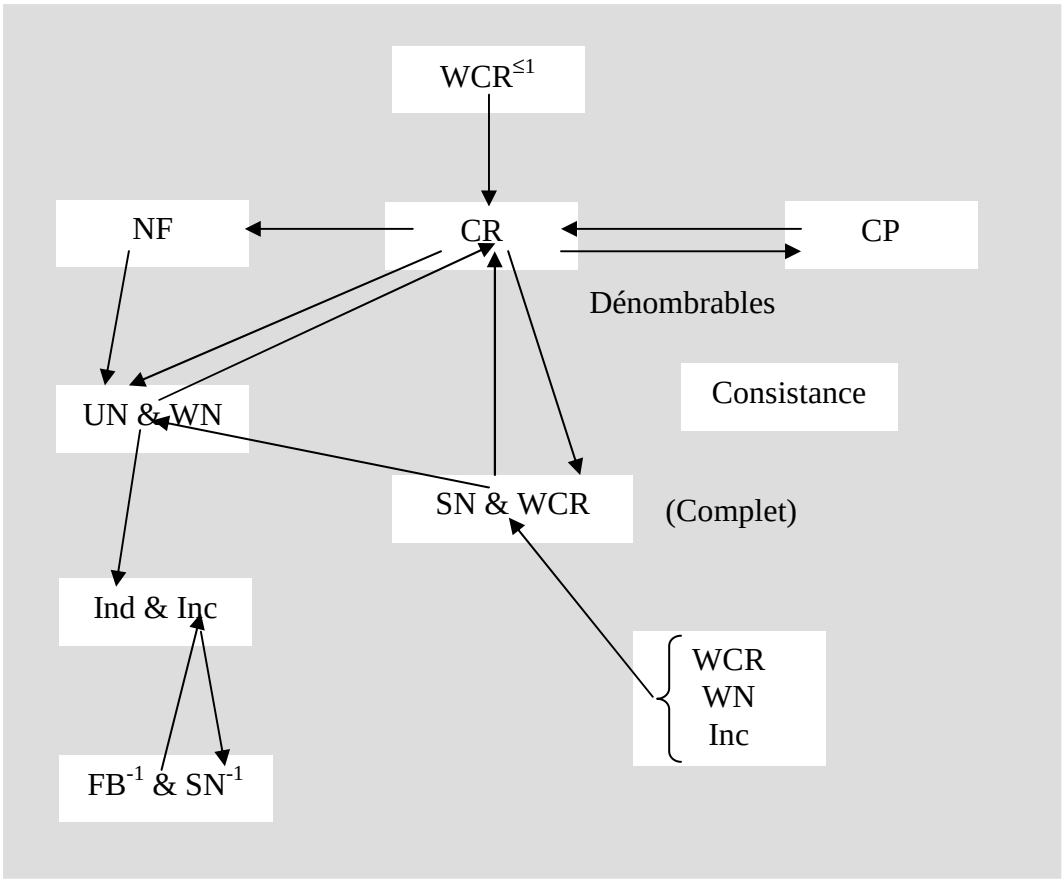


Fig. I.1.3

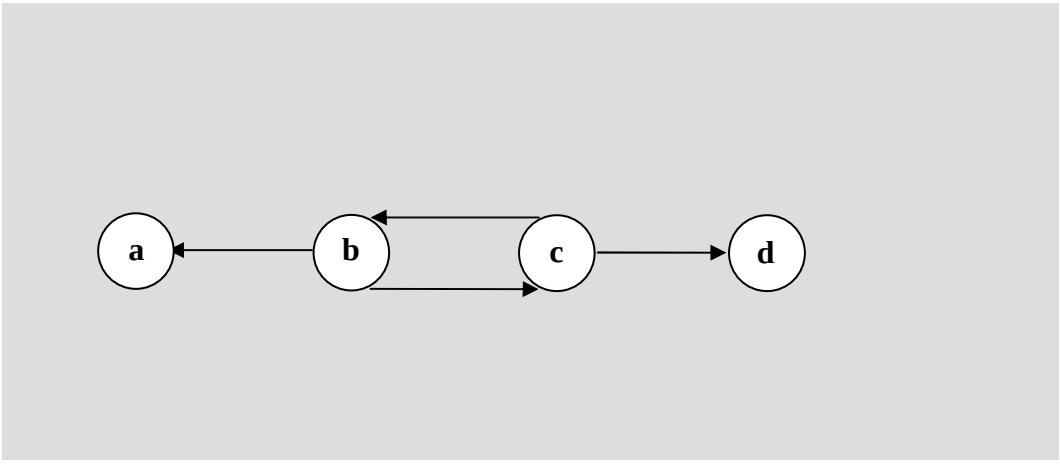


Fig. I.1.4

[1] Les systèmes de réécriture de termes (TRS) seront l'objet de la 2ième section

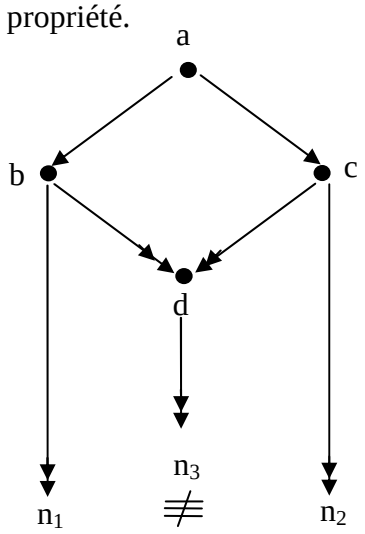
Preuve des propriétés :

i. Immédiate à partir des définitions.

ii. Des preuves courtes du Lemme de Newman sont données dans [HUE 80] et dans [BAR 84] qui est très simple :

Par la propriété SN chaque élément $a \in A$ se réduit à au moins une forme normale. Il suffit pour CR de démontrer que cette forme normale est unique.

Un élément dans l'ARS considéré est appelé ambigu s'il peut se réduire à deux (ou plus) formes normales. A présent nous affirmons qu'un élément ambigu possède un réduit en une étape qui est de nouveau ambigu. Pour cela, admettons que a est ambigu, se réduisant à des formes normales différentes $n_1, n_2 : a \to b...n_1$ et $a \to c...n_2$. Si $b \equiv c$ l'affirmation est prouvée : b est ambigu. Autrement, on applique WCR aux étapes divergentes $a \to b$, $a \to c$ pour obtenir une réduction d commune de telle sorte que $b \to\!\!\succ d$ et $c \to\!\!\succ d$. Si d est ambigu, b et c sont ambigus aussi. Si d n'est pas ambigu, il se réduit à n_1, n_2 ou une autre forme normale n_3. Dans tous les cas b ou c est ambigu (Fig. I.1.5). Cela prouve l'affirmation. Mais l'affirmation contredit la supposition de SN, ce qui démontre la propriété.



Les formes Normales

Fig. I.1.5

La preuve des propriétés : (iii), (iv), (v), (vi), (vii) sont dans [TERESE 03]

I.2 Relations et Réécriture

Dans cette section nous allons définir les notions fondamentales des *systèmes de réécriture*. Nous préciserons dans un premier temps des considérations sur les *relations binaires* générales avant de nous concentrer sur les relations d'*ordre*, puis et après avoir défini les *termes* sur une signature d'une relation de *réécriture, n*ous décrivons les propriétés les plus recherchées sur de telles relations avec un intérêt tout particulier pour la *terminaison*.

Une relation binaire sur un ensemble $\sum$ est un ensemble de couples d'éléments de $\sum$. On dit pour deux éléments formant un couple qu'ils sont *en relation*.

I.2.1 Notions Préliminaires

Définition I.2.1 Une relation binaire $\Re$ sur un ensemble $\sum$ est une partie de $\Sigma x \Sigma$. Les éléments (u,v) de cette relation sont en général notés $u \Re v$.

On utilise la notation $\circ$ pour la composition des relations : pour deux relations $\Re$ et $\Im$ sur un ensemble $\sum$, $\Re \circ \Im$ désigne l'ensemble $\{(u,v) \mid$ il existe $u' \in \Sigma$ tel que uRu' et $u' \Im v \}$

Enfin, pour deux relations $\Re$ et $\Im$ sur un ensemble $\sum$, on note $\Re - \Im$ la différence ensembliste $\Re / (\Re \cap \Im)$.

Définition I.2.2 Une Relation Binaire $\Re$ sur $\sum$ est :

- Symétrique si pour tous u et v de $\sum$, $u \Re v$ entraîne $v \Re u$;
- Antisymétrique si pour u et v de $\sum$, $u \Re v$ et $v \Re u$ entraîne $v = u$;
- Réflexive si pour tout élément u de $\sum u \Re u$;
- Antiréflexive si pour tout élément u de $\sum$, on n'a pas $u \Re u$;
- Transitive si pour tout u, v et t de $\sum$, $u \Re v$ et $v \Re t$ entraîne $u \Re t$.

Nous nous intéresserons dans la suite à la terminaison éventuelle des séquences d'éléments en relation et en particulier aux relations *noethériennes.*

Définition I.2.3 Une relation $\Re$ sur un ensemble $\sum$ est Noethérienne s'il n'existe pas de chaîne infinie d'éléments de $\sum$ en relation.

Définitions I.2.4 Soit R une relation sur un ensemble E, R définit :

- Une relation d'équivalence si elle est symétrique, réflexive et transitive ;
- Un pré ordre si elle réflexive et transitive ;
- Un ordre si elle est réflexive, transitive et antisymétrique.
- Un ordre strict si elle est antiréflexive, transitive et antisymétrique.

Dans une relation donnée par un pré ordre $\geq$ la partie stricte est l'ensemble $\{(u,v) | (v,u) \notin \geq\}$; le pré ordre $\geq$ peut être désigné comme la partie large de la relation.

Définition I.2.5 Un pré ordre est dit *bien fondé* si sa partie stricte définit une relation noethérienne.

Proposition I.2.1 Toute relation incluse dans un ordre bien fondé est bien fondée.

I.2.2 Relations sur les termes

I.2.2.1 Signatures et Termes

Une signature F est un ensemble de symboles muni d'une application :

$AR_F : F \rightarrow N$. (N est l'ensemble des entiers naturels). On appelle *arité* du symbole $f \in F$ l'entier naturel $AR_F(f)$.

On distingue parmi les symboles de F ceux d'arité nulle qu'on appelle *constantes*.

Définition I.2.6 Soient F une signature et X un ensemble dénombrable de symboles disjoint de F . L'algèbre de *termes* sur F et X , notée $T(F,X)$ est le plus petit ensemble tel que :

1. $X \subset T(F,X)$ et on que X est l'ensemble des *variables*.
2. Si $t_1, \ldots t_n \in T(F,X)^n$ et si $f \in F$ avec $AR(f) = n$ alors $f(t_1, t_2, \ldots t_n) \in T(F,X)$.

Un élément de $T(F,X)$ est un *terme*. Si t est un terme, on note $Var(t)$ l'ensemble des variables qu'il contient. Un terme sans variables est dit *clos*.

Définition I.2.7 Une *position* dans un terme est un mot sur l'alphabet $N - \{0\}$. Le mot vide qui représente une position à la racine est noté Λ . La concaténation de p et q est notée $p.q$ ou pq . $Pos(t)$ désigne l'ensemble des positions du terme t .

Le *symbole de tête* est le nom de la fonction (ou de la variable) à la position racine. On désigne par $\Lambda(t)$ le symbole de tête du terme t .

On définit le *sous terme* de s à la position p (noté $s|_p$) par :

- $s|_\Lambda = s$
- $f(t_1,\ldots,t_n)|_{ip} = t_i|_p$.

Un sous terme à la position p est dit propre si $p \neq \Lambda$.

Le remplacement de s par t à la position p (noté $s[t]_p$) est définit par :

- $s[t]_\Lambda = t$,
- $f(t_1,\ldots,t_n)[t]_{ip} = f(t_1,\ldots t_i\,[t]_p,\ldots,t_n)$.

Le terme $t = f(f(a,x),g(h(x,b,y)\ )\ )$ est représenté par l'arbre de la figure I.2.1.
On peut trouver dans le schéma par exemple que $\Lambda(t) = f$ et que le sous terme de t à la position 21 est $h(x,b,y)$.

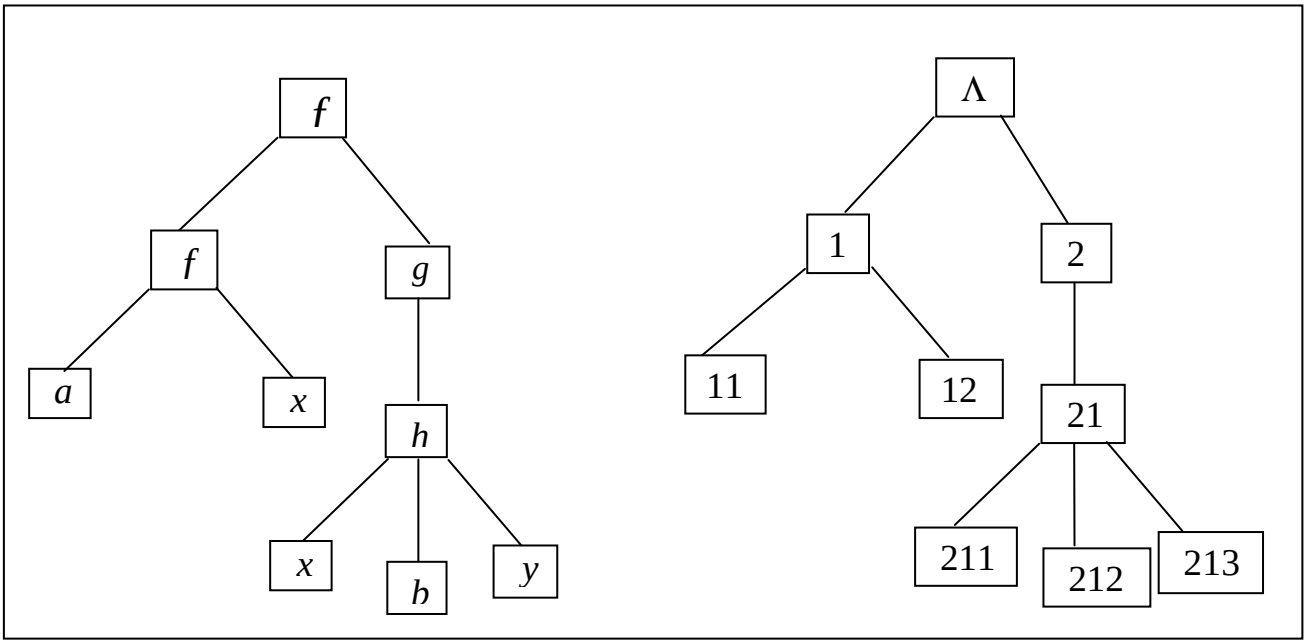


Fig.I.2.1 Exemple de Terme et positions sous forme d'arbre.

Définition I.2.8 Un contexte C est un terme qui contient une ou plusieurs occurrences d'une constante spéciale $\# \notin F$ appelé « hole ».
Dans le cas où $C[...]$ est un contexte contenant n occurrences de #(à des positions déterminées) et où $t_1, t_2,\ldots t_n$ représentes n termes de l'algèbre, $C[t_1,\ldots t_n]$ dénote le remplacement des # par les termes t_i.

Définition I.2.9 Soit $T(F,X)$ une algèbre de termes, une *substitution* est une application σ de l'ensemble des variables X vers l'ensemble des termes. L'ensemble $\{x \in X | \sigma(x) \neq x\}$ est appelé le support de la substitution. Une substitution peut être aussi décrite à l'aide de son support : par exemple $\{x \mapsto t_1, y \mapsto t_2\}$ représente une substitution σ de support $\{x, y\}$ telle que $\sigma x = t_1$ et $\sigma y = t_2$.

Remarques I.2.1

- l'application d'une substitution est écrite on notation post fixée (exemple : $x\sigma$ désigne $\sigma(x)$)
- une substitution σ peut être étendu comme endomorphisme de $T(F,X)$ en posant: $f(t_1...,t_n)\sigma = f(t_1\sigma,...,t_n\sigma)$.
- Le terme $t\sigma$ est appelé *instance* de t.
- Une substitution est dite *close* si l'image de son support est un ensemble qui ne contient que des termes clos.

Définition I.2.10 On dit qu'un terme u filtre un terme t s'il existe une substitution σ telle que $\sigma u = t$. Deux termes u et v sont dits *unifiables* s'il existe une substitution σ (alors appelé unificateur) telle que $\sigma u = \sigma v$.

I.2.2.2 Ordres sur les Termes

La notion d'ordres sur les termes (aussi connue comme ordering pair [KMY 99] est très utilisée dans les preuves de terminaison.

Définition I.2.11 Un ordre sur les termes est une paire de relations sur les termes $(\geq, \succ)$ qui vérifie :

1) $\geq$ est un pré ordre ;
2) $\succ$ est un ordre strict ;
3) $\succ \circ \geq \subseteq \succ$ et $\geq \circ \succ \subseteq \succ$.

Un ordre sur les termes $(\geq, \succ)$ est dit :

- Bien fondé si $\succ$ est bien fondé ;
- Stable par substitution si $u \geq v \Rightarrow \sigma u \geq \sigma v$ et si $u \succ v \Rightarrow \sigma u \succ \sigma v$ pour toute substitution σ.
- *(Faiblement) Monotone* si $\forall$ le contexte C, $u \geq v \Rightarrow C[u] \geq C[v]$;

- *Strictement monotone* s'il est monotone et que $u \succ v \Rightarrow C[u] \succ C[v]$ pour tout contexte C.

Remarque I.2.2 La comparaison stricte de deux termes vis-à-vis d'un pré ordre $\geq$ ne correspond pas dans le cas général à l'ordre strict associé $\succ$. Cette partie stricte n'est en effet pas stable par instanciation de termes. L'exemple suivant donné par Enno Ohlebusch illustre cette particularité.

Exemple I.2.1

Considérons la signature $F = \{f : \text{unaire} ; a : \text{constante}\}$ et le système $\Re(F)$ réduit à la seule règle: $f(x) \to f(a)$. On peut définir le pré ordre $\geq$ sur $T(F, X)$ par $\geq = \xrightarrow{*}_{\Re}$. Pour cette relation $f(x) \geq f(a)$ et $f(a) \geq f(x)$; donc par définition de la partie stricte, $f(x) \succ f(a)$. Pourtant en appliquant la substitution $\sigma = \{x \to a\}$, nous obtenons $f(a)\sigma = f(a) \geq f(a) = f(x)\sigma$, ce qui montre que le caractère strict de l'orientation est perdu.

Afin d'obtenir une relation stable sous l'application de substitution on définit la notion de partie *stricte stable*.

Définition I.2.12 Soit $\geq$ un pré ordre sur $T(F, X)$ de partie stricte $\succ$. La partie *stricte stable* $>$ de $\geq$ est définie par : $u > v$ si et seulement si $u\sigma > v\sigma$ pour toute substitution close σ sur n'importe quelle extension de F.

Deux classes d'ordre sur les termes nous intéresseront tout particulièrement du fait de leurs Propriétés caractéristiques.

Définition I.2.13 Un ordre sur les termes $(\geq, \succ)$ est un ordre de :

- Réduction (reduction ordering) s'il est bien fondé, monotone et stable ;
- Réduction stricte s'il est strictement monotone et stable;
- Réduction *faible* s'il est (faiblement) monotone et stable;
- De simplification s'il est stable, strictement monotone et possède la propreté de sous terme: $C[t] \succ t$.

Lorsqu'on veut comparer des termes d'une algèbre $T(F, X)$, on considère des ordres où le premier élément est $\geq$ et le second noté $\succ$ est la partie stricte stable de $\geq$.

I.2.2.3 Extension des ordres sur les termes

On peut obtenir de nouveaux ordres par la composition lexicographique de deux ordres $(\geq^1, \succ^1)$ et $(\geq^2, \succ^2)$.

Définition I.2.14 On définit la *composition lexicographique* $((\geq^1, \succ^1), (\geq^2, \succ^2))_{lex} = (\geq, \succ)$ de deux ordres sur les termes $(\geq^1, \succ^1)$ et $(\geq^2, \succ^2)$ de la façon suivante :

- $u \succ v$ si $u \succ^1 v$ ou si $u \geq^1 v$ et $u \succ^2 v$;
- $u \geq v$ si $u \succ^1 v$ ou si $u \geq^1 v$ et $u \geq^2 v$

Proposition I.2.2

Si $(\geq^1, \succ^1)$ est strictement monotone et si $(\geq^2, \succ^2)$ est faiblement monotone, alors la composition lexicographique $((\geq^1, \succ^1), (\geq^2, \succ^2))_{lex}$ est faiblement monotone.

Si $(\geq^1, \succ^1)$ et $(\geq^2, \succ^2)$ sont strictement monotone, alors $((\geq^1, \succ^1), (\geq^2, \succ^2))_{lex}$ est strictement monotone.

Il est aussi possible d'étendre une relation d'ordre sur les termes d'une algèbre à une relation d'ordre sur les multi ensembles de termes.

Définition I.2.15 Soit $(\geq, \succ)$ un ordre sur les termes. On définit inductive ment l'extension lexicographique de $(\geq, \succ)$ sur les suites de termes comme suit :

$[e_1, \ldots e_n] \succ^{lex} [\,]$ si $n \succ 0$;

$[e_1, \ldots e_n] \succ^{lex} [e'_1, \ldots e'_n]$ si $e_1 \succ e'_1$;

$[e_1, e_2, \ldots e_n] \succ^{lex} [e'_1, e'_2, \ldots e'_n]$ si $e_1 = e'_1$ et $[e_2, \ldots e_n] \succ^{lex} [e'_2, \ldots e'_n]$;

$[e_1, \ldots e_n] \geq^{lex} [\,]$ si $n \geq 0$;

$[e_1, e_2, \ldots e_n] \geq^{lex} [e'_1, e'_2, \ldots e'_n]$ si $e_1 \succ e'_1$;

$[e_1, e_2, \ldots e_n] \geq^{lex} [e'_1, e'_2, \ldots e'_n]$ si $e_1 \geq e'_1$ et $[e_2, \ldots e_n] \geq^{lex} [e'_2, \ldots e'_n]$.

Définition I.2.16 Un multi ensemble est une application M_E d'un ensemble E vers N telle que $\{e \in E \mid M_E(e) \neq 0\}$ est fini. On appelle $M_E(e)$ la multiplicité de e dans M_E.

Les multi ensembles sont souvent notés en extension entre accolades, chaque e apparaissant en $M_E(e)$ occurrences, En l'absence d'ambiguïté on peut omettre la précision d'un ensemble.

On dit qu'un élément e appartient à un multi ensemble M (ce qu'on note alors $e \in M$) si $M(e) \geq 1$. Soient M et N deux multi ensembles; M est inclus dans N (notation $M \subseteq N$) si pour tout e, $M(e) \leq N(e)$.

Enfin $M' = M / N$ est défini par $M'(e) = Max(0, M(e) - N(e))$.

Définition I.2.17 Soient M et N deux multi ensemble de termes, L'*extension lexicographique* d'un ordre sur les termes $(\geq, \succ)$ est un couple $(\geq_{mul}, \succ_{mul})$ défini inductive ment par :

- $M \geq_{mul} M$;
- Si $M \geq_{mul} N$ et si $e \geq e'$ alors $M \cup \{e\} \geq_{mul} N \cup \{e'\}$;
- Si $M \geq_{mul} N$ et si $e \geq e_1,...e_k$ pour $k \geq 0$ alors $M \cup \{e\} \succ_{mul} N \cup \{e_1,...e_k\}$;
- Si $M \succ_{mul} N$ et si $e \geq e'$ alors $M \cup \{e\} \succ_{mul} N \cup \{e'\}$.

I.2.3 Réécriture de termes

Une relation de réécriture est une relation $\rightarrow$ sur les termes, monotone et stable par instanciation. On note $\rightarrow^+$ sa clôture transitive et $\rightarrow^*$ sa clôture réflexive/transitive.

Du fait qu'on ne peut pas représenter une relation de réécriture (qui peut être infinie) en extension, il est suffisant de donner une représentation canonique de cette relation par ce qui est appelé : un *système de réécriture*, souvent simplifié en TRS (de l'anglais « Term Rewriting System » voir [BN 04], [TERESE 03], [KLO 80,82], [DP 01]).

Définition I.2.18 Etant donné une algèbre de termes $T(\Sigma, V)$.

A) Une règle de réécriture sur Σ est un couple (l, r) (noté $l \rightarrow r$) de termes de l'algèbre $T(\Sigma, V)$ vérifiant :

- $l \notin V$ (la partie gauche de la règle n'est pas une variable) ;
- $Var(r) \subseteq Var(l)$.

B) Un système de réécriture $\Re(\Sigma)$ est un ensemble de règles de réécriture sur Σ. Si aucune confusion ne peut être posée pour la signature de base, le système de réécriture est noté simplement $\Re$ pour dire $\Re(\Sigma)$. Le système de réécriture $\Re$ engendre la relation de réécriture $\rightarrow_{\Re}$.

Remarques I.2.3

- $u \to_{\Re} v \Rightarrow \exists (l \to r) \in \Re$, une position p et une substitution σ telles que : $u|_p = l\sigma$ et $v = r\sigma\ |_p$.
- $u|_p$ est appelé un radical et v est le réduit de u par la règle $l \to r$ à la position p. Ce qui est noté $U \xrightarrow[l \to r]{p,\sigma} V$ et dans le cas où ces précisions ne sont pas nécessaires on peut écrire tout simplement $U \xrightarrow[\Re]{p} V$.

D'après la terminologie utilisée par Thomas Arts et Jüirgen Giesl [AG 97], deux types de symboles peuvent être distingués : les symboles *définis* et les symboles *Constructeurs*.

Définition I.2.19 Soit $\Re$ un système de réécriture sur une signature $\sum$. On définit la partition $\sum = D \cup C$ telle que :

- $f \in D \Leftrightarrow \exists$ une règle $l \to r \in \Re$ telle que $\Lambda(l) = f$; (D est un ensemble qui contient les fonctions qui existent à la position racine)
- $C = D / \Sigma$.

Les éléments composant l'ensemble D sont appelés les symboles *définis*, les éléments de C les symboles *constructeurs*.

I.2.3.1 Propriétés Syntaxiques des Systèmes de Réécriture

Une réduction n'est autre que le remplacement d'expressions syntaxiques par d'autres, les caractéristiques syntaxiques donnent intuitivement certaines informations sur les propriétés du système.

Définition I.2.20

- Un système est dit à *branchement fini* si tout radical n'est filtré que par un nombre fini de membres gauches des règles.
- Un terme est dit *linéaire* si chacune de ses variables n'apparaît qu'une seule fois.
- Une règle est dite *linéaire à gauche* si son membre gauche est linéaire.
- Une règle est dite *linéaire à droite* si son membre droit est linéaire.
- Une règle est dite *dupliquante* si elle contient une variable dont le nombre d'occurrences dans la partie droite est plus grand que celui dans la partie gauche.
- Un système de réécriture dupliquant est un système qui comporte des règles dupliquantes.
- Une règle est dite *projective* si son membre droit est réduit à une seule variable.

- Si le membre gauche peut être réduit en plusieurs variables distinctes (par des règles différentes) on dit que le système est projectif ND (*projection non déterministe*).

Exemple I.2.2 On considère le système composé des règles suivantes :

$$\begin{cases} f(x,y,z) \xrightarrow{1} g(y,y) \\ f(x,y,y) \xrightarrow{2} g(x,y) \\ g(x,y) \xrightarrow{3} y \\ h(x) \xrightarrow{4} x \\ f(x,y,z) \xrightarrow{5} h(x) \end{cases}$$

La première règle est linéaire à gauche ; la deuxième règle est linéaire à droite ; la troisième, quatrième et la cinquième règle sont linéaires à droite et à gauche donc elles sont linéaires. Il est facile de vérifier que la première règle est dupliquante et que la troisième, et la quatrième règle sont projectives.

Prenons le terme $f(x,y,z)$.

$f(x,y,z) \rightarrow_5 h(x) \rightarrow_4 x$

$\downarrow_1$

$g(y,y)$

$\downarrow_3$

y

Ce qui montre que le système est projectif ND.

I.2.3.2 Aspect calculatoire des Systèmes de Réécriture

Un système de réécriture fait un calcul, le résultat est toujours un terme sur lequel le calcul ne peut être poursuit.

Exemple I.2.3 Soit le système de réécriture $\Re$ qui spécifie les nombres naturels avec les opérations de l'addition, la multiplication, le successeur et la constante nulle (le zéro) défini par :

$$\begin{cases} A(x,0) \xrightarrow{1} x \\ A(x,S(y)) \xrightarrow{2} S(A(x,y)) \\ M(x,0) \xrightarrow{3} 0 \\ M(x,S(y)) \xrightarrow{4} A(M(x,y),x) \end{cases}$$

Le terme $M(S(S(0)),S(S(0)))$ se réduit à $S(S(S(S(0))))$ par les étapes de calculs suivantes :

$M(S(S(0)),S(S(0))) \rightarrow$
$A(M(S(S(0)),S(0)),S(S(0))) \rightarrow$
$S(A(M(S(S(0)),S(0)),S(0))) \rightarrow$
$S(S(A(M(S(S(0)),S(0))0))) \rightarrow$
$S(S(M(S(S(0)),S(0)))) \rightarrow$
$S(S(A(M(S(S(0)),0),S(S(0))))) \rightarrow$
$S(S(A(0,S(S(0))))) \rightarrow$
$S(S(S(A(0,S(0))))) \rightarrow$
$S(S(S(S(A(0,0))))) \rightarrow$
$S(S(S(S(0))))$.

Remarques I.2.4

- Un terme qui ne peut être réduit par un système de réécriture $\Re$ est en forme normale (au sens du système $\Re$).
- Un terme est $\Re$ normalisable s'il admet une forme normale par le système $\Re$ c'est-à-dire s'il existe une réduction qui aboutit à la forme normale.
- Un terme est dit fortement normalisable si toute réduction mène à une forme normale.
- Un système est normalisant si tout terme est normalisable. Il est fortement normalisable (on dit qu'il termine) lorsque la relation $\rightarrow_{\Re}$ est noethérienne, c'est-à-dire que tout terme est fortement normalisable.

Exemple I.2.4

Soit la signature $\Sigma = \{f : 1; a : 0\}$, et le système $\Re$ défini sur Σ tel que :

$$\begin{cases} f(x) \rightarrow a \\ f(x) \rightarrow f(x) \end{cases}$$

Chaque terme a une forme normale qui est a. mais le système n'est pas fortement normalisant car $f(x)$ peut être réduit infiniment.

I.2.4 Confluence des systèmes de réécriture

Comme nous l'avons vu dans l'exemple du système ND précédent, l'application de règles d'une manière non déterministe peur être une source de problème pour l'unicité du résultat et même pour l'existence de ce résultat de calcul. La propriété de confluence[2] nous permet d'ignorer le choix de la règle à appliquer.

I.2.4.1 Paires Critiques

[2] La propriété de confluence a été définie dans la section I.1

Avant de donner la définition formelle des paires critiques, nous donnons l'exemple d'introduction suivant :

Exemple I.2.5

Le système de réécriture non confluent ci-dessous est tiré de la théorie équationnelle des groupes en orientant les règles.

$$\begin{cases} ex \xrightarrow{1} x \\ I(x)x \xrightarrow{2} e \\ (x.y).z \xrightarrow{3} x.(y.z) \end{cases}$$

Où e représente l'élément neutre et $I(x)$ représente l'inverse de x.

Le rédex $I(x).x$ peut être unifié (après renom mage des variables) avec un sous terme qui n'est pas une variable (s : le sous terme souligné) du rédex $\left(\underbrace{x.y}_{s}\right).z$ ce qui donne comme résultat de l'unification le terme : $(I(x).x).z$. Ce nouveau terme est sujet à deux réductions possibles :

- $(I(x).x).z \xrightarrow{2} e.z$
- $(I(x).x).z \xrightarrow{3} I(x).(x.z)$

La paire de réduits $\langle e.z, I(x).(x.z)\rangle$ est appelée une paire critique, la propriété de confluence dépend étroitement des possibilités de réduction des composants de cette paire.

I.2.4.2 Définition formelle

Soient $\alpha \to \beta$ et $\gamma \to \delta$ deux règles de réécriture tel que α est unifiable (après renom mage des variables) avec un sous terme de γ qui n'est pas une variable. Cela veut dire qu'il existe un contexte $C[\ ]$, un terme non variable t et un unificateur le plus général σ tel que $\gamma \equiv C[t]$ et $t\sigma \equiv \alpha\sigma$. Le terme $\gamma\sigma \equiv C[t]\sigma$ peut être réduit de deux possibles manières : $C[t]\sigma \to C[\beta]\sigma$ et $\gamma\sigma \to \delta\sigma$.

Maintenant la paire de réduits $\langle C[\beta]\sigma, \delta\sigma\rangle$ est appelée une paire critique obtenue par la *superposition* de la règle $\alpha \to \beta$ sur $\gamma \to \delta$. Si $\alpha \to \beta$ et $\gamma \to \delta$ sont les mêmes règles de réécriture on exige que α s'unifie avec un propre non variable sous terme de γ (c'est-à-dire : non $\equiv \alpha$).

Remarques I.2.5

- Un système de réécriture comportant des paires critiques est dit Overlapping ;
- Les paires critiques sont définies modulo renom mage des variables.

Définition I.2.21 Une paire critique $\langle s,t\rangle$ est dite convergente si s et t ont un réduit commun. Dans l'exemple I.1 la paire critique $\langle e.z, I(x).(x.z)\rangle$ n'est pas convergente : le terme $I(x).(x.z)$ est une forme normale et le terme $e.z$ ne peut se réduire qu'à z .

Remarque I.2.6 Dans la théorie des groupes l'égalité des deux composantes de la paire critique peut être dérivable, mais elles n'ont aucun réduit en commun en utilisant le système de réécriture donné dans l'exemple.

I.2.5 Les Systèmes de Réécriture Orthogonaux

Nous avons vu précédemment que la présence des paires critiques dans un TRS détruit la propriété de confluence et nous présentons une catégorie de systèmes qui n'ont pas de paires critique, en d'autres termes, les systèmes dont les règles de réécriture ne se superposent pas. Nous allons aussi imposer la linéarité à gauche ce qui donne comme résultat des règles orthogonales l'une par rapport aux autres. Plus formellement nous avons :

Définition I.2.22

- Un TRS $\Re$ est orthogonal si $\Re$ est linéaire à gauche et ne contient aucune paire critique.
- $\Re$ est faiblement orthogonal si $\Re$ est linéaire à gauche et $\Re$ ne contient que des paires critiques convergentes, i.e. si $\langle s,t\rangle$ est une paire critique alors $t \downarrow s$.

Notons que pour les règles qui sont linéaires à gauche seule le pattern de la règle qui est important. Un problème avec les règles non linéaires à gauche est que leurs applications requirent un test pour l'identité syntaxique de tous les arguments substitués aux variables qui ont plus d'une occurrence. Comme les termes sont largement variés ce test est très laborieux. Un autre problème est que l'absence de la linéarité à gauche peut détruire la propriété de confluence, comme le montre l'exemple suivant :

Exemples I.2.6

1) Soit $\Re$ le système formé par les règles :

$$\begin{cases} D(x,x) \to E \\ C(x) \to D(x, C(x)) \\ A \to C(A) \end{cases}$$

Ce système est faiblement confluent mais pas confluent, nous avons les réductions $C(A) \to^* E$ et $C(A) \to^* C(E)$; $C(E)$et E non pas de réduit commun. Le système $\Re$ n'a pas de paires critiques. En admettant pour le moment que les systèmes orthogonaux sont confluents, la non confluence de $\Re$ est causée alors par la règle non linaire à gauche $D(x,x) \to E$.

2) le système de réécriture suivant est présenté dans [HUE 80]

$$\begin{cases} \infty \to S(\infty) \\ eq(x, x) \to True \\ eq(x, S(x)) \to False \end{cases}$$

Il est à noter que ces règles ne sont pas overlapping. Mais le système n'est pas confluent du moment où $eq(\infty, \infty) \to True$ et aussi $eq(\infty, \infty) \to False$.

La détection des paires critiques est plutôt ambiguë, mais leur absence est facile à prévoir. On va essayer de donner une explication de l'absence des paires critiques d'une façon intuitive. Soit $\Re$ le système représenté par les règles :

$$\begin{cases} F(G(x, S(0)), y, H(z)) \xrightarrow{1} x \\ G(x, S(S(0))) \xrightarrow{2} 0 \\ P(G(x, S(0))) \xrightarrow{3} S(0) \end{cases}$$

Un pattern d'une règle est un contexte avec plusieurs « hole #». Le contexte $F(G(\ \#, S(0), \#, H(\ \#\ )\ )$ est le pattern de la règle (1). Dans le schéma en arbre I.2.2 ce pattern est la partie encadrée.

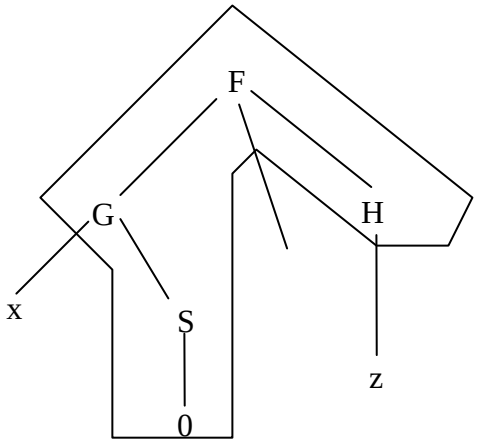


Figure I.2.2 Pattern de la règle (1)

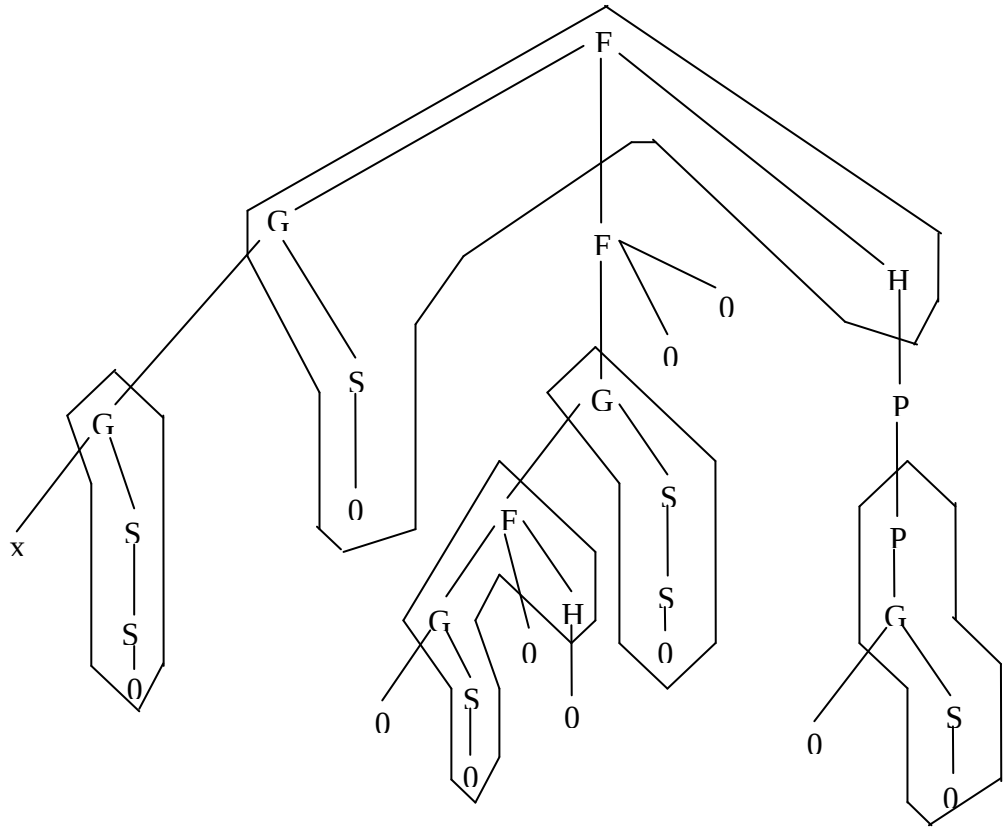


Figure I.2.3. Patterns de règle non overlapping

Le système de réécriture précédent à la propriété que les patterns de ces termes sont non « overlapping ». La figure I.2.3 montre un terme dans le système avec tous ces patterns.

Exemples I.2.7 Systèmes (faiblement) Orthogonaux

Nous allons maintenant donner des exemples de systèmes orthogonaux et faiblement orthogonaux.

- La logique combinatoire basée sur les combinateurs S, K, I , a les règles :

$$\begin{cases} Ap(Ap(Ap(S,x),y),z) \to Ap(Ap(x,y),Ap(y,z)) \\ Ap(Ap(K,x),y) \to x \\ Ap(I,x) \to x \end{cases}$$

Ce système a les patterns de règles comme il est montré dans la figure I.2.4

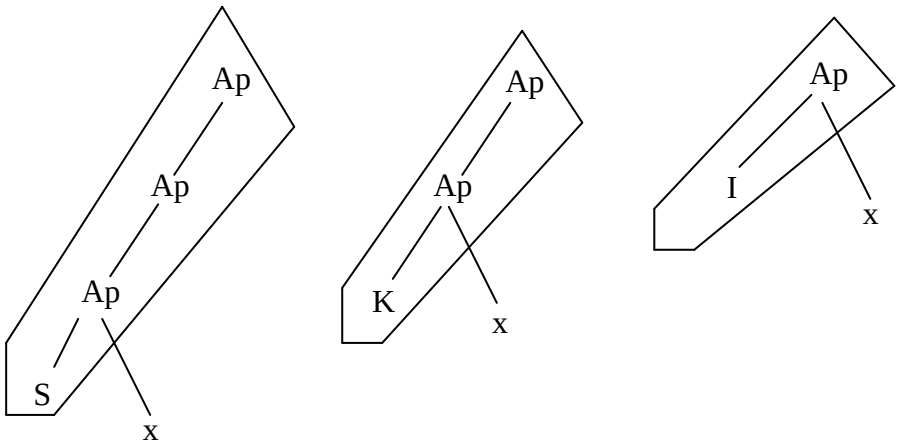


Figure I.2.4 les patterns de la logique combinatoire

D'après le schéma, les patterns ne peuvent pas « overlaper » et comme le système est linéaire à gauche, il est alors orthogonal.

- Considérons les règles du prédécesseur et du successeur :

$$\begin{cases} P(S(x)) \to x \\ S(P(x)) \to x \end{cases}$$

Ces règles constituent un système faiblement orthogonal, il comporte deux paires critiques triviales, une est obtenue de $P(S(P(x)))$ qui est $\langle P(x), P(x) \rangle$, l'autre de $S(P(S(x)))$ et est $\langle S(x), S(x) \rangle$.

I.2.6 Confluence des Systèmes orthogonaux et faiblement Orthogonaux

L'intérêt porté aux systèmes orthogonaux est dû au fait qu'ils sont tous confluents comme le montre les théorèmes suivants :

Théorème I.2.1. Tout système orthogonal est confluent. (Voir [ROS 73]).

Théorème I.2.2 Tout système faiblement orthogonal est faiblement confluent.

Preuve : voir [TERESE 03].

I.2.7 Terminaison des systèmes orthogonaux

Il est aussi possible de connaître la propriété de terminaison des systèmes orthogonaux en étudiant la relation qu'ils induisent en suivant une *stratégie* d'application particulière.

I.2.7.1 Stratégie de réduction

Il est possible de restreindre la relation de réécriture en réduisant le non déterminisme et cela en imposant la ou les règles à appliquer à un terme parmi toutes les règles du système ce qui donne ce qu'on appelle une *stratégie de réduction*. On s'intéresse particulièrement à une stratégie proche de l'évaluation des langage fonctionnels : la stratégie « *innermost* ».

Définition I.2.23 (Stratégie « *innermost* »)

La stratégie « innermost » réécrit un radical de l'ensemble de tous les radicaux les plus internes. Cette stratégie met en première priorité la normalisation des arguments avant l'évaluation de la fonction ce qui correspond dans ce sens à l'*évaluation stricte* de langage comme CAML [XP 93] ou SML [RTH 97] contrairement à l'évaluation paresseuse comme dans HASKELL [SPJ 03]. Le choix d'une stratégie peut donner aux systèmes des propriétés qu'ils n'auraient pas dans le cas général. On a en particulier pour les systèmes orthogonaux le résultat suivant.

Théorème I.2.3 (O'Donnell 77)

Un système orthogonal est fortement normalisant si est seulement si il est fortement normalisant pour la stratégie « innermost » [O'D 95].

Exemple I.2.8 Considérons le système :

$$\begin{cases} f(g(x)) \to f(g(x)) \\ g(x) \to a \end{cases}$$

Ce système n'est pas orthogonal puisqu'il admet la paire critique $\langle f(g(x)), a \rangle$.

Dans le cas général il autorise la réduction infinie $f(g(x)) \to f(g(x)) \to \ldots$, mais il termine sur tout terme pour la stratégie innermost. Celle-ci va en effet éliminer tous les symboles g nécessaires au radical $f(g(x))$ et cela à l'intérieur du radical.

I.3 Terminaison Des Systèmes de Réécriture

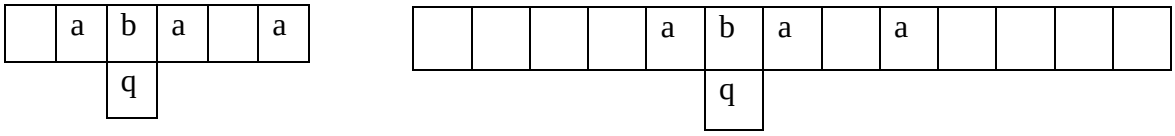
I.3.1 Problème de l'arrêt et indécidabilité

La propriété de terminaison, garantie de l'existence d'un résultat de calcul sur tout terme est nécessaire au préalable pour étudier toute autre propriété, mais d'une manière générale elle n'admet pas d'algorithme de décision. Le problème de l'arrêt de machines de Turing s'y réduit.

I.3.1.1 Réécriture et machine de Turing

Nous allons voir comment coder n'importe quelle machine de Turing sous forme d'un système de règles. Dershowitz a proposé à ce sujet un codage en deux règles seulement [DER 87], nombre encore réduit par Dauchet ; il est en fait possible de se limiter à une seule règle linéaire à gauche [DAU 92]. On reprend ici une traduction simple, des travaux de Huet & Lankford [HD 78], en quelques règles de réécriture de mots.

a)

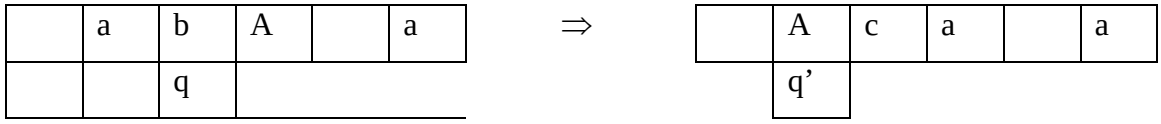


Configuration configuration équivalente

b)

Règles de transition (déplacement à droite) Règles de transition (déplacement à gauche)

c)



Transition entre configuration

Figure I.3.1

Une machine de Turing M est un triplet (Q, S, δ) :

1. Q est un ensemble finie d'états $\{q_0, q_1, ..., q_n\}$
2. $S = \{\bullet, s_1, s_2, ..., s_m\}$ est un ensemble se symbole où $\bullet$ est le symbole blanc tel que $Q \cap Q = \{\ \}$
3. δ une fonction partielle de $Q \times S$ vers $Q \times S \times \{L, R\}$ appelée fonction de transition, les symboles L, R représentent le déplacement à gauche et à droite.

La machine opère sur une bande infinie dans les deux sens. Cette bande peut être représentée par une fonction $\tau : \mathrm{Z} \to S$ tel que $\{i \in \mathrm{Z} \mid \tau(i) \neq \bullet\}$ est finie. A chaque instant la machine est dans un état, disons q, lisant une position p dans la bande qui contient un symbole s ; ceci correspond à une configuration. Si $\delta(q, s) = \langle q', s', \mathrm{R} \rangle$, l'état de la machine change de q à q', le symbole s est remplacé par s' et la machine lit la position immédiatement à droite de p (ou à gauche le cas de L). Ce processus se répète infiniment ou jusqu'à la lecture d'un symbole s où la machine est dans un état q tel que $\delta(q, s)$ est indéfinie alors la machine s'arrête. Des exemples de configurations sont montrés dans la figure I.3.1. L'état q est attaché à la position en lecture par la tête. Souvent les chaînes de caractères sont utilisées pour encoder les configurations. Une chaîne de la forme $w_1 q w_2 \in S^*, q \in Q$ appelée description instantanée, désigne la machine dans un l'état q avec la bande $w_1 w_2$ (le reste de la bande contient des blancs).la tête lit le 1er symbole à droite de q, i.e le 1er symbole de w_2 ou $\bullet$ si $w_2 = \varepsilon$ (le mot vide). Par exemple la configuration donnée dans Fig. I.3.1 (b) correspond à $\bullet aqba \bullet a$ et $\bullet\bullet\bullet\bullet aqba \bullet a \bullet\bullet\bullet\bullet$.

A une machine de Turing $M = \langle Q, S, \delta \rangle$ on associe un système de réécriture R_M comme suit :

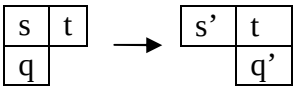
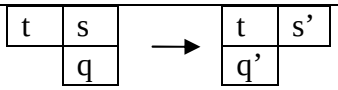
A chaque état $q \in Q$ est associé une fonction binaire dénotée par la même lettre. A chaque lettre $s \in S$ correspond une fonction unaire dénoté par la même lettre. L'alphabet du système R_M est augmenté avec le caractère Δ qui représente une infinité de blancs.

Un mot $w \in S^*$ est traduit en un terme $\phi(w)$ comme suit :

$\phi(\varepsilon) = \Delta$ où ε est le mot vide.

$\phi(sw) = s(\phi(w))$ Pour $s \in S, w \in S^*$.

Les transitions peuvent donc être codées de la manière suivante :

(L/R) déplacement	transition	Règle de réécriture
$\delta(q,s) = \langle q', s', R \rangle$		$q(x,sy) \to q'(s'x, y)$
$\delta(q,s) = \langle q', s', L \rangle$		$q(tx,sy) \to q'(x, ts'y)$

On peut représenter une machine de Turing comme un système de réécriture en utilisant les descriptions instantanées ce qui donne naturellement un système de réécriture de chaînes de caractères. Nous utilisant ici une autre représentation en codant les états comme des fonctions binaires.

Il reste à ajouter les cas dégénérés, c'est-à-dire faisant intervenir le caractère blanc, pour obtenir la simulation désirée, pour cela on peut travailler modulo l'identité $\Delta = \bullet\, \Delta$ ou ajouter des règles additionnelles qui sont déduites des règles principales en prenant en compte l'identité $\Delta = \bullet\, \Delta$.

On obtient ainsi une correspondance entre les configurations successives de la machine et les réductions à l'aide du système R_M. Il suffit de réduire le problème d'arrêt de la machine de Turing au problème de décision de la normalisation forte du système réécriture obtenu.

La clé de preuve consiste à :

1. si α, β sont deux configuration tel que $\alpha \mapsto \beta$ alors $t \to s$ (t, s sont les représentants de α, β respectivement).
2. si t, s sont deux termes tel que $t \to s$ alors $\alpha \mapsto \beta$ (avec : t représente α et s représente β).

I.3.1.2 Preuve de terminaison

La terminaison est clairement une propriété indécidable, plusieurs méthodes ont été développées permettant de prouver la normalisation forte dans la plus part des cas de systèmes rencontrés dans la pratique. D'une manière générale on montre la terminaison d'un programme en prouvant qu'au cours d'une exécution, un variant de celui-ci décroît strictement vis-à-vis d'un ordre bien fondé. De part la nature de l'ordre, ce variant ne peut décroître indéfiniment, et donc le programme s'arrête sur un résultat.

Exemple I.3.1 Pour le programme donné par la seule règle $f(f(x)) \to x$ il est suffisant de prendre pour variant le nombre de symboles f du terme à réduire à chaque pas de réduction.

Dans un formalisme de programme quelconque, la recherche de ce variant peut reposer très lourdement sur l'expérience et l'intuition de qui cherche à montrer la terminaison.

Dans le cas des systèmes de réécriture, la preuve peut toutefois être largement simplifiée par le fait que le système R décrit lui-même une relation entre les termes. Si cette relation $\to R$ peut être plongée dans une relation d'ordre bien fondée, la proposition (I.2.3) entraîne alors que $\to R$ est bien fondée.

On peut remarquer en particulier qu'un système R, s'il termine, décrit une relation $\to R$ bien fondée par définition ; c'est l'ordre de réécriture. Cet ordre est à rapprocher du variant associé au programme qui peut, puisque le programme termine par hypothèse, être la différence entre le nombre de pas maximal possible sur l'exécution considérée et le nombre de pas déjà effectués.

Dans le cadre de cette approche plus générale, la recherche ne porte donc plus sur une mesure de complexité qui décroît au fur et à mesure des applications de R mais directement sur un ordre bien fondé contenant $\to R$.

Exemple I.3.2 En reprenant dans ce contexte l'exemple I.2.1, on peut montrer que le système $f(f(x)) \to x$ termine en considérant le nombre de f dans le terme avec l'ordre naturel sur les entiers, pour toute instanciation σ, $f(f(x))\sigma$ est supérieur à σx.

Théorème I.3.1 **[DER 82]**. Un système R termine si et seulement s'il existe un ordre de réduction $\succ$ tel que $\to_R \subseteq \succ$.

Preuve. Immédiat puisqu'en particulier pour un système R fortement normalisant, l'ordre de réécriture est un ordre de réduction.

Un ordre apte à prouver la terminaison d'un système de réécriture doit donc se plier à certaines contraintes. La recherche d'une preuve de terminaison revient alors dans la plupart des cas à déterminer ces contraintes d'ordre et à les résoudre. La génération de ces contraintes s'est

historiquement d'abord limitée à l'orientation stricte des règles du système étudié. On cherchait alors à trouver directement un ordre convenable pour le théorème (I.2.3) à l'aide de ce que nous désignerons dans la suite comme « Méthodes classiques ». Une analyse plus fine de la structure des termes non fortement normalisables par un système R à donner naissance aux critères de terminaison à l'aide de paires de dépendance. Les contraintes requises, même si elles sont plus nombreuses, deviennent moins complexes ; il suffit d'orienter largement les règles de R, la décroissance stricte n'étant nécessaire que sur les dites paires de dépendance.

I.3.2 Méthodes classiques

I.3.2.1 Terminaison à l'aide d'ordres de simplification

L'importance des travaux consacrés aux ordres de simplification, qui concernent la définition de nouveaux ordres ou l'étude de leurs propriétés, vient en particulier du fait que parmi les ordres susceptibles d'être utilisés dans les preuves de terminaison ils étaient il y peu les seuls qu'on pouvait déterminer automatiquement.

Théorème I.3.2 **[DER 87]** Sur une signature finie, tout ordre de simplification est bien fondé. En particulier s'il existe un ordre de simplification $>$ tel que pour toute règle ($l \to r$ d'un système R) : $l > r$ alors R termine.

Définition I.3.1 L'ordre dit de plongement $\trianglelefteq$ est défini par :

$t = f(t_1,...,t_n) \trianglelefteq g(t'_1 t_1 ...,t'_m) = t' si$

1. Il existe $i, 1 \le i \le m$ tel que $t \trianglelefteq t'_i$, ou bien
2. $f = g$ et pour tout $j, 1 \le j \le n, t_j \trianglelefteq t'_i$

Remarque I.3.1 Le plongement, bien fondé par le théorème de Kruskal, est un ordre de simplification et tout ordre de simplification le contient.

Deux termes sont en fait en relation par l'ordre de plongement s'il est possible d'obtenir l'un à partir de l'autre simplement à l'aide de projections. On peut donc considérer les systèmes qui préservent la terminaison quand on autorise le passage au sous terme.

Définition I.3.2 On dit d'un système R qu'il termine simplement si

$R \cup_{f|AR(f)\ge l} \{f(x_1,...,x_n) > x_1 | 1 \le i \le n\}$ est fortement normalisant.

Cette définition est bien équivalente à une preuve de terminaison par le théorème (I.2.3) et à l'aide d'un ordre de simplification.

Lemme I.3.1 (Kurihara & Ohuchi). [MA 90].Un système sur une signature finie termine simplement si et seulement s'il existe un ordre de simplification > tel que pour toute règle $l \to r \in R$ on ait $l \succ r$

I.3.3 Critères des paires de dépendance

La démarche qui consiste à destiner les ordres que nous venons de voir à l'application directe du théorème (I.2.3) est trop restrictive. Art & Giesl proposent en 1997 une analyse plus fine de la structure des termes non fortement normalisables qui permet de dégager de nouvelles contraintes d'ordres, plus nombreuses mais plus souples [AG 97].

Ils consistent en effet qu'à partir de tout terme donnant lieu à une réduction infinie on peut obtenir une dérivation d'une forme bien particulière. Prouver qu'il n'existe pas de dérivation de la sorte, quel que soit le terme considéré, suffit alors – en montrant par là que tous les termes ne donnent lieu qu'à des réductions finies – à mettre en évidence la terminaison du système. Remarquons que grâce à la plus grande souplesse de ces contraintes, on peut davantage en espérer une résolution automatique.

I.3.3.1 Paires de dépendance

Principe de base. Un argument de minimalité suffit pour montrer si un terme ***t*** donne lieu à une réduction infinie, alors il admet un sous terme non fortement normalisable $f(u_i,...,u_n)$ tel que tous les u_i sont pourtant fortement normalisables.

On obtient ainsi l'existence d'une réduction particulière comme celle de la figure I.3.1 Pour garantir la terminaison il reste à prouver qu'une telle dérivation ne peut se produire. Cette dérivation reposant essentiellement sur l'existence de sous termes pouvant déclencher une réduction, c'est cette propriété que nous allons chercher à contenir dans de nouvelles contraintes.

Définition I.3.3 Considérons un système de réécriture $R(F)$ une paire $\langle l, s \rangle$ tel qu'il existe une règle $l \to r \in R$ ou s est un sous terme de r dont le symbole de tête $\Lambda(s)$ est défini est une paire de dépendance de $l \to r \in R$

L'union des paires dépendance de toutes les règles de R forme l'ensemble des paires de dépendance de R, notée $DP(R)$. (Voir [TERESE 03] et [AG 2000])

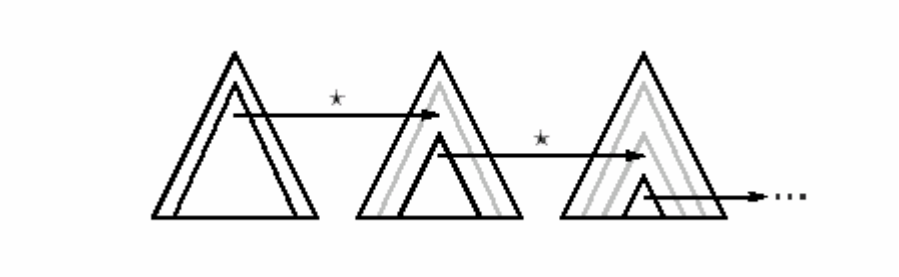


Fig. I.3.2 Structure de réduction particulière

Il est en fait possible d'afficher cette définition en distinguant les symboles de tête des membres de paires. Cette différenciation est justifiée en particulier car la position marquée délimite le terme minimal non fortement normalisant dans la réduction particulière, figure I.3.2 Nous utilisons ici et sauf mention contraire ces paires « marquées » et préciserons le cas échéant leurs avantages et les difficultés techniques pouvant en résulter.

Définition I.3.4 Soit $R(F)$ un système de réécriture. Pour chaque symbole $f \in F$ défini dans R on considère sa copie marquée $\hat{f}$ telle que $\hat{f} \notin F$.

On note $\hat{f}\ R = F \bigcup_{f \in F} \left\{ \hat{f} \right\}$ f défini dans R

La mention du système de réécriture de référence étant omise en l'absence d'ambiguïté.
Si un terme s de $T(F, Var)$ a en tête un symbole défini, $\hat{s}$ représente le terme de $T(\hat{F}, Var)$ obtenu en remplaçant $\Lambda(s)$ par sa copie marquée. Nous appellerons *marquage* cette opération.

Définition I.3.5 Pour un système de réécriture $R(F)$, une paire $(\hat{l}, \hat{s})$ telle qu'il existe une règle $l \to r \in R$ où s est un sous terme de r dont le symbole de tête $\Lambda(s)$ est défini est une paire de dépendance marquée de $l \to r$.

Définition I.3.6 Une chaîne de dépendance d'un système $\Re$ est une séquence $..., (S_j, t_j), ...$ munie d'une substitution σ telle que pour deux paires (S_i, t_i), (S_{i+1}, t_{i+1}) consécutives quelconque : $t_i\sigma \xrightarrow{\neq A}{}^{*} S_{i+1}\sigma$

Les paires de dépendance sont définies modulo renom mage des variables, Nous considérerons toujours que, dans une chaîne de dépendance, les variables sont distinctes d'une paire à l'autre. La chaîne de dépendance permettent en particulier de décrire la réduction de forme particulière qui nous intéresse, En effet, dans une telle dérivation les DP relient le terme minimal non

fortement normalisant à un sous- terme de son réduit en tête, lui- même minimal et non fortement normalisant, comme illustré par la figure I.3.2.

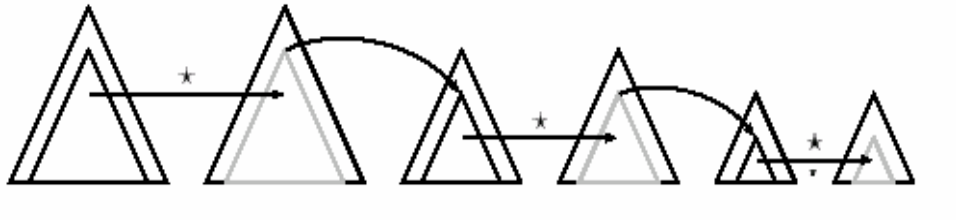


Fig. I.3.3 Réduction avec paires de dépendance

Les chaînes de dépendance que nous rencontrerons dans la suite seront dites « minimales ». Dans le sens où elles correspondront toujours à la réduction particulière du principe de base (fig. I.3.3).

Définition I.3.7 Une chaîne de dépendance est minimale si les sous- termes propres de toute instance d'un membre gauche de paire sont fortement normalisables. Puisqu'elle est capable de décrire la réduction minimale particulière dont le caractère fini ou infini est celui de la relation $\xrightarrow[R]{}$ nous allons désormais nous intéresser à la relation $\xrightarrow[R]{} \bigcup \xrightarrow[DP(R)]{\Lambda}$

I.3.3.2 Critères de terminaison

Ainsi passés des dérivations aux chaînes, il nous reste à trouver un ordre bien fondé dans lequel plonger notre chaîne de dépendance minimale. Les contraintes sur les pas de réécriture (et donc les règles) sont considérablement allégées : si les règles et les paires doivent encore décroître, il suffit désormais que seule la décroissance des pas de DP soit stricte, la terminaison étant alors assurée par le caractère fini des réduction entre instance de paires. Les paires de dépendance permettent de définir un critère de terminaison, c'est- à- dire une condition à la fois nécessaire et suffisante pour garantir qu'un système est fortement normalisant**.**

Théorème I.3.3 [AG 97].Complétude.

Si un système $\Re$ est fortement normalisant, alors il n'existe aucune chaîne de dépendance infinie.

Théorème I.3.4 [AG 97]. Correction.

Soit $\Re$ un système de réécriture s'il n'existe aucune chaîne de dépendance de $\Re$ infinie, alors $\Re$ est fortement normalisant.

Ce théorème de correction admet un corollaire définissant de nouvelles contraintes d'ordre.

Corollaire I.3.1 [AG 97]. Soit R un système d'écriture, s'il existe un ordre de réduction faible. Sur les termes ($\succeq$, $\succ$) tel que :

1. $l \succeq$ r pour toute règle $l \to r \in R$ et
2. $s \succ t$ pour toute paire $\langle s, t \rangle \in DP(R)$.

Alors R est fortement normalisant.

Preuve. On a en effet $\to R \subseteq \succeq$. s'il existait une chaîne de dépendance infinie, on aurait par définition une substitution σ telle que

$$t_{i\sigma} \xrightarrow[R]{} \neq \Lambda^* s_{i+1}\sigma$$

Pour tout i est donc une suite infinie.

$$s_i\sigma \succ t_i \sigma \succeq s_i +_1 \sigma \succeq \ldots$$

Ce qui serait en contradiction avec l'hypothèse : $\succ$ bien fondé.

I.3.3.3 Graphes de dépendance

La condition de stricte décroissance sur *toutes* les paires de dépendance est parfois encore trop forte. En effet, certaines paires ne peuvent apparaître qu'un nombre fini de fois dans une chaîne infinie. Les détecter, c'est se donner les moyens d'affaiblir les contraintes sur l'ordre et ainsi permettre un traitement plus efficace.

Exemple I.3.3

Considérons le système réduit à la seule règle $f(f(x)) \to f(s(f(x)))$.

Ses deux paires de dépendance sont :

$$\langle \hat{f}(f(x)), \hat{f}(x) \rangle \quad (1)$$

$$\langle \hat{f}(f(y)), \hat{f}(s(f(y))) \rangle \quad (2)$$

On ne trouvera toutefois jamais plusieurs occurrences de la seconde paire dans une chaîne puisqu'il n'existe aucune substitution σ telle que $\hat{f}(s(f(y))\sigma) \to^* \hat{f}(f(x))\sigma$. Il n'est donc pas nécessaire d'en réclamer une orientation.

Afin de détecter, pour prouver la terminaison d'un système R, les paires décisives de DP(R), on construit un graphe dont les nœuds sont des paires et tel que les paires qui peuvent apparaître consécutivement dans une chaîne soient reliées. Toute chaîne infinie correspondant donc à un

chemin infini dans le graphe qui est fini si le système lui-même l'est et qui par conséquent doit dans ce cas contenir des cycles.

Dans la suite de cette section, les systèmes considérés sont tous finis.

Définition I.3.8 Soit R un système de réécriture. On appelle graphe de dépendance de R le graphe G dont les nœuds sont les paires de dépendance de R et tel que $(\langle s,l\rangle,\langle s',l'\rangle)\in E(G)$ si et seulement si il existe une substitution σ vérifiant.

$$t\sigma \xrightarrow{\neq \wedge}{}^{*} s'\sigma$$

Prouver la terminaison, revient en fait à vérifier la décroissance des paires qui interviennent dans les cycles pourvu qu'au moins l'une d'elles décroissent strictement.

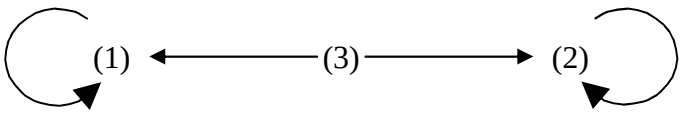
Exemple I.3.4 [AG 97] Considérons un système de division des entiers de Peàno *:*

$$\begin{cases} x-0 \to x \\ S(x)-S(y) \to x-y \\ 0 \div S(y) \to 0 \\ S(x) \div S(y) \to S((x-y) \div S(y)) \end{cases}$$

Il n'existe pas d'ordre de simplification pour ce système. On extrait trois paires de dépendance :

$$\langle S(x) \doteq S(y), x \doteq y\rangle \qquad (1)$$
$$\langle S(x) \dot{\div} S(y), (x-y) \dot{\div} S(y)\rangle \qquad (2)$$
$$\langle S(x) \dot{\div} S(y), x \doteq y\rangle \qquad (3)$$

Qu'on peut agencer en :



Puisque (3) n'est pas sur un cycle du graphe, son orientation n'est pas à considérer.

Le raffinement des graphes préserve la correction des critères par paires de dépendance. Il suffit en faite de limiter l'analyse aux cycles élémentaires du graphe considéré.

Théorème I.3.5 [AG 97].Critères de correction avec graphes.
Soient R un système de réécriture fini et G son graphe de dépendance. S'il existe un pré ordre $\succeq$ de réduction faible tel que :

1. $l \succeq r$ pour toute règle $l \to r \in R$,
2. $s \succeq t$ pour tout cycle élémentaire C de G et pour toute paire $\langle s,t \rangle \in C$
3. Pour tout cycle élémentaire C de G il existe une paire $\langle s,t \rangle \in C$ telle que $s \succ t$,

Alors R est fortement normalisant.

Preuve. Un graphe de dépendance est fini car les paires sont en nombre fini. Le chemin associé à une chaîne infinie passe donc un nombre infini de fois par un cycle élémentaire. Les cycles élémentaires sont de longueur finie donc à chaque fois une décroissance stricte est effectuée, se qui contredit le caractère bien fondé de l'ordre. On conclut par le théorème I.5.2.

Il n'est pas possible de déterminer automatiquement le graphe de dépendance : l'existence de σ telle que $s\sigma \to^* t\sigma$ pour s et t données est indécidable en général. On introduit donc la notion de graphe de dépendance approché dont on sait qu'il contient le graphe de dépendance. Le théorème de correction avec graphe reste bien sûr valable sur ce graphe approché.

I.3.3.4 Approximation du graphe

Plusieurs d'approximations des graphes de dépendance ont été proposées. L'approximation originale de Arts & Giesl repose l'unifiabilité des membres gauches avec une transformation syntaxique des membres droits et admet une amélioration par *narrowing* [CUR 86]. Toyama et Kusakari [KEI 00] utilisent les concepts de réduction ω et Ω. Middeldorp [MID 01] quant à lui met en jeux des automates d'arbre.

Nous décrivons ici la méthode de Arts & Giesl souvent suffisamment puissante et peu coûteuse.

On veut déterminer un ensemble d'arcs qui contient le graphe de dépendance. Les seuls symboles de racine stables par réduction sont les constructeurs : Le critère de sélection retenu quant aux paires à considérer est la possible unification des membres dont les sous termes ayant des symboles définis à la racine ont été remplacés par des variables distinctes *quelques soient les occurrences.*

Définition I.3.9 Soit $R(F)$ un système de réécriture. Pour tout terme t,

- $CAP(t)$ désigne le terme obtenu en remplaçant par des variables des sous termes de t dont le symbole à la racine est défini dans R.
- $REN(t)$ est le terme obtenu en remplaçant les variables de t par de nouvelles variables distinctes (pour toute concurrence).

Le terme t est *connectable* à un terme t' si $REN(CAP(t))$ et t' sont unifiables.

Par unification syntaxique sur les termes modifiés par REN et CAP on détermine des arcs entre les paires de dépendance du système. Le graphe obtenu contient bien le graphe de dépendance.

Proposition I.3.1 Soit $R(F)$ un système de réécriture. S'il existe une substitution σ telle que **:**

$t\sigma \xrightarrow[R]{\neq \wedge^*} t'\sigma$, Alors t est connectable à t'.

Preuve. Par induction sur la structure de t, supposant que $t\sigma$ se réécrive en un terme t'' par Le système R.

- Si t est une variable ou si son symbole de tête est défini alors $REN(CAP(t))$ est sous variable et donc unifiable à t''.
- Si t a pour symbole de tête un constructeur c, on peut alors poser $t = c(...,t_i,...)$. Ainsi $(REN(CAP(t)))$ s'écrit $c(...,REN(CAP(t_i)),...)$. Comme un terme dont la racine est un constructeur ne peut se réduire qu'en un (autre) terme dont la racine est le même constructeur, t'' s'écrit $c(...,t''_i,\cdots)$ avec les $t_i\sigma$ se récrivant en t''_i. Par hypothèse et par distinction des variables après application de REN, $REN(CAP(t))$ est unifiable à t''.

On a le résultat pour $t\sigma \to^* t''$, donc en particulier pour $t\sigma \to^* t'$, il vient ainsi que $REN(CAP(t))$ est unifiable à $t'\sigma$ et finalement, comme il ne contient que de nouvelles variables, à t'.

Remarque I.3.2 Il s'agit bien d'unification et pas de filtrage : prenons $s = f(x,s(y))$ et $t = f(s(u),v)$ où f est le seul symbole défini, s et t sont unifiables mais s ne filtre pas t. Pourtant avec $R:\{f(x,y) \to s(x)\}$ et $\sigma = \{x \to f(u,u), v \to s(y)\}$ on peut avoir la réduction

$$s\sigma = f(f(u,u),s(y)) \to t\sigma \xrightarrow[R]{\neq \wedge^*} t'\sigma,$$

Enfin, puisque plusieurs règles peuvent s'appliquer à un même terme, il est important de renommer chacune des occurrences des variables pour ne pas obtenir de conclusion erronées

Exemple I.3.5 On Considère le système de Toyama [TOY 87] :

$$\begin{cases} G(0,1) & \to & 0 \\ G(0,1) & \to & 1 \\ f(0,1,x) & \to & f(x,x,x). \end{cases}$$

Ce système ne termine pas :

$$f(G(0,1),G(0,1),G(0,1)) \to f(0,G(0,1),G(0,1)) \to f(0,1,G(0,1)) \to f(G(0,1),G(0,1),G(0,1)) \to \ldots$$

On ne peut extraire de cet ensemble de règles qu'une seule paire de dépendance : $\langle \hat{f}(0,1,x), \hat{f}(x,x,x) \rangle$ et si on ne renommait pas toutes les occurrences, on ne trouverait pas d'arête et donc pas de cycle sur le graphe. On pourrait alors conclure à tort que le système termine.

I.3.4 Ordres et Terminaison

Les ordres sont très utilisées dans la recherche de terminaison des systèmes de réécritures, ils se divisent en deux grandes classes : les *ordres syntaxiques* qui comparent les termes de point de vue de leur formes et les *ordres sémantiques* qui interprètent les termes dans un domaine D muni d'un ordre $\succ_D$.

I.3.4.1 Ordres syntaxiques

L'idée de base de ces méthodes est de définir un ordre sur l'alphabet de base et de ne prendre en considération de la structure des termes lors de la comparaison.

Les ordres les plus utilisés dans ce cadre sont les *ordres sur les chemins* où on définit au préalable une *précédence*, c'est-à-dire un ordre (arbitraire) sur les symboles de la signature (les symboles de fonctions).

Ces ordres ont été introduits par Nachum Dershowitz et David Plaisted (voit [DER 79], [PLA 78] et [DAL 78]), ils reposent sur l'idée qu'un terme u est inférieur à un terme v s'il est composé de sous–termes inférieurs au sens de l'ordre à ceux de v et que si les sous termes de u sont des arguments de fonctions, ces dernières doivent être inférieur au sens de la précédence à ceux de v.

Définition I.3.10 Soit $\sum$ une signature. On appelle *précédence* un pré ordre sur $\sum$.

Une *fonction de statut admissible* pour une précédence $\geq$ est une application S de $\sum$ vers $\{mul; lex\}$telle que $f \cong g$ entraîne $S(f) = S(g)$ et si $S(f) = S(g) = lex$, alors f et g ont la même arité.

I.3.4.1.1 Ordre Récursif sur les Chemins

Soit $\sum$ une signature V et un ensemble dénombrable de variables, soient $\geq$ une précédence et S une fonction de statut admissible pour $\geq$. L'ordre récursif sur les chemins (RPO) est la relation $\geq_{RPO}$ sur $T(\sum, V)$ définie par $u \geq_{RPO} v$ si et seulement si :

- $u = x \in V$ et $v = x$ ou
- $u = f(u_1,...,u_n)$ avec $f \in \sum$ et
  - $u_i \geq_{RPO} v$ pour un $i, 1 \leq i \leq n$ ou
  - $v = g(v_1,...,v_n)$ avec $g \in \sum$ et
    - $f \succ g$ et pour tout $j, 1 \leq j \leq m, u \succ_{RPO} v_j$ ou
    - $f \approx g$ et
      - $S(f) = mul$ et $\{s_1,...,s_n\}(\geq_{RPO})_{mul}\{v_1,...,v_n\}$ ou
      - $S(f) = lex \Rightarrow n = m$ et $(u_1,...,u_n)(\geq_{RPO})_{lex}(v_1,...,v_n)$ avec pour tout $j, 1 \leq j \leq m, u \geq_{RPO} v_j$ et $s \succ_{RPO} v$ si $u \geq_{RPO} v$ et $u \overline{\geq_{RPO}} v$.

Le RPO peut être utilisé pour prouver la terminaison des systèmes de réécritures.

Proposition I.3.2 DER 82] et [DER 87]) RPO est un ordre de simplification.

Exemple I.3.6 La terminaison du système décrivant la fonction d'Ackerman

$$\begin{cases} Ack(0,x) \to s(x) \\ Ack(s(x),0) \to Ack(x,s(0)) \\ Ack(s(n),s(m)) \to Ack(n,Ack(s(n),m)) \end{cases}$$

Peut être montré par RPO avec la précédence $Ack \succ s$ où $S(Ack) = lex$. En appliquant par exemple les clauses du RPO sur la dernière règle : $s(n) \succ_{RPO} n$ car n est un sous terme de $s(n)$, il reste à vérifier que $Ack(s(n),s(m)) \succ_{RPO} Ack(s(n),m)$, ce qui est vrai car $s(m) \succ_{RPO} m$ par la propriété du sous terme de l'ordre.

I.3.4.2 Ordres Sémantiques

L'idée de base des méthodes sémantiques est d'interpréter les termes dans un domaine muni d'un ordre bien fondé.

Définition I.3.11 Soient $T(\sum,V)$ une algèbre de termes et D un domaine muni d'un ordre $\geq_D$. On définit une interprétation homomorphique Φ par l'association à chaque symbole $f \in \sum$ d'arité n d'une fonction $\|f\|_\Phi : D^n \rightarrow D$ étendue à tous les termes par :

$$\Phi(x) = x$$
$$\Phi(f(u_1,...,u_n)) = \|f\|_\Phi(\Phi(u_1),...,\Phi(u_n))$$

Les termes interprétés peuvent être alors comparés par l'ordre d'évaluation sur les fonctions sur le domaine D.

I.3.4.3 Ordres d'évaluation

Soit D un domaine muni d'un ordre $\geq_D$. L'ordre d'évaluation $\geq_D$ est définit par :

- $f \geq_D g \Leftrightarrow \forall(x_1,...,x_n) \in D^n,\ f(x_1,...,x_n) \geq_D g(x_1,...x_n)$;
- $f \succ_D g \Leftrightarrow \forall(x_1,...x_n) \in D^n,\ f(x_1,...x_n) \succ_D g(x_1,...x_n)$.

Remarque I.3.3 L'ordre $\succ_D$ de la définition précédente correspond bien à la partie stricte stable de $\geq_D$ mais n'est pas sa partie stricte associé.

Lemme I.3.2 L'ordre $(\geq_\varphi, >_\varphi)$ est stable par instanciation.
Preuve : Soient u et v deux termes tels que $u >_\varphi v$ et σ une substitution. Par définition de $>_\varphi$, $\forall x_1,...x_n, \Phi(u)(x_1,...x_n) \succ_D \Phi(v)(x_1,...,x_n)$. En particulier :

$$\Phi(u\sigma)(x_1,...,x_n) = \Phi(u)(x_1\sigma,...,x_n\sigma) \succ_D \Phi(v)(x_1\sigma,...,x_n\sigma) = \Phi(v\sigma)(x_1,...,x_n) \Rightarrow u\sigma >_\varphi v\sigma$$

Idem pour $\geq_\varphi$

Lemme I.3.3 Si pour chaque symbole f la fonction $\|f\|_\Phi$ est croissante (respectivement strictement croissante) pour chacun de ces arguments, alors $\geq_\varphi$ (respectivement $>_\varphi$) est monotone.

Preuve : Il est vrai que si $\Phi(s) \geq_D \Phi(t)$ et si $\|f\|_\Phi$ est croissante (respectivement strictement croissante) pour chacun de ces arguments, on :

$$\Phi(f(u_1,...,s,...,u_n)) =$$
$$\|f\|_\Phi(\Phi(u_1),...,\Phi(s),...\Phi(u_n)) \geq_D \|f\|_\Phi(\Phi(u_1),...,\Phi(t),...,\Phi(u_n)) = \Phi(f(u_1,...,t,...u_n))$$

Et donc $f(u_1,...,s,...,u_n) \geq_\varphi f(u_1,...,t,...,u_n)$.

Lemme I.3.4 Si $\geq_D$ est bien fondé alors $\geq_\varphi$ est bien fondé.

Preuve : immédiate.

Remarque I.3.4 La détermination d'une interprétation sémantique n'est pas toujours évidente et elle fait appel à l'intuition et l'expérience. Dans le cadre de l'automatisation de la preuve de terminaison et vu les choix infinis des domaines, on se limite souvent à une partie des entiers, i.e : $D = D_\mu = \{n \in Z \text{ tel que } n \geq \mu\}$ où $\geq$ est l'ordre usuel sur Z. Ces interprétations sont appelées : *interprétation arithmétiques.*

I.3.4.4 Interprétation polynomiale

Elles étaient introduites par Lankford [DAL 79], les termes sont interprétés par des fonctions polynomiales sur les entiers.

Définition I.3.12 Une *interprétation polynomiale* de termes de $T(\sum, V)$ est une interprétation arithmétique sur un domaine D_μ telle que pour tout $f \in \sum, \|f\|$ est une fonction polynomiale.

Remarque I.3.5

- On peut passer du domaine D_μ au domaine D_0 en effectuant la translation

  $f_0(x_1,...,x_n) = f_\mu(x_1 + \mu,...,x_n + \mu) - \mu$ et de définir un ordre $(\geq_{\phi 0}, \succ_{\phi 0})$ à partir de l'ordre $(\geq_\phi, \succ_\phi)$.

- Il est clair que l'ordre sémantique à base d'interprétation polynomiale est monotone si les coefficients des polynômes sont positifs.

Exemple I.3.7

- Soit le système

$$\begin{cases} x + 0 \to 0 \\ x + S(x) \to S(x + y). \end{cases}$$

Et l'interprétation polynomiale

$$\|0\| = 1,$$
$$\|S\|(x) = x + 1,$$
$$\|+\|(x, y) = x + 2y,$$

L'application Φ induite est :

$$\Phi(x) = x \qquad \Phi(x + 0) = x + 2$$
$$\Phi(x + S(y)) = x + 2y + 2 \qquad \Phi(S(x + y)) = x + 2y + 1$$

En remarque que l'image de la partie gauche est supérieur à celle de la partie droite pour chaque règle, ce qui nous permet de montrer la terminaison par cette interprétation.

Si on veut passer du domaine $D_1 = \{x \in Z | x \geq 1\}$ à $D_0 = N$ l'interprétation sera :

$$\|0\| = 0,$$
$$\|S\|(x) = x + 1,$$
$$\|+\|(x, y) = x + 2y + 2,$$

- Soit le système $\Re$ définit par les règles :

$$\begin{cases} f(x, y) \to x; \\ g(a) \to h(a, b, c); \\ i(x) \to f(x, x); \\ h(x, x, y) \to g(x); \end{cases}$$

L'interprétation suivante assure la terminaison :

$\|a\| = 8;$
$\|b\| = 2;$
$\|g\|(x) = x.x;$
$\|i\|(x) = 4.x;$
$\|f\|(x, y) = x + y;$
$\|h\|(x, y, z) = 2.x.y + z$

I.3.5 Méthodes transformationnelles

Avec ces méthodes, on transforme le système $\Re$ dont on veut prouver la terminaison, en un système $\Re'$, qui à la propriété d'être plus facile à étudier telles que si $\Re'$ est fortement normalisant alors $\Re$ est fortement normalisant.

Une des classes les plus utilisées de ces méthodes transformationnelles est *l'étiquetage sémantique* proposé par Hans Zantema [ZAN 95]. Il s'agit ici de donner à chaque symbole de fonction une étiquette qui dépend de la sémantique de ses arguments.

Exemple I.3.8

Le programme fonctionnel calculant la fonction factorielle peut être décrit par le système de réécriture suivant :

$$Fact(S(x)) \to Fact(P(S(x))) * S(x)$$
$$P(S(0)) \to 0$$
$$P(S(S(x))) \to S(P(S(x)))$$

La terminaison de ce programme est facile à vérifier, il est claire qu'à chaque appel récursif de fact l'argument décroît strictement, mais si on oubli la sémantique des termes qui représentent les nombres entiers la terminaison ne devient plus évidente. Car la partie gauche de la première règle peut être incluse dans la partie droite et donc la terminaison ne peut être prouvé par les

techniques standards tel que le RPO. La terminaison du RS étiqueté obtenu à partir de l'original en remplaçant la première règle par une infinité de règles étiquetées : $Fact_{i+1}(S(x)) \to Fact_i(P(S(x)))*S(x)$ pour chaque entier naturel est facile à prouver avec RPO.

I.3.5.1 Transformation des contraintes d'ordre

Plutôt que prouver la terminaison d'un nouveau système de réécriture, il est possible de transformer directement les contraintes d'ordre issues du système original. Ces approches permettent, par exemple par élimination des arguments inopportuns, de construire un ordre de réduction faible à partir d'un ordre de réduction donné. Elles sont en particulier tout à fait adaptées aux critères par paires de dépendance où les ordres requis ne sont pas strictement monotones.

On appelle schéma de programme récursif (RPS) un système de réécriture tel que :

- Chaque symbole défini n'est à la racine que d'une seule règle ;
- Toutes les règles sont de la forme $f(x_1,...,x_n) \to r$ où les x_i sont des variables deux à deux disjointes et r est un terme arbitraire.

La terminaison des RPS est un problème décidable : il suffit qu'il n'y ait pas de règles récursives ou mutuellement récursives.

Un système à filtrage d'argument (abrégé en AFS, de l'anglais Argument Filtering System) est un RPS tel que pour chaque règle $f(x_1,...,x_n) \to r$, le membre droit r est ou bien :

- Une variable (un des x_i) ou bien
- Un terme $N_f(y_1,...,y_n)$ où N_f est un nouveau symbole n'apparaissant que dans cette règle et où l'ensemble des y_i est inclus dans celui des x_i.

Tout AFS est fortement normalisant et confluent (car orthogonal).

Théorème I.3.6 Soient P un RPS et $\geq$ un pré ordre de réduction faible. La relation $\geq$ définie par $u \geq v$ si et seulement si $u \downarrow p \geq v \downarrow p$ est un pré ordre de réduction faible.

On a ainsi gagné un moyen de construire des prés ordres de réduction faibles

Exemple I.3.9 (Ferriera & Zantema) [DAL 79]

Considérons le système :

$$\begin{cases} f(g(x)) \to g(f(f(x))) \\ f(h(x)) \to h(g(x)) \end{cases}$$

On en extrait deux paires critiques :

$$\left\langle \hat{f}(g(x)), \hat{f}(x) \right\rangle,$$
$$\left\langle \hat{f}(g(x)), \hat{f}(f(x)) \right\rangle.$$

En appliquant le RPS: $\{f(x) \to x; h(x) \to 0\}$, les contraintes deviennent :

$$\left\{ \begin{array}{l} g(x) \geq g(x), \\ 0 \geq 0; \\ \hat{f}(g(x)) \succ \hat{f}(x) \end{array} \right\}$$

Il est alors possible de conclure grâce à l'ordre RPO en prenant les symboles égaux dans la précédence.

Chapitre II

Calculs

Et

Substitutions explicites

Dans ce chapitre, on commence par introduire le λ-calcul en présentant sa définition, et ses propriétés. Nous abordons ensuite les λ-calculs avec substitutions explicites : nous en donnons le principe fondateur, puis les principales propriétés, et en fin nous en faisons un tour d'horizon des différents calculs qui ont été définis.

II.1. Le λ-calcul

Le λ-calcul[3] a été inventé vers 1930 par le logicien américain Alonzo CHURCH dans le but de formaliser le calcul mathématique, fondé sur la notion de fonction, c'est à dire où l'on établit pas de distinction entre objets, fonction, opérateurs etc... En effet, lorsqu'en mathématiques on écrit $f : x \mapsto f(x)$, f est définie en extension et cela est le seul moyen pour définir la fonction f, on ne peut pas écrire $f = E$ où E est une expression remplaçant f. En λ-calcul on peut écrire une telle expression : $f = \lambda x.(f(x))$.

II.1.1 Définition des λ-Termes

Supposons avoir un ensemble infini de variables V. Les termes du λ-calcul (λ-termes) sont construits à l'aide de la grammaire suivante ou x est une variable quelconque de V :

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

Un terme est soit une variable x, soit une fonction comportant une variable (le paramètre formel) et un sous terme (le corps de la fonction) $\lambda x.t$, soit l'application d'un, sous terme (la fonction, à gauche) à un autre sous terme (le paramètre effectif, à droite) $t\ t$. L'ensemble des λ-termes est noté Λ.

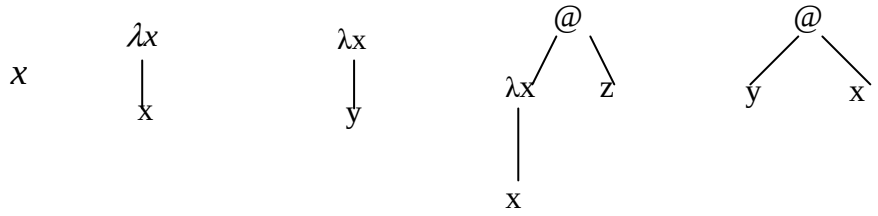
[3] La référence de base est [**BAR 84**] et [**KRI 90**]

Exemple de II.1.1

Voici quelques exemples de λ-termes :

- x : toute variable est un terme.
- $\lambda x.x$: c'est la fonction identité ; ce qu'elle prend en paramètre, elle le renvoie comme résultat.
- $\lambda x.y$: cette fonction n'utilise pas son paramètre.
- $(\lambda x.x)z$: c'est l'application de la fonction identité à la variable z .
- $y\ \ x$: c'est l'application de y à x .

Les λ-termes peuvent être représentés par des arbres dont les nœuds sont soit des λx soit des application (notées @) et dont les feuilles sont des variables. Voici de gauche à droite les arbres correspondants aux termes présentés ci-dessus.



Dans le λ-calcul, on dit que les fonctions sont des " sont des citoyens de première classe " ; cela signifie que les fonctions sont des valeurs qu'on peut, par exemple, passer comme paramètre à d'autres fonctions. Cette possibilité est beaucoup plus utilisée en informatique qu'en mathématique. Par exemple, le λ-terme $(\lambda x.x)(\lambda y.z)$ représente l'application de la fonction $(\lambda x.x)$ avec comme paramètre la fonction $(\lambda y.z)$.

Une fois les termes construits, il est nécessaire de pouvoir faire des calculs sur les termes : en effet il est insuffisant de faire une application purement syntaxique d'une fonction à un argument - i.e. écrire $f(x)$ -, il faut passer cet argument à la fonction pour ne plus avoir un terme représentant la fonction f appliquée à x mais un terme représentant la valeur de $f(x)$. L'opération qui effectue cela est la β -réduction donnée par :

$$(\lambda x.M)N \xrightarrow{\beta} M[x \leftarrow N]$$

M et N sont appelés des méta variables de terme, et x est une méta variables. Le membre gauche de la règle correspondre à l'application d'une fonction à un argument. Le filtrage permet d'appliquer cette règle à tout les terme dont l'arbre commence par un nœud @ et se poursuit à gauche par un λ. Le membre droit est un peu plus complexe, nous allons l'étudier de plus prés.

La notation $M[x \leftarrow N]$ représente un calcul implicite : celui de remplacer dans M toutes les occurrences libres de x par le terme N, ce calcul implicite est effectué instantanément avant de poursuivre la réécriture est appelé substitution implicite et qui peut être décomposé en deux parties : premièrement, il faut propager la substitution jusqu'aux variables, et deuxièmement, il faut effectuer la substitution des variables concernées.

II.1.2 Substitution implicite, problèmes de variables et α-équivalence

Comme une méta opération (c à d opération implicite), la substitution peut être définie de la façon suivante, par induction sur les termes :

$x[x \leftarrow M] = M$

$y[x \leftarrow M] = y$ Si $x \neq y$

$(MN)[x \leftarrow V] = (M[x \leftarrow V]N[x \leftarrow V])$

$(\lambda x.M)[x \leftarrow N]$ Possibilité d'avoir des problèmes

$(\lambda y.M)[x \leftarrow N]$ Possibilité d'avoir des problèmes

Ces deux dernières lignes nécessitent plus d'attention, comme le montrent les deux exemples suivants :

- Dans le terme $(\lambda x.x)[x \leftarrow y]$ la variable x de la substitution correspond au paramètre formel d'une fonction qui n'apparaît plus dans le terme ; elle n'a rien avoir avec la variable x de la fonction $(\lambda x.x)$. La fonction $(\lambda x.x)$ ne doit donc pas être modifiée par la substitution.
- $(\lambda x.(\lambda x.x))y$ est un terme qui peut conduire à $(\lambda x.x)[x \leftarrow y]$. Cela signifie qu'on a effectué la réduction suivante : $(\lambda x.(\lambda x.x))y \rightarrow (\lambda x.x)[x \leftarrow y]$

Un calcul naïf de la substitution, mène à écrire : $(\lambda x.x)[x \leftarrow y] = \lambda x.([x \leftarrow y]) = \lambda x.y$ et donne un résultat complètement erroné.

Le problème vient du fait que, dans le terme d'origine, $(\lambda x.(\lambda x.x))y$ on a employé deux fois la variable x comme paramètre formel.

- $(\lambda y.x)[x \leftarrow y]$, ici la variable y du terme $(\lambda y.x)$ est le paramètre formel de cette fonction et n'a donc de sens que dans le corps de celle-ci. Par contre, la variable y de la substitution est extérieure, et donc indépendante.

Par exemple le terme $(\lambda x.(\lambda y.x))y$ peut nous mener à la substitution précédente et donc la réduction est $(\lambda x.(\lambda y.x))y \rightarrow (\lambda y.x)[x \leftarrow y]$, si on continue ainsi on obtiendra le terme $(\lambda y.y)$ dans

lequel la variable y correspond maintenant au paramètre formel de la fonction, ce qui serait une erreur. Le problème vient du fait que, dans le terme d'origine $(\lambda x.(\lambda y.x))y$ on a employé la variable y à la fois comme une simple variable et comme le paramètre d'une fonction.

Pour résoudre ces problèmes, nous devons étudier de plus près les différents types de variables qui se trouvent dans les λ-termes. Une notion fondamentale, autant en informatique qu'en mathématique, est celle de variables liées. Lorsqu'on écrit la définition d'une fonction en mathématiques, on choisit une variable pour être le paramètre formel comme par exemple, x dans $f(x) = x + 1$, on aurait tout aussi bien pu choisir y ou z, ce qui nous aurait donné :

$$f(y) = y + 1 \qquad \text{ou} \qquad f(z) = z + 1$$

Dans tous les cas la fonction f reste la même et son paramètre est une variable liée ne servant qu'à sa définition. Dans le λ-calcul, les variables liées sont celle introduites entre les symboles (« λ » et «. »). La particularité des variables liées réside dans le fait qu'on peut les renommer sans modifier le sens de la fonction. Regardons quelques exemples de termes qui sont équivalents:

Exemple II.1.2

- $\lambda x.x$ et $\lambda y.y$
- $(\lambda x.\lambda y.(xy))$ et $(\lambda v.\lambda u.(vu))$.

La notion symétrique de variable liée est celle de variable libre. Dans un terme, une variable est libre si elle n'est pas dans la portée d'un lieur pour cette variable. Dans une représentation arborescente, une variable x sera libre si il n'y a pas, au dessus d'elle, un λx dont elle sera une feuille (même lointaine).

Exemple II 1.3

- dans le terme $\lambda x.(xy)$, x est liée et y est libre.
- dans le terme $\lambda x.\lambda y.(yx)$ x et y sont liées.
- dans le terme $\lambda x.(yz)$ y et z sont libres.

Remarques II 1.1

- En renommant une ou plusieurs variables dans un terme, on doit veiller à ce que le nouveau terme obtenu après renommage et le terme original gardent le même sens.

- Pour éviter tout conflit il est préférable de choisir des variables qui n'existent pas dans le terme d'origine (variables fraîches) comme dans le cas du système $\lambda@$ [GMMS 04], qui sera abordé dans le prochain chapitre.

II.1.3 α-conversion, α-équivalence

On dit que deux termes M et M' sont α-équivalents s'ils ne se diffèrent que par un renom mage de leurs variables liées (on le note $M \approx_\alpha M'$). L'α-conversion d'un λ-terme est le remplacement de tous ces sous- termes de la forme $\lambda x.M$ par $\lambda y.M'$, ou y est une variable fraîche et M' est le terme M dans lequel on a remplacé toutes les variables x (liées par ce λ) par y . Formellement l'α-équivalence est définie inductive ment par :

- $x \approx_\alpha x$
- $MN \approx_\alpha M'N'$ avec $M \approx_\alpha M'$ et $N \approx_\alpha N'$
- $\lambda x.M \approx_\alpha \lambda y.(M'[x \leftarrow y])$ où y est une variable fraîche et $M \approx M'$

Le changement de l'état d'une variable (libre, liée) entre le terme d'origine et celui après substitution est appelé une capture de variable. Ce problème est résolu par l'α-conversion, ce qui nous donne une formalisation plus complète de la substitution.

II.1.4 Substitution implicite

La substitution implicite est définie, modulo α-équivalence, de la façon suivante, par induction sur la structure des termes :

- $x[x \leftarrow M] = M$
- $y[x \leftarrow M] = y$ si $x \neq y$
- $(MN)[x \leftarrow V] = (M[x \leftarrow V]N[x \leftarrow V])$
- $(\lambda y.M)[x \leftarrow V] = \lambda y.(M[x \leftarrow V])$ avec $x \neq y$ et y non libre dans V

Exemple II.1.5 Voici quelques exemples de substitutions (on fait apparaître quelques étapes du calcul) :

- $(xy)[y \leftarrow z] = (x[y \leftarrow z]y[y \leftarrow z]) = (xz)$
- $(\lambda x.y)[y \leftarrow x] = \lambda t.(y[y \leftarrow z]) = \lambda t.x$ (renommage de x)
- $(\lambda x.(xy))[y \leftarrow x] = (\lambda z.(zy))[y \leftarrow x] = \lambda z.((zy)[y \leftarrow x]) = \lambda z.(zx)$

II.1.10 Ensemble des variables libres d'un terme

L'ensemble des variables libres d'un terme t, noté $FV(t)$ est défini par induction sur la structure du terme de la façon suivante :

- si le terme est une variable x , alors $FV(x)=\{x\}$,
- si le terme est une application (MN), $FV(MN)=FV(M)\bigcup FV(N)$
- si le terme est une fonction $(\lambda x.M)$, $FV(\lambda x.M) = FV(M) \setminus \{x\}$

Exemple II.1.6

- $FV(xy)=\{x,y\}$
- $FV(\lambda x.x)=\{x\} \setminus \{x\}=\{\ \}$
- $FV(\lambda x.\lambda y.(xy))=\{x,y\} \setminus \{x,y\}=\{\ \}$
- $FV((\lambda x.(xy))(xz))=(\{x,y\}\setminus\{x\})\bigcup\{x,z\}=\{x,y,z\}$

II.2 Les indices de de Bruijn

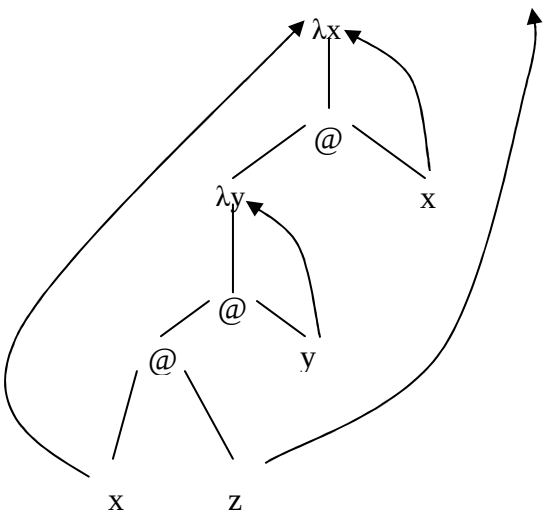
II.2.1 Présentation et définition

Les difficultés liées à la capture de variables et le temps coûteux nécessaire à l'implantation de l'opération de l'α-conversion on conduit à une notation des λ-termes, sans utilisation de noms de variables, c'est la notation de de Bruijn [dB 72] et [dB 78].

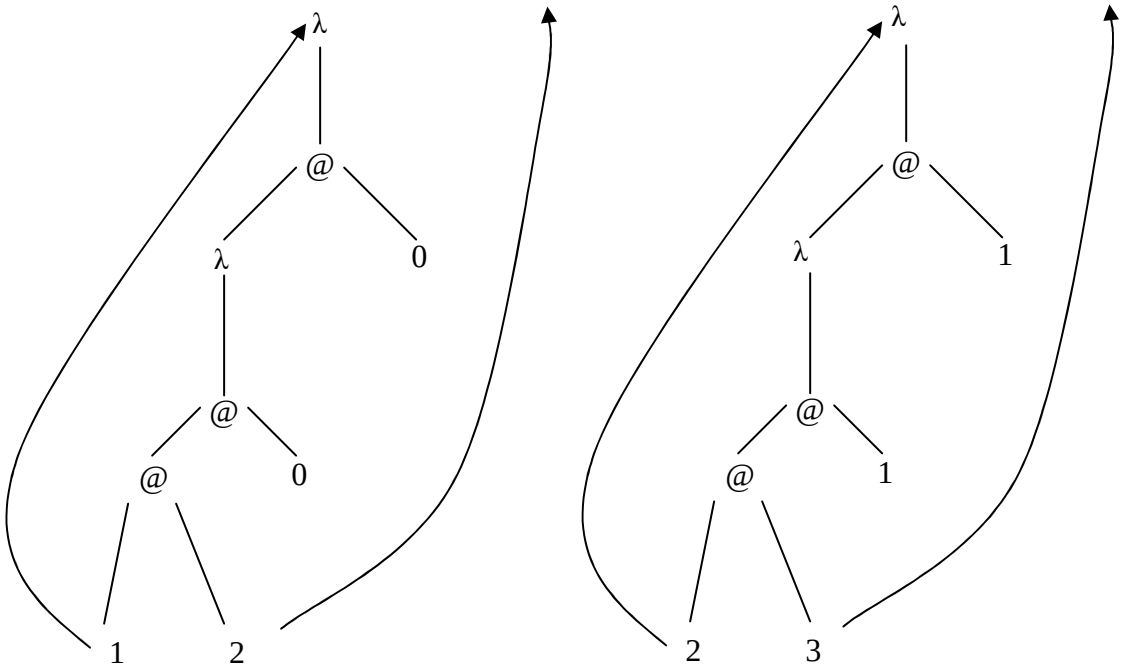
L'idée est que toute variable liée peut-être codée

- par sa hauteur de liaison (le nombre de λ que l'on rencontre avant d'atteindre le λ lieur et dans ce cas la numérotation commence par 0)
- ou en indiquant qu'il s'agit du $n-ième$ lieur au dessus en remontant toujours l'arbre du terme.

Exemple II.1.7 Soit le λ-terme: $M=\lambda x.((\lambda y.((xz)y))x)$, qui est représenté par l'arbre suivant



Donc le terme M est soit $\lambda((\lambda((12)0))0)$ ou $\lambda((\lambda((23)1))1)$. Les voici sous forme d'arbres.



Dans l'exemple on voit que chaque flèche relie une variable à son lieur (sauf z qui n'a pas de lieur). Dans les deux cas, on numérote les variables libres en utilisant des nombres suffisamment grands pour qu'ils ne référencent aucun lieur. Les indices de de Bruijn résolvent le problème des variables nommées, mais, en contrepartie, on obtient un système dans lequel les termes sont difficilement lisibles.

Définition II.2.1

Les terme du λ-calcul avec indice de de Bruijn sont construits à l'aide de la grammaire suivante ou n est un entier naturel :

$a ::= n \mid aa \mid \lambda a$

La règle de β – réduction, dans laquelle t et u sont des métas variables de terme :

$$(\lambda a)v \to a[1 \leftarrow v]$$

II.2.2 Substitution avec indice de De Bruijn

Les problèmes qui se posent pour la substitution implicite de calcul nommé persistent encore dans les calculs avec indice et donc il faut être attentive à la capture de variable lorsqu'on traite une substitution qui tente de remplacer une variable liée.

Exemple II.2.1

Soit le terme $M = (\lambda x.\lambda y.(xy))z = (\lambda\lambda(12))1$, dans le calcul nommé, M se réduit à $(\lambda y.(xy))[x \leftarrow z]$ c'est à dire $(\lambda y.(zy))$. Dans le calcul avec indice $(\lambda\lambda(12))1 \rightarrow (\lambda\lambda(12))[1 \leftarrow 1]$, et ce n'est pas le 1 qui doit être remplacée, mais plutôt le 2, lorsque la substitution va entrer dans le corps de la fonction l'indice à substituer ne sera plus 1, mais 2. En regardant attentivement, on constate que c'est normal, puisque entre la substitution et sa variable, il y a alors un λ de moins. Par contre, après être passées sous le λ, les variables de corps de la substitution (ce qui est à droite de la flèche) ont un λ de plus à traverser pour atteindre leur lieur. C'est le problème de la capture de variable qui apparaît ici. Cependant, l'indice à remplacer a comme valeur le nombre de lieurs traversés depuis la création de la substitution, il suffira donc d'ajouter ce nombre à toutes les variables libres du corps de la substitution. Dans la définition qui suit on donne formellement ce qu'elle est la substitution implicite du λ-calcul avec indice de de Bruijn.

Définition II.2.2

La substitution implicite est définie de la façon suivant, par induction sur la structure des termes :

$$n\{n \leftarrow u\} = \bigcup_0^n (u)$$

$$m\{n \leftarrow u\} = m \quad \text{si } m < n$$

$$m\{n \leftarrow u\} = m - 1 \text{ si } m > n$$

(tu) $\{n \leftarrow v\} = (t\{n \leftarrow v\}u\{n \leftarrow v\})$

(λt) $\{n \leftarrow u\} = \lambda(t\{n+1 \leftarrow u\})$

La fonction $\bigcup_i^n (t)$ effectue la mise à jour des indices pour éviter la capture des variables.

Voici sa définition par induction structurelle :

$$\bigcup_i^n (tu) = \bigcup_i^n (t) \bigcup_i^n (u)$$

$$\bigcup_i^n (\lambda t) = \lambda \bigcup_{i+1}^n (t)$$

$$\bigcup_i^n (m) = m \quad \text{si } m \le i$$

$$\bigcup_i^n (m) = m + n - 1 \text{ si } m > i$$

II.3 Propriétés du λ-calcul

Les propriétés les plus recherchées sont : la terminaison et la confluence. Dans ce qui suit ces propriétés sont valables aussi bien pour la version nommée que pour les version avec indice de de Bruijn.

A) Terminaison :

Le λ-calcul n'est pas fortement normalisant et voici le contre-exemple le plus connu. On appelle ω le terme $(\lambda x.xx)$ et Ω le terme $\omega\omega = (\lambda x.(xx))(\lambda x.(xx))$, sa réduction donne $(xx)[x \leftarrow (\lambda x.(xx))] = \Omega$. Après une étape de réduction, on obtient à nouveau le même terme, on peut alors recommencer à l'infini :

$$\Omega \to \Omega \to \Omega \to \Omega \to \Omega \to ...$$

B) Confluence

Théorème II.3.1

Le λ-calcul est confluent.

Preuve : voir [BAR 81]. Pour la version avec indices de de Bruijn, on trouvera une preuve dans [KR 98] qui utilise des traductions entre les termes avec indices et les termes avec noms.

Lemme II.3.1 (lemme de substitution)

Soit t, u et v des λ-termes tels que la variable x n'est pas libre dans v $(x \notin FV(v))$ On a

$$t[x \leftarrow u][y \leftarrow v] = t[y \leftarrow v][x \leftarrow u[y \leftarrow v]]$$

Preuve : par induction sur la structure des termes [HIN 80] :

II.4 les λ-calculs avec substitution explicites

La substitution de variables telle qu'on l'a vu est une opération implicite qui fait appel à un algorithme extérieur non compté dans les étapes de calcul. Si cette substitution nous parait théoriquement suffisante, elle est en revanche insatisfaisante d'un point de vue opérationnel ou calculatoire car elle ne fournit aucun moyen pour observer les propriétés opérationnelles de l'exécution des algorithmes implantant la $\beta-$réduction tels que la consommation de temps ou d'espace. Si on veut par exemple calculer la forme normale du terme $(\lambda x.M)N$, l'opération $M[x \leftarrow N]$ est vue comme une seule étape de calcul nous donnant illusoirement un temps de calcul négligeable. Mais, si on suppose que la variable x a un nombre d'occurrence très important dans le terme M et que le terme N ait une très grande taille. La substitution pourrait nécessiter un temps de calcul considérable. En rendant l'opération de substitution interne aux systèmes de réécriture des λ-calculs, on peut combler ce manque de précision.

Les calculs avec substitutions explicites rendent, comme leur nom l'indique, l'opération de substitution interne au calcul. Ils étendent la grammaire des termes afin d'y intégrer la substitution, et ils ajoutent de nouvelles règles de réduction pour gérer ces substitutions essentiellement leur propagation jusqu'aux variables. Cela fait, on dispose d'un outil plus fin pour analyser la réduction des termes : on peut choisir quelle substitution propager et à quelle moment le faire cependant, dès l'arrivée des premiers calculs avec substitutions explicites, il est apparu que certaines propriétés (terminaison et confluence) étaient difficiles à établir, voir perdues dans ces nouveaux systèmes. Lorsqu'un calcul avec substitutions explicites est défini, on appelle terme pur un terme sans substitution.

II.5 Panorama des calculs avec substitutions explicites

II.5.1 Le λv-calcul

Ce calcul [BBLR 96], est un λ-calcul qui utilise les indices de de Bruijn, A cause des opérations sur les indices de de Bruijn, sa définition est bien plus complexe que celle du λx. La figure Fig II.1 donne les termes et les règles de réécritures du λv-calcul. On remarque que les substitutions ne comportent pas l'indication directe sur la variable qu'elles doivent substituer. Cette information est cachée dans les $\Uparrow$ qui sont ajoutées à chaque traversée d'un λ. La mise à jour des indices après passage de la substitution est assurée par l'opération $\uparrow$, comme le montre bien la règle Varshift. Ce calcul est fortement normalisant, confluent sur les termes clos et ses formes

normales sont des termes purs ; le λv-calcul vérifie la simulation de la β-réduction, la confluence, et la préservation de la normalisation forte (voir [BBLR 96]).

Les termes du λv:

$$t ::= n \mid (tt) \mid \lambda t \mid t[s]$$
$$s ::= a/ \mid \Uparrow(s) \mid \uparrow$$

Les règles de réduction :

$$(\lambda t)u \to_{B} t[u/]$$
$$(tu)[s] \to_{App} (t[s])(u[s])$$
$$(\lambda t)[s] \to_{Lamda} \lambda(t[\Uparrow(s)])$$
$$1[t/] \to_{F\,\mathrm{var}} t$$
$$n+1[t/] \to_{R\,\mathrm{var}} n$$
$$1[\Uparrow(s)] \to_{F\,\mathrm{var}\,lift} 1$$
$$n+1[\Uparrow(s)] \to_{R\,\mathrm{var}\,lift} n[s][\uparrow]$$
$$n[\uparrow] \to_{Farshift} n+1$$

Fig.II.1 Définition du λv-calcul

II.5.2 Le $\lambda\sigma$-calcul

L'intérêt du $\lambda\sigma$-calcul est qu'il remplace la notion complexe de substitution $M[n \leftarrow N]$ d'indices de de Bruijn par une série de règles de réécriture sur des termes de premier ordre, comme pour les combinateurs S, K et I [HS 86]. Mais contrairement à la logique combinatoire, le $\lambda\sigma$-calcul interprète correctement la β-réduction ce qui permet de réaliser de véritables machines à réduction des λ-termes. Historiquement $\lambda\sigma$-calcul considéré comme le premier calcul avec substitutions explicites [ACCL 91], [CUR 86] et [HAR 89]. Il propose de gérer les substitutions explicites sous de forme de listes plutôt que par unités séparées. Comme λv, ce calcul est basé sur les indices de de Bruijn, mais il offre une plus grande dynamique car il possède des règles pour gérer la composition des substitutions (règle Clos et Map). Cependant, il n'est pas confluent. Une version légèrement modifiée de ce calcul, le $\lambda\sigma_{\Uparrow}$-calcul [CHL 92], a été proposée peu de temps après pour combler ce manque. Ce calcul simule la β-réduction mais ne vérifie pas la PSN [MEL 95].

Les termes du $\lambda\sigma$-calcul sont donnés par:

$$t ::= 1 \mid (tt) \mid \lambda t \mid t[s]$$

$$s ::= id \mid \uparrow \mid t.s \mid s \circ s$$

Et les règles sont :

$$(\lambda t)u \rightarrow_{\beta} t[u.id]$$

$$(tu)[s] \rightarrow_{App} (t[s])(u[s])$$

$$(\lambda t)[s] \rightarrow_{Lambda} \lambda(t[1.(s \circ \uparrow)])$$

$$1[id] \rightarrow_{Varid} 1$$

$$1[t.s] \rightarrow_{Varcons} t$$

$$t[s][s'] \rightarrow_{Clos} t[s \circ s']$$

$$id \circ s \rightarrow_{IDL} s$$

$$\uparrow \circ id \rightarrow_{ShiftId} \uparrow$$

$$\uparrow \circ (t.s) \rightarrow_{Shiftcons} s$$

$$(t.s) \circ s' \rightarrow_{Map} t[s'].(s \circ s')$$

$$(s_1 \circ s_2) \circ s_3 \rightarrow_{Ass} s_1 \circ (s_2 \circ s_3)$$

FIG.II.2 Définition du $\lambda\sigma$-calcul

II.5.3 Le $\lambda\sigma_n$-calcul

Le $\lambda\sigma_n$-calcul [ACCL 91] est une version nommée du $\lambda\sigma$-calcul. Les deux calculs possèdent les mêmes propriétés. Ces règles de réécritures sont données par la figure II.3.

Voici le programme des termes :

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

$$s ::= id \mid (t/x).s \mid s \circ s$$

Voici les règles de réduction :

$$(\lambda x.t)u \rightarrow_{\beta} t[(u/x).id]$$

$$(tu)[s] \rightarrow_{App} (t[s])(u[s])$$

$$(\lambda x.t)[s] \rightarrow_{Lambda} \lambda y.(t[(y/x).s]) \text{ avec y variable fraîche}$$

$$x[id] \rightarrow_{VarId} x$$

$$x[(t/x).s] \rightarrow_{\text{VarCons1}} t$$

$$x[(t/y).s] \rightarrow_{\text{VarCons2}} x[s] \qquad (x \neq y)$$

$$t[s][s'] \rightarrow_{Clos} t[s \circ s']$$

$$id \circ s \rightarrow_{IdI.} s$$

$$((t/x).s) \circ s' \rightarrow_{Map} (t[s']/x).(s \circ s')$$

$$(s_1 \circ s_2) \circ s_3 \rightarrow_{Ass} s_1 \circ (s_2 \circ s_3)$$

FIG.II.3 Définition du $\lambda\sigma_n$-calcul

II.5.4 Le $\lambda\sigma_{ws}$-calcul

Ce calcul [DG 99] et [DG 01] qui utilise la notation de de Bruijn est le premier calcul a possédé toutes les propriétés recherchées (préserve la propriété de forte normalisant, confluant, simule la β-réduction). La figure II.4 montre la définition du $\lambda\sigma_{ws}$-calcul

Les termes sont définis par :

$$t ::= n \mid (tt) \mid \lambda t \mid [n/t, n]t \mid \langle n \rangle t$$

Les règles de réduction :

b_1	$(\lambda t)u \rightarrow [0/u,0]t$	
b_2	$(\langle k \rangle \lambda t)u \rightarrow [0/u, k]t$	
f	$[i/u, j]\lambda t \rightarrow \lambda[i+1/u, j]t$	
a	$[i/u, j](tv) \rightarrow (([i/u, j]t)([i/u, j]v))$	
e_1	$[i/u, j]\langle k \rangle t \rightarrow \langle j + k - 1 \rangle t$	si $i < k$
e_2	$[i/u, j]\langle k \rangle t \rightarrow \langle k \rangle [i - k/u, j]t$	si $i \geq k$
n_1	$[i/u, j]k \rightarrow k$	si $i \succ k$
n_2	$[i/u, j]i \rightarrow \langle i \rangle u$	
n_3	$[i/u, j]k \rightarrow j + k - 1$	si $i < k$
c_1	$[i/u, j][k/v, l]t \rightarrow [k/[i - k/u, j]v, j + l - 1]t$	si $k \leq i < k + l$
c_2	$[i/u, j][k/v, l]t \rightarrow [k/[i - k/u, j]v, l][i - l + 1/u, j]t$	si $i \geq k + 1$
d	$\langle i \rangle \langle l \rangle t \rightarrow \langle i + j \rangle t$	

FIG.II.4 Définition du $\lambda\sigma_{ws}$-calcul

De nombreux calcul de substitutions explicites ont été définis, Il serait trop long de les présenter tous. Voici, une liste non exhaustive de calculs qu'on peut trouver dans la littérature spécialisée : le plus ancien calcul qui introduit une notion d'explicitation da la substitution est le $C\lambda\zeta\Phi$ [dB 78]; le calcul des combinateurs catégoriques [CUR 86] qui a donné naissance au $\lambda\sigma$-calcul ; d'autres calculs sont : λ_d et λ_{dn} [FKP 99], λs_e [KR 97], etc.

Dans le prochain chapitre nous allons étudier un calcul qui n'utilise ni substitution ni renom mage de variable à savoir le système λ@ [GMMS 02].

Chapitre III

Etude du système De Substitution explicite λ@

Nous allons dans ce chapitre étudier le λ@-calcul proposé dans [GMMS 02]. Ce calcul est un formalisme très simple qui n'utilise aucun opérateur de substitution et qui fait un renom- mage explicite des variables en introduisant la α-conversion comme une règle dans un système de réécriture de termes d'ordre supérieur. Nous montrons que la β-réduction classique est simulée par les règles de réécriture de ce système. Nous montrons aussi que ce système préserve les propriétés fondamentales du λ-calcul classique à savoir la confluence qui garantit l'unicité du résultat de calcul et la normalisation forte qui assure la terminaison des calculs.

III.1. Introduction :

Le λ-calcul classique gère la substitution d'une manière implicite. Les λ-calculs avec substitution explicite étudiés par plusieurs auteurs (Voir [ACCL 95], [BBLR 96], [BLO 97],…). La plupart de ces calculs utilisent la notation de de bruijn. Certains calculs comme le λ-Exp [BLO 97] utilise les variables nommées. Le λ-Exp par exemple est augmenté par de nouveaux opérateurs qui rendent son interprétation difficile. Le λ@ est une extension très simple du λ-calcul, il résout avec élégance le problème de capture de variables en introduisant quand il est nécessaire et systématiquement une nouvelle variable fraîche (notées $g_1, g_2 ...$) pour renommer les variables liées. Dans ce qui suit nous allons étudier ces principales propriétés.

III.2. Présentation du λ@-calcul :

III.2.1 Définition des termes

Les termes du λ@-calcul sont une extension des termes du λ-calcul classique (représentés par les symboles M, N, O, P, Q...) et il sont définis par la grammaire suivante :

$M ::= x \mid \lambda x.M \mid MN \mid @(\lambda x.M; N)$

Les symboles $xyzvuwt$ représentent un ensemble infini de variables. Dans le nouveau constructeur $@(\lambda x.M; N)$, on exige que le terme N soit libre pour x dans le terme M c'est à dire qu'aucune occurrence libre de x dans M ne se trouve dans la portée d'un (λy) où y est une variable libre dans le terme N. Les conventions du λ-calcul classique sur l'utilisation des parenthèses pour indiquer les structures des sous termes restent valables dans le λ@, par exemple le terme $\lambda x.\lambda y.M$ désigne $\lambda x.(\lambda y.M)$ et MNP représente le terme $(MN)P$ et le terme $@(\lambda x.M; N)$ est muni de la plus grande précédence donc $\lambda x.MN\ @(\lambda y.P; Q)$ dénote $\lambda x.MN(@(\lambda y.P; Q))$ (ainsi toutes les variables x dans les sous termes de M,N,P et Q sont liées par le premier λ).

L'ensemble des λ@ termes inclus un sous ensemble des termes ordinaires du λ-calcul noté Λ, nous disons qu'un terme est pur s'il est un élément de l'ensemble Λ, c'est à dire s'il ne contient pas un sous terme de la forme $@(\lambda x.M; N)$. La β-réduction classique est dénoté par $\rightarrow^{\beta}$ et elle n'opère que sur les termes purs bien sur.

III.2.2 Variables libres et liées.

L'ensemble des variables libres d'un λ@ terme M dénoté par $fv(M)$ est défini inductive ment sur la structure de M par :

$$
\begin{aligned}
&fv(x) = \{x\}\\
&fv(\lambda x.M) = fv(M) - \{x\}\\
&fv(MN) = fv(M) \cup fv(N)\\
&fv(@(\lambda x.M; N)) = (fv(M) - \{x\}) \cup fv(N)
\end{aligned}
$$

Un λ@ terme M est dit clos si est seulement si $fv(M) = \Phi$.

On dit que deux λ@ termes M, N sont α-équivalents (notation $M \equiv N$) s'il ne diffèrent que par les noms des variables liées.

Dans λ@-calcul les termes α-convertibles sont identifiés mais aucun changement de noms n'est nécessaire car les règles du système effectuent quand il est nécessaire un renom mage systématique des variables liées.

Le résultat de changement des noms de toutes les occurrences libres de la variable y dans le terme M sans capture de variables est exprimé par $M[y := z]$ (substitution classique).

III.2.3 règles de réécritures du système λ@.

Le système λ@ est défini par les règles de réécriture suivantes :

1. $@(\lambda x.x;N)\rightarrow N$
2. $@(\lambda x.y;N)\rightarrow y$ si $x\neq y$
3. $@(\lambda x.(\lambda y.M);N)\rightarrow @(\lambda g.@(\lambda x.@(\lambda y.M;g);N))$
4. $@(\lambda x.MN;P)\rightarrow @(\lambda x.M;P)@(\lambda x.N;P)$
5. $(\lambda x.M)N\rightarrow @(\lambda x.M;N)$
6. $@(\lambda x.P)\rightarrow \lambda x.P$

Dans se qui suit la relation $\rightarrow_{\lambda @}$ dénotera la fermeture transitive de la réduction par le système λ@.

III.2.4 Exemples de réductions par λ@.

a) Soit $M=(\lambda x.(\lambda y.y)x)z$ un λ@-terme, nous avons :

$$\begin{aligned}(\lambda x.(\lambda y.y)x)z &\rightarrow_1 (\lambda x.(@(\lambda y.y;x)))z\\ &\rightarrow_1 (\lambda x.x)z\\ &\rightarrow_1 @(\lambda x.x;z)\\ &\rightarrow z\end{aligned}$$

b) Soit $M=(\lambda x.\lambda y.xy)(ty)$ un λ@-terme, nous avons :

$$\begin{aligned}(\lambda x.\lambda y.xy)(ty) &\rightarrow_5 @(\lambda x.\lambda y.xy;ty)\\ &\rightarrow_3 @(\lambda g.@(\lambda x.@(\lambda y.xy;g);(ty)))\\ &\rightarrow_4 @(\lambda g.@(\lambda x.@(\lambda y.x;g)@(\lambda y.y;g);(ty)))\\ &\rightarrow_{1et2} @(\lambda g.@(\lambda x.xg;(ty)))\\ &\rightarrow_4 @(\lambda g.@(\lambda x.x;ty)@(\lambda x.g;ty))\\ \rightarrow_{1et2} &@(\lambda g.(ty)g)\\ &\rightarrow_6 \lambda g.(ty)g\end{aligned}$$

c) Soit $M\equiv(\lambda z.(\lambda t.x)(\lambda z.x)y)((\lambda z.x)y)$ (le contre exemple de Mellies pour la PSN[4])

[4] Preservation of strong normalisation

$$
\begin{aligned}
M \equiv (\lambda z.(\lambda t.x)(\lambda z.x)y)((\lambda z.x)y) &\to_5 @(\lambda z.(\lambda t.x)(\lambda z.x)y;(\lambda z.x)y) \\
&\to_5 @(\lambda z.@(\lambda t.x;\lambda z.x)y;(\lambda z.x)y) \\
&\to_{2et5} @(\lambda z.xy;@(\lambda z.x;y)) \\
&\to_2 @(\lambda z.xy;x) \\
&\to_4 @(\lambda z.x;x)@(\lambda z.y;x) \\
&\to_{2et2} xy
\end{aligned}
$$

III.3 Etude des propriétés du λ@-calcul.

Comme nous l'avons vu, le λ@-calcul est une extension du lambda calcul classique, mais nous souhaitons toujours que cette extension soit «conservative » c'est à dire que toutes les propriétés vérifiées dans le lambda calcul restent valides dans le cas du nouveau calcul. Les propriétés les plus recherchées sont la confluence et la normalisation forte (unicité du résultat de calcul et sa terminaison dans le cas des termes qui sont fortement normalisant).

Dans la suite du travail nous utiliserons les conventions suivantes :

1. si $\to$ est une relation de réduction sur les termes alors on écrit $\to$ * pour désigner sa fermeture transitive.
2. la relation de réduction λ@-calcul est l'union des deux sous relations :

$(@_\beta)$ $(\lambda x.M)N \to @(\lambda x.M; N)$ (Introduction de @)

$(\to_{AR})$ Le reste des règles du système c'est à dire :

$$
\begin{aligned}
&@(\lambda x.x; N) \to N \\
&@(\lambda x.y; N) \to y \quad \text{si } x \neq y \\
&@(\lambda x.(\lambda y.M); N) \to @(\ \lambda g.@(\lambda x.@(\lambda y.M; g); N)\ ) \\
&@(\lambda x.MN; P) \to @(\lambda x.M; P)@(\lambda x.N; P) \\
&@(\lambda x.P) \to \lambda x.P
\end{aligned}
$$

La règle $(@_\beta)$ permet d'introduire le constructeur @ à chaque β-redex rencontré et puis c'est le sous-système $(\to_{AR})$ qui permet simplifier le β-redex jusqu'à ça forme normale si elle existe. Le $(\to_{AR})$ à les propriétés intéressante suivantes :

Lemme III.3.1.

$(\rightarrow_{AR})$ est fortement normalisant, confluent et a la propriété de la forme normale unique.

Preuve.

A. La normalisation forte peut être démontrée par l'interprétation p: $\lambda@\rightarrow \mathrm{N}$ (de l'ensemble des λ@ vers l'ensemble des entiers naturels) et qui décroît à chaque λ@ réduction.

$p(x)=1$
$P(MN)=p(M)+p(N)+1$
$p(\lambda x.M)\rightarrow p(M)+1$
$p(@(\lambda x.M;N))=p(M)(p(N)+1)$

B. On démontre par induction sur la structure de M que : si $M \rightarrow_{AR} N$ alors $p(M) > p(N)$

1. $M \equiv @(\lambda x.x; A)$ Alors $N \equiv A$; $p(M) = 1 * (p(A) + 1) > p(N) = p(A)$
2. $M \equiv @(\lambda x.y; A)$ alors $N \equiv y$; $p(M) = 1 * (1 + 1) > p(N) = p(y) = 1$
3. $M \equiv @(\lambda x.(\lambda y.A); B)$ alors $N \equiv @(\lambda g.@(\lambda x.@(\lambda y.A; g); B))$;

$p(M)=p(\lambda y.A)*(p(B)+1)>=(p(A)+1)*(p(b)+1)$ et $p(N)=1+p(C)*(1+p(B))$ ou $C \equiv_{\alpha} A$, il est claire que $p(M)>p(N)$.

B. Le $(\rightarrow_{AR})$ est orthogonal (linéaire à gauche et sans paire critique) alors il est confluent.

Lemme III.3.2.

Si $M, N \in \Lambda$, $@(\lambda xM; N) \rightarrow_{AR} M[x := N]$.

Preuve. Par induction sur la structure du terme M

1. $M \equiv x$; alors $@(\lambda x.x; N) \rightarrow_{AR} N \equiv x[x := N]$
2. $M \equiv y$ $(y \neq x)$; alors $@(\lambda x.y; N) \rightarrow_{AR} y \equiv y[x := N]$
3. $M \equiv (\lambda y.U)$; alors $@(\lambda x.(\lambda y.U); N) \rightarrow_{AR} \lambda g.@(\lambda x.@(\lambda y.U; g); N)$ par hypothèse d'induction il est facile de voir que $@(\lambda y.U; g) \rightarrow_{AR} U[y := g]$ et donc $\lambda g.@(\lambda x.@(\lambda y.U; g); N) \equiv \lambda g.[x:=N][y:=g]U = \lambda y.[x:=N]V$ (avec $V \equiv_{\alpha} U$)
4. $M \equiv (UV)$; alors $@(\lambda xUV; N) \rightarrow_{AR} @(\lambda x.U; N)@(\lambda x.V; N) \equiv ([x:=N]U[x:=N]V)$ (par hypothèse d'induction) et c'est équivalent à $([x:=N]UV)$.

Lemme III.3.3 (lemme de substitution).

On rappelle ici le lemme de substitution du λ-calcul, qui nous sera très utile pour démontrer la confluence du λ@. Soient M, N, P des λ-termes.

Si $x \neq y$, si $x \notin Fv(P)$, alors : $M[x / N][y / P] \equiv M[y / P][N[y / P] / x]$

Preuve : voir [CHL 92] et [HS 86].

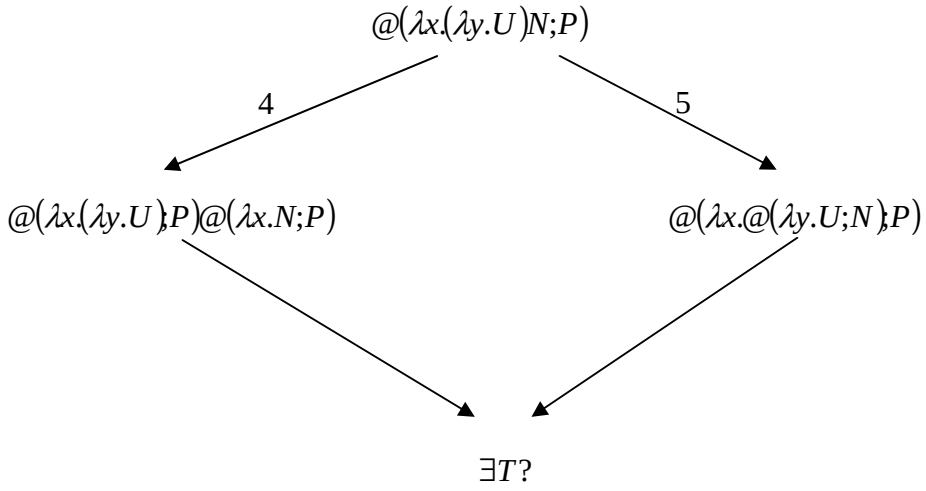
III.4 Confluence du λ@-calcul.

Le λ@ comporte une seule paire critique entre les règles :

$@(\lambda x.MN;P) \rightarrow @(\lambda x.M;P)@(\lambda x.N;P)$ … (Règle 4) et

$(\lambda x.M)N \rightarrow @(\lambda x.M;N)$ … (Règle 5) du fait de l'existence d'une substitution σ donné par :

$\sigma = \{\sigma(N) = N; \sigma(M) = \lambda y.U\}$ qui peut donner deux chemin de réduction pour un terme similaire à la partie gauche de la règle (4), donc :



Les deux termes de la paire critique sont équivalents, il nous reste à le démontrer :

$@(\lambda x.(\lambda y.U);P)@(\lambda x.N;P)$ ($1^{ère}$ composante de la paire critique)

↓

$(\lambda g.@(\lambda x.@(\lambda y.U;g);P))@(\lambda x.N;P)$ Par la règle (3)

↓

par le lemme(IV.3.2), avec $U_1=[g/y]U$

↓

$(\lambda g.U_2)N_1$ par le lemme(IV.3.2),avec $U_2=[P/x]U_1$, $N_1=[P/x]N$

↓

$[N_1/g]U_2 \equiv ([P/x]N/g)\,(\,([P/x])([g/y])U\,) \equiv [P/x]([P/x]N/y)U$ …**(1)**

$@(\lambda x.@(\lambda y.U;N);P)$ ($2^{ième}$ composante de la paire critique)

↓

$@(\lambda x.U_3;P)$ Avec $U_3=[N/y]U$

↓

$[P/x]U_3 \equiv [P/x]([N/y]U)$ … **(2)**

Les équations **(1)** et **(2)** sont équivalentes par le lemme de substitution du λ-calcul classique, ce qui nous montre que le λ@-calcul est localement confluent.

III.5 Normalisation forte.

III.5.1 Modèle de terminaison.

Comme nous l'avons vu dans les chapitres précédents, un système de réécriture à la propriété de terminaison s'il n'admet pas des séquences de réécritures infinies ; alors toutes les séquences de réécriture s'arrêtent sur une forme normale.

La terminaison d'un système de réécriture R peut être prouvée en translatant chaque terme vers un élément d'un ensemble non vide A équipé d'un ordre bien fondé $\succ$ de tel sorte que $[s]\succ[t]$ Pour tout $s\ R\ t$ où $[.]$ dénote la translation.

III.5.2 Préservation de la normalisation forte pour λ@.

Dans ce qui suit nous allons utiliser la combinaison des technique de RPO (recursif path ordring) [DER 79, 82,87], [DP 01] et l'étiquetage sémantique (semantic labelling) [KLO 80]..

Pour démontrer la préservation de la normalisation forte du λ@ nous avons suivi les étapes suivantes :

1. trouver un sous ensemble de λ@ termes qui contient les termes purs et fortement normalisant par la β-réduction du λ-calcul.
2. traduire les termes de cet ensemble vers un ensemble de terme du premier ordre.
3. définir un système de réécriture de premier ordre sur ce nouveau ensemble.
4. monter que le système de réécriture du premier ordre préserve la réduction du λ@.

Définitions. III.5.1

- On dit que le terme $M\in SN_R$ si M est fortement normalisant par la relation de réduction R
- On utilise $AR(M)$ pour dénoter la forme normale de M par le système $\rightarrow_{AR}$ et $\beta(M)$ pour la forme normale de M si elle existe.
- La norme $Max\_red(M)$ correspond à la longueur maximum de β-réduction qui commence par $AR(M)$, si $AR(M)\in SN_\beta$.

Lemme III.5.1. $\forall M,N\in\lambda@$: $AR(@(\lambda x.M;N))\equiv AR(M)[x:=AR(N)]$

Preuve.

Soit $M \equiv x$ alors par hypothèse d'induction $AR(N) \equiv AR(x)[x := AR(N)] \equiv AR(N)$;

1. soit $M\equiv y$ avec $y\neq x$ alors on a par hypothèse d'induction $AR(y)\equiv AR(y)[x:=AR(N)\equiv AR(y)]$; ;
2. soit $M\equiv PQ$ $AR(@(\lambda x.PQ;N))\equiv [x:=AR(N)](P)[x:=AR(N)](Q)$; (par hypothèse d'indiction)
3. soit $M\equiv\lambda y.P$ nous avons $AR(@(\lambda x.(\lambda y.P);N))\equiv AR(\lambda g.@(\lambda x.@(\lambda y.P;g);N))$ par hypothèse d'induction $(\lambda g.[x:=AR(N)][y:=g]AR(P))$

Lemme III.5.2.

- Pour tous les termes M, N :
  1. si $M\to_{\lambda@} N$ alors $AR(M)\equiv AR(N)$
  2. si $M\to_{@\beta} N$ alors $AR(M)\to_{\beta}^{*} AR(N)$

Preuve :

1. triviale
2. par induction sur la structure de M, on traite ici les cas important :
   - si $M\equiv(\lambda x.M_1)M_2$, $N\equiv@(\lambda x.M_1;M_2)$. Alors $AR(M)\equiv AR(\lambda x.AR(M_1))AR(M_2)$ qui se réduit par β à $AR(M_1)[x:=AR(M_2)]\equiv AR(@(\lambda x.M_1.M_2))\equiv AR(N)$ (par le lemme III.5.1)
   - si $M\equiv(@(\lambda x.M_1;M_2))$, $N\equiv@(\lambda x.M_1;C)$.Or :
     $AR(M)\equiv AR(M_1)[x:=AR(M_2)]\to_{\beta}^{*} AR(M)\equiv AR(M_1)[x:=AR(C)]$ (par hypothèse d'induction) $\equiv AR(N)$ par le lemme III.5.1.
   - $M\equiv(@(\lambda x.M_1;M_2))$, $N\equiv@(\lambda x.C;M_2)$; analogue au cas précédent.
   - $M\equiv(@(\lambda x.M_1;M_2))$, $N\equiv@(\lambda x.C_1;C_2)$; analogue au 2ième cas.

III.6. Rappel de la définition du RPO.

Dans cette section nous allons rappeler la définition de l'ordre récursif sur les chemins (récursif path ordering) déjà présenté dans le chapitre I dans le cadre des méthodes de preuve de terminaison des systèmes de réécritures du premier ordre.

III.6.1 Définition.

Soient F un ensemble de symboles de fonctions, X un ensemble de variables tel que $F \cap X = \{\}$ et $T(F, X)$ l'ensemble des termes définis sur F et X. Soient $\triangleright$ un ordre partiel sur F et τ une fonction qui assigne à chaque symbole de fonction $f \in F$ un statut qui est l'un des mots mult ou lex.

L'ordre récursif sur les chemins $\succ_{RPO}$ sur $T(F,X)$ induit par $\triangleright$ et τ est définit par :

$f(s_1,...,s_m) \succ_{RPO} g(t_1,...,t_n)$

ssi $\exists i[s_i = g(t_1,..., t_n) \vee s_1 \succ_{RPO} g(t_1,..., t_n)]$

$\vee \left(f \triangleright g \wedge \forall j[f(s_1,..., s_m) \succ_{RPO} t_j]\right)$

$\vee \left(f = g \wedge \forall j[f(s_1,..., s_m) \succ_{RPO} t_j] \wedge \langle s_1,..., s_m\rangle \succ_{RPO}^{\tau(f)} \langle t_1,..., t_n\rangle\right)$

$\succ_{RPO}^{lex}$, $\succ_{RPO}^{mult}$ sont respectivement les extensions lexicographique et multi ensemble de l'ordre récursif sur les chemins $\succ_{RPO}$ et ils sont définis par :

- $(s_1,...,s_m) \succ_{RPO}^{lex} (t_1,...,t_n)$ ssi pour un $i \le m,n$ $s_1=t_1,...,s_{i-1}=t_{i-1}$, $s_i \succ_{RPO} t_i$ ou $s_1=t_1,...,s_m=t_m$, $m>n$.
- $\langle s_1,...,s_m\rangle \succ_{RPO}^{mult} \langle t_1,...,t_n\rangle$ ssi le multi ensemble $\{t_1,...,t_n\}$ peut être obtenu à partir de $\{s_1,...,s_m\}$ en faisant des remplacement d'un élément s par des éléments t tel que $s \succ_{RPO} t$, cette opération de remplacement peut être faite une ou plusieurs fois.

Théorème III.6.2. (Dershowitz)

Soit $\triangleright$ un ordre partiel et τ une fonction de statut définie sur l'ensemble des symboles de fonctions F, soit $\succ_{RPO}$ l'ordre récursif sur les chemins induit par $\triangleright$ et τ. Alors

$\succ_{RPO}$ est bien fondé $\Leftrightarrow$ $\triangleright$ est bien fondé.

Preuve : voir [DER 82], [FZ 94].

III.7 PSN pour le λ@.

Ici nous allons utilisé les techniques (RPO) et le semantic labelling(SL) pour démontrer que le λ@ préserve la forte normalisation du λ-calcul, c'est à dire que si un terme est fortement normalisant dans le λ-calcul il est aussi fortement normalisant dans le λ@. Il nous reste à signaler que la technique de RPO est valable pour les systèmes de réécriture du premier ordre, donc on a besoin de traduire les termes du λ@ en des termes du premier ordre et cette traduction doit préserver la réduction, Donc on va étiqueter certain symbole de fonctions par des entiers qui représentent les longueurs maximum des séquences de réductions, c'est à dire qu'on va travailler sur un sous ensemble de λ@ termes.

Définition III.7.1.

On définit l'ensemble $SubSn_{@}$ (un sous ensembles de l'ensemble des λ@ termes) par :

$SubSn_{@} = \{M \in \lambda @ \ / \ \forall N$ un sous terme de M , $AR(N) \in SN_{\beta} \}$, en d'autres termes $Max\_red(N) < \infty$ pour tout sous terme N de M .

Lemme III.7.1.

$Max\_red(@(\lambda x.M;N)) < Max\_red((\lambda x.M)N)$

Preuve :

Nous avons $AR((\lambda x.M)N) \equiv (\lambda x.AR(M))AR(N) \rightarrow_{\beta} [x := AR(N)]AR(M) \equiv AR(@(\lambda x.M;N))$

Il est clair que si $Max\_red((\lambda x.M)N) = n$ alors $Max\_red(@(\lambda x.M;N)) = n-1$.

Lemme III.7.2.

Si $M \in SubSn_{@}$ et $M \rightarrow_{\lambda @} N$ alors $N \in SubSn_{@}$.

Preuve : facile par induction sur la structure de M

- $M \equiv @(\lambda x.x;A)$ avec $A \in SubSn_{@}$; $M \rightarrow_{\lambda @} A$
- $M \equiv @(\lambda x.y;A)$, nous avons dans ce cas $M \rightarrow_{\lambda @} y \in SubSN_{@}$
- $M \equiv @(\lambda x.(\lambda y.A);B) \rightarrow N \equiv @(\lambda g.@(\lambda x.@(\lambda y.A;g);B))$; par le lemme III.5.2. (partie 1) on $AR(M) \equiv AR(N)$ ce qui implique que $N \in SubSn_{@}$.
- $M \equiv @(\lambda x.AB;P)$; pour le même argument que le cas précédent.
- $M \equiv (\lambda x.A)B \rightarrow_{@\beta} N \equiv @(\lambda x.A;B)$ **;** par lemme III.5.2. (partie 2)

Définition III.7.2.

Soit λ_e un ensemble de termes du premier ordre augmenté par des étiquettes i avec i un entier naturel et qui est défini par la syntaxe suivante :

$$T ::= \# \mid T \|n\| T \mid \lambda T \mid T|T|_n \mid IT$$

Tel que I est le combinateur d'identité de la logique combinatoire

On défini aussi la transformation $\phi : \lambda@ \rightarrow \lambda_e$ par :

$$\phi(x) = \#$$
$$\phi(MN) = \phi(M) \|n\| \phi(N) \quad \text{Avec n= } Max\_red(MN)$$
$$\phi(\lambda x.M) \rightarrow \lambda\phi(M)$$
$$\phi(@(\lambda x.M; N)) =$$
$$\begin{cases} Q(M)|Q(N)|_n \; n = \max\_ red(@(\lambda x.M; N)) \\ IN \quad / si(x = M) \end{cases}$$

Il est clair que ϕ est bien définie sur tous les $\lambda@$ termes.

Définition III.7.3.

Soit $\rightarrow_{\lambda e}$ une relation de réduction définie sur l'ensemble λ_e par le système de réécriture du premier ordre suivant :

$$(\lambda M) \|m\| N \rightarrow M|N|_n \text{ si } m>n$$
$$(M \|m\| N)|P|_n \rightarrow (M|P|_p) \|q\| (N|P|_r) \quad \text{si } n \geq p,q,r$$
$$(\lambda M)|P|_n \rightarrow \lambda(M|P|_n)$$
$$I|P| \rightarrow P$$
$$M|P| \rightarrow M$$

Et les deux règles Down :

$$M \|m\| N \rightarrow M \|n\| N \qquad \text{si } m>n$$
$$M|N|_m \rightarrow M|N|_n \qquad \text{si } m>n$$

Remarque :

Les deux dernières règles sont appelées Decr dans [ZAN 95], elles servent à décrémenter les étiquettes dans l'applications et l'@ si une bêta réduction est effectuée à l'intérieur.

Lemme III.7.3.

La relation de réduction $\rightarrow_{\lambda e}$ est une sous relation d'un certain ordre récursif sur les chemins. (il existe une relation de précédence $\triangleright$ telle que $\forall M, N \in \lambda_e$ si $M \rightarrow_{\lambda e} N$, alors $M \succ_{RPO} N$ où $\succ_{RPO}$ est l'ordre récursif sur les chemins induit par $\triangleright$)

Preuve : soit la fonction de précédent $\triangleright$ définie par :

$$\| \|_m \triangleright | \|_n \triangleright \| \|_n \triangleright \lambda \triangleright \#$$

Et la fonction de statut τ donnée par :

$$\tau(\| \|_n) = \tau(| |_n) = \tau(\lambda) = \text{lex.}$$

$\rightarrow_{\lambda e}$ Est une sous relation de l'ordre récursif sur les chemins induit par les fonctions $\triangleright$ et τ.

Corollaire III.7.1.

$\rightarrow_{\lambda e}$ est fortement normalisant.

Preuve :

Par le théorème de Dershowitz le $\succ_{RPO}$ défini plus haut est fortement normalisant. Donc par le lemme III.7.3. $\rightarrow_{\lambda e}$ est fortement normalisant.

Lemme III.7.4. (Préservation de la réduction)

Si M est $\lambda @$ terme et $M \rightarrow_{\lambda @} M$ alors $\varphi(M) \rightarrow^{+}_{\lambda e} \phi(N)$.

Preuve : par induction sur la structure de M ; les cas les plus intéressants sont :

$M \equiv @(\lambda x.x, M_1) \rightarrow_{\lambda @} M_1 \equiv N$ alors $\varphi(M) = \#|\varphi(M_1)|_m \rightarrow_{\lambda e} \varphi(M_1) \equiv \varphi(N)$

Où $m = Max\_\text{Re}\, d(M)$

$M \equiv @(\lambda x.y, M_1) \rightarrow_{\lambda @} y \equiv N$ alors $\varphi(M) \equiv \varphi(M_1)|\varphi(M_2)|_m \rightarrow_{\lambda e} \varphi(M_1) \equiv \varphi(N)$ avec

$m = Max\_\text{Re}\, d(M)$

$M \equiv @(\lambda x.(\lambda y.M_1); M_2) \rightarrow_{\lambda @} (\lambda g.@(\lambda x.@(\lambda y.M_1); M_2)) \equiv N$;

$\varphi(M) \equiv \varphi(\lambda y.M_1)|\varphi(M_2)|_n \rightarrow_{\lambda e} (\lambda \varphi(M_1))|\varphi(M_2)|_n \equiv \lambda(\varphi(M_1)|\varphi(M_2)|_n)$

$\phi(N) \equiv \lambda(\phi(@(\lambda y.M_1; g))|\phi(M_2)|_n) \equiv \lambda(|\phi(M_1)|\phi(g)|_m|\phi(M_2)|_n) \equiv \lambda(\phi(M_1)|\phi(M_2)|_n)$

où $M \equiv (\lambda x.M_1)M_2$ alors $M \rightarrow_{\lambda @} N \equiv @(\lambda x.M_1; M_2)$

$\phi(M) \equiv (\lambda\phi(M_1))_{\|m\|}\phi(M_2) \rightarrow_{\lambda e} \phi(M_1)(\phi(M_2))_{\|n\|} \equiv \phi(N)$ Où $m=Max\_red(M)$ et $n = Max\_\mathrm{Re}d(N)$; il est à noter que $m>n$ est assuré par le lemme III.7.1.

$M \equiv @(\lambda x.M_1 M_2; M_3)$ alors $M \rightarrow_{\lambda @} N \equiv @(\lambda x.M_1; M_3)@(\lambda x.M_2; M_3)$. On a

$\phi(M) = (\phi(M_1)_{\|m\|}\phi(M_2))\phi(M_3)|_n \rightarrow_{\lambda e} (\phi(M_1)\phi(M_3)|_p)_n(\phi(M_2)\phi(M_3)|_q) \equiv \phi(N)$

Où $n = Max\_\mathrm{Re}d(M)$, $p = Max\_\mathrm{Re}d(@(\lambda x.M_1; M_3))$, $q = Max\_\mathrm{Re}d(@(\lambda x.M_2; M_3))$, et $n \geq p,q$.

$M \equiv M_1 M_2 \rightarrow_{\lambda @} M_3 M_2$ alors $\varphi(M) \equiv \varphi(M_1)_{\|m\|}\varphi(M_2) \rightarrow_{\lambda e} \varphi(M_3)_{\|m\|}\varphi(M_2)$

(par hypothèse d'induction) $\rightarrow_{\lambda e} \varphi(M_3)_{\|n\|}\varphi(M_2)$ où $m = Max\_\mathrm{Re}d(M) \geq n = Max\_red(N)$

Lemme III.7.5.

$\forall M \in \lambda @$ Termes, $M \in SubSn_{@} \Leftrightarrow M$ est $\lambda @$ fortement normalisant.

Preuve : découle immédiatement du lemme III.7.4.

Théorème III.7.1

Le $\lambda @$ préserve la normalisation forte du λ-calcul.

Preuve : soit M un terme fortement normalisant (M est pure), alors $M \in SubSn_{@}$, donc par le lemme III.7.5 M est fortement normalisant.

III.8 Théorème de confluence du λ@

Théorème III.8.1

Le $\lambda @$ est confluent.

Preuve :

$\lambda @$ est faiblement confluent et termine, donc sa confluence résulte immédiatement par le lemme de Newman.

Conclusion

Le λ@ calcul est une extension simple du λ-calcul classique qui n'utilise aucun codage supplémentaire pour désigner les termes.

Dans ce mémoire nous nous sommes principalement intéressé à l'étude de la confluence et la préservation de normalisation forte du λ@ calcul.

En constatant, à travers notre étude des calculs avec substitution explicite, que la confluence et la préservation de la normalisation forte des calculs ne sont pas des propriétés évidentes à établir directement, nous avons considéré le λ@ en le présentant comme un formalisme de réécriture d'ordre supérieur.
Ainsi, nous avons prouvé la normalisation forte du λ@ en utilisant une combinaison des techniques du 'semantic labelling' et l'ordre récursif sur les chemin. La confluence faible du λ@ est évidente après l'étude et la résolution de ses paires critiques.

La confluence du λ@ découle immédiatement de sa préservation de normalisation forte et sa confluence faible par le lemme de Newman.

La simplicité des règles de réécriture du λ@ calcul rend intéressant la définition d'une machine efficace exécutant les réductions de ce calcul. Il serait aussi intéressant de définir une version typée pour calcul.

Bibliographie

[ACCL 91] Abadi M, Cardelli L, Curien P.L, and Lévy J.J **Explicit substitution**, *Digital systems* research center publication 1991.

[ACCL 90] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J Lévy. ***Explicit substitutions.*** *Journal of Functional Programming,* 1990.

[AG 2000] Thomas Arts et Jürgen Giesl ***Termination of term rewriting using dependency pairs.*** *Theoretical Computer Science 236,133-178,* 2000.

[AG 97] T. Arts et J. Giesl ***Automatically proving termination where simplification ordering fail***. In Michel Bidiot et Max Dauchet, éditeurs, *theory and practice of software development,* volume 1214 des *Lecture notes in Computer Science*, Lille, France, avril 1997. Springer – Verlag.

[ART1 47] ARTIN, E. (1947): ***Theorie of braids.*** *Ann of math*. (2) 48 (1947) 101-126

[ART2 47] ARTIN, E. (1947).***Braids and permutations.*** *Ann* of Math.vol.48 (1947),235-254

[BAR 84] H.P. Barendregt ***Lambda-calculus, its Syntax and its Semantics***. Coll. Studies in Logic and the Foundation of Mathematics, *Elsevier Science Publishers* B.V. (North Holland), Amsterdam, 1984.

[BAR 81] H. P. Barendregt ***The Lambda Calculus: its Syntax and Semantics.*** N 103 in Studies in Logic and the Foundations of Mathematics. 1981.

[BBLR 96] Z.-E.-A, Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. ***λv, a calculus of explicit sub- stitutions which preserves strong normalisation***. *Journal of Functional Programming*, 1996.

[BD 86] Leo Bachmair, Nachum Dershowitz: **Commutation, Transformation, and Termination**. *CADE* 1986: 5-20

[BLO 99] R. Bloo and H. Geuvers ***Explicit substitution: on the edge of strong normalisation***. Theoretical Computer Science, 211 1999.

[BLO 97] R. Bloo ***Preservation of Termination for Explicit Substitution.*** PhD thesis, Eindhoven University, 1997.

[BN 04] F. Baader and T. Nipkow. **Term Rewriting and All That**. *Cambridge University Press*, 2004.

[CHL 92] P.-L. Curien, T. Hardin, and J.-J. Levy ***Confluence properties of weak and strong calculi of explicit substitutions***. Technical Report 1617, *INRIA* Rocquencourt, 1992

[CUR 86] P.-L. Curien ***Categorical combinators, sequential algorithms and functional programming***. Pitman, 1986.

[DAL 79] Dallas S.Lankford ***On proving term rewriting systems are Noetherian.*** Rapport technique MTP-3, Mathematics Department, *Louisiana Tech. Univ*, 1979.

[DB 72] N.G. de Bruijn ***Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem.*** *Indagationes Math*. 1972

[DAU 92] Max Dauchet ***Simulation of Turing machines by a regular rewrite rule***. *Theoretical Computer science,* September 1992.

[DB 78] N.G. de Bruijn ***A namefree lambda calculus with facilities for internal definition of expressions and segments***. Technical Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology 1978.

[DER 87] Nachum Dershowitz ***Termination of rewriting.*** *Journal of Symbolic Computation,* février 1987.

[DER 82] Nachum Dershowitz ***Termination of rewriting systems***. *Theoretical Computer Science* mars 1982.

[DER 79] Dershowitz, N ***A note on simplification ordering***, *information processing Letters* 1979.

[DG 01] R. David and B. Guillaume ***A λ-calculus with explicit weakening and explicit substitutions***. Mathematical Structure in Computer Science, 11(1), 2001.

[DG 99] R. David and B Guillaume. ***The λ_I -calculus.*** *In Proceedings of WEST APP*, 1999.

[DP 01] Nachum Dershowitz and David Plaisted, **Rewriting, in Handbook of Automated Reasoning**, Volume 1, *Alan Robinson and Andrei Voronkov, eds*, North-Holland, 2001, pp. 535-610

[DZ 79] Nachum Dershowitz et Zohar **Manna Proving termination with multiset ordering.** *Communications of ACM***, aôut 1979.**

[FKP 79] Maria C. F. Ferreira, Delia Kesner, Laurence Puel: **lambda-Calculi with Explicit Substitutions Preserving Strong Normalization**. *Appl. Algebra Eng. Commun. Comput.* 9(4): 333-371 (1999)

[FZ 96] M.C.F Ferreira et H.Zantema ***Total termination of term rewriting***. *Applicable Algebra in Engineering,Communication and Computing,*1996.

[FZ 94] Ferreira, M.C.F and Zantema, H ***Well-foundedness of Term Ordrings,*** Technical Report UU-CS-1994-46, Utrecht University.

[GAR 69] GARSIDE, F.A. (1969). **The braid group and the other groups**. *Quart. J. Math 20* (1969) 235-254

[GMMS 04] M.Gandrieu, C.Massoutie, Mohamed Mezghiche, P.Sallé **λ@ a Calculus without substitution and variable renaming.**

[HAR 89] T. Hardin ***Confluence results for the pure strong categorical combinatory logic CCL : lambda- calculi as subsystems of CCL.*** *Theoretical Computer Science*, 1989.

[HD 78] Gérard Huet et Dallas S.Lankford ***On the uniform halting problem for term rewriting systems***. *Rapport de recherche 283, INRIA*, mars 1978.

[HIN 80] Hindley, J.R (1964).**The churche-Rosser property and applications to term rewriting systems**. *JACM*, vol.27, No.4 (1980), 797-821.

[HS 86] J.R Hindley and Seldin J.P ***Introduction To combinators and λ-calculs;*** Cambridge University Press 1986.

[HUE 80] G. Huet, **Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems,** *Journal of the ACM* 27 (1980) 797-821.

[JAN 88] *M. Jantzen.* M. Jantzen: **Confluent String Rewriting**. Springer Verlag, 1988 EATCS Mon. 14.

[KEI 01] Keiichirou Kusakari et Yoshihito Toyama ***On proving AC-termination by AC-Dependency pairs.IEICE*** *Transformations on Information and Systems, E84-D (5),* 2001.

[KEI 00] Keiichirou Kusakari ***Termination, AC-Termination and dependency pairs of term rewriting systems.*** These de doctorat, JAIST, 2000.

[KLO 82] KLOP, J.W. (1982) ***Term rewriting systerms.*** In Vol.2 of *Handbook of logic in Computer Science*(eds.S Abramsky, D. Gabbay & T. Maibaum), Oxford University Press 1992, p.1-116

[KLO 80] KLOP, J.W. (1980).***Combinatory reduction systems***. *Mathematical Centre Tracts* 127, Amsterdam 1980.

[KMY 99] Keiichirou Kusakari, Masaki Nakamura et Yochihito Toyama ***.Argument filtering transformation***. In Gopalon Nadathur, éditeur, *Principales and practice of Declarative Programming, International Conference PPDP'99,* volume 1702 *des lecture in computer science,* Paris 1999. Springer-Verlag.

[KR 98] F. Kamareddine and A Rïos. Bridging de Bruijn i***ndices and variable names in explicit substitutions calculi.*** Logic Journal of the Inter est Group of Pure and Applied Logic, 1998.

[KR 97] F. Kamareddine and A. Rïos ***Extending a λ-calculus with explicit substitutions which preserves strong normalization in to a confluent calculus on open terms.*** Journal of Functional Programming ,1997

[KRI 90] KRIVINE ***Lambda-calcul, types et modèles.*** *Coll. études et recherches en informatique*, Masson, Paris, 1990. 5Jean-Louis)

[LES 94] P. Lescanne ***From lambda-sigma to lambda-epsilon: a journey through calculi of explicit substitutions***. *In Proceedings of the 21st ACM Symposium on Principles of Programming Languages* 1994

[MA 90] Masahito Kurihara et Azuma Ohuchi ***Modularity of simple termination of term rewriting systems with shared constructors***. Rapport de technique SF-36, Hokkaido University, Sapporo, Japan, 1990.

[MAK 68] MAKANIN, G.S (1968).**The conjugacy problem in the braid group**. *Soviet Math* Dok1.9 (1969)1156-1157

[MEL 95] P.-A. Mellies ***Typed λ-calculi with explicit substitutions ma y not terminate***. *In Proceedings of TLCA'95 Springer*, 1995.

[MID 01] Aart Middeldorp ***Approximating dependency graphs using tree automata techniques.*** *In First International Joint Conference on Automated reasoning,* volume 2083 *des lectures notes in Artificial Intelligence,* Sienne, Italie, Juin 2001. Springer-Verlag.

[NEW 42] NEWMAN, M.H.A (1942).**On theories with a combinatorial definition of "equivalence"**. *Annals of Mathematics*, 43(2): 223-243, April 1942.

[O'D 95] Michael J.O'Donnell ***Computing in systems described by equations***. In *Lecture in Computer Science, 136,* 1995

[PLA 78] David A Plaisted. ***A recursively defined ordering for proving termination of term rewriting systems.*** *Rapport de recherche R-78-943*, University of Illinois, Urbana, IL, Etats-unis, 1978.

[ROS 73] Barry K.Rosen ***Tree-manipulating systems et Church-Rosser theorems***. *J. of the Association for Computing Machinery,* 20(1), Janvier 1973.

[ROSE 92] K.H. Rose ***Explicit cyclic substitutions.*** *In Proceedings of CTRS, Springer- Verlag*, 1992.

[RTH 97] Robin Milner, Mds Tofte et Robert W.Harper ***the Definition of standard ML (Revised).*** *MIT Press, Cambridge*, Mssachustts, Etats Unis, 1997.

[SPJ 03] S.Peyton Jones. **Haskell 98 Language and Libraries**, combridge university press 2003

[SS 91] SCHMIDT, G & STRÖHLEIN, T. (1991).**Reletions and Graphs** – Discrete Mathematics for Computer Scientists. Springer-Verlag, EATCS Monographs on *Theoretical Computer Science*, 1991

[STA 75] J. Staples. **Church-Rosser theorems for replacement systems**. *In J. Crossley, editor, Algebra and Logic, number 450 in Lecture Notes in Mathematics,* pages 291--307. Springer-Verlag, 1975. 27

[TERESE 03] Terese: ***Term Rewriting Systems***. *Cambridge university press*, 2003.

[TOY 87] Y. Toyama ***Counterexample to terminating for the direct sum of term rewriting systems.*** *Information Processing Letters,* 25, Avril 1987.

[XP 93] Xavier Leroy et Pierre Weis**. Manuel de Référence du langage Caml**, *InterEditions, Paris 1993, ISBN 2-7296-0492-8*

[ZAN 95] H. Zantema ***Termination of term rewriting, by semantic labelling***. *Fundamental informatiae 24,* 1995