Stage DEA : Extension au second ordre du λ-calcul typé avec motifs

Pierre COURTIEU

Juin 1996

Table des matières

1 Introduction

1.1 Motivations

L'un des aspects les plus importants des langages déclaratifs et en particulier des langages fonctionnels comme ML[MTH90], (MIRANDA[Tur85]) ou HASKELL[HPJe92], est la possibilité de définir des fonctions par cas. Ce n'est pas la caractéristique principale de ces langages mais elle a sans doute contribué pour une large part à leur succès. Elle permet en effet l'écriture de programmes clairs et faciles à lire. La définition récursive de la factorielle (écrite ici en CAML) en est un exemple bien connu:

```
let rec fact = function
0 -> 1
|n -> n * fact(n-1)
```

Il existe classiquement deux manières de définir une fonction: la manière extensionnelle, qui consiste à donner l'ensemble des couples (argument, résultat) définissant complètement la fonction; et la manière intentionnelle, qui consiste à décrire la manière d'associer un argument à son résultat, c'est-à-dire la manière de "calculer" le résultat en fonction des arguments. La définition par cas permet de combiner les avantages des deux méthodes en proposant au programmeur une solution beaucoup plus souple qui consiste à définir une liste de couples (forme des arguments, calcul). On peut ainsi choisir l'une des deux méthodes, ou une solution intermédiaire dans laquelle chaque couple correspond à un ensemble (pas nécessairement fini) d'arguments possibles. La "forme des arguments" est définie à l'aide des motifs, comme dans l'exemple suivant écrit en CAML :

```
let rec pgcd = function
0, n -> n
| m,n -> pgcd(n mod m, m);;
```

Ici, les deux motifs 0,n et m,n permettent de différencier deux cas dans la forme des arguments de la fonction pgcd, et de leur associer un calcul différent. La notion de définition par cas est donc intrinsèquement liée à celle de motif (ou pattern).

On sait que le λ-calcul est souvent pris comme modèle de la programma-

tion fonctionnelle. Or les définitions par cas et le traitement explicite des motifs n'y sont pas possibles. Delia Kesner, Laurence Puel et Val Breazu-Tannen présentent donc dans [BTKP93] et [Kes93] un λ-calcul typé avec motifs qui permet de formaliser l'une et l'autre de ces particularités. Ils se basent sur l'isomorphisme de Curry-Howard et s'inspirent du calcul des séquents de Gentzen du 1^{er} ordre pour construire un système de règles de typage au 1^{er} ordre pour ce calcul avec motifs. Ils obtiennent un calcul monomorphe qui possède toutes les propriétés que l'on attend : décidabilité du typage, un système de réduction confluent, etc.
L'idée de se baser sur un système de déduction logique comme celui de Gentzen pour en tirer un système de typage se justifie par les propriétés que l'on connaît depuis longtemps sur le premier qui pourront être transposées au second. Outre le fait d'être un formalisme plus proche du fonctionnement des langages fonctionnels, ce calcul est aussi une des premières interprétation "à la Curry-Howard" du calcul des séquents de Gentzen. Interprétation rendue difficile notamment par la présence des règles gauches.

Le fait que ce calcul soit monomorphe constitue un handicap. En effet, un des atouts majeurs des langages fonctionnels actuels est qu'ils incluent tous d'une manière ou d'une autre le polymorphisme. Le polymorphisme donne un pouvoir d'expression plus fort aux langages, et permet d'éviter d'écrire plusieurs fois la même fonction pour des types différents. Il permet de définir des fonctions qui peuvent s'appliquer à des arguments de plusieurs types. Pour cela on introduit dans les types des variables de type, qui peuvent prendre plusieurs valeurs, et qui définissent à chaque application le type attendu par une fonction en argument. On peut par exemple définir l'identité polymorphe:

$$\mathrm{let\ id} = \mathrm{fun(t{:}type)fun(x{:}t)x}$$

et son application aux types int et bool:

$$\begin{array}{c} \mathrm{id[int]} : \mathrm{int} \to \mathrm{int} \\ \mathrm{id[bool]} : \mathrm{bool} \to \mathrm{bool} \end{array}$$

On remarque ici que la variable t est un paramètre de la fonction id, c'est le polymorphisme paramétrique. Le fait que la fonction id peut être appliquée à des arguments de différents types est classiquement exprimé à l'aide du quantificateur universel, on dit que la fonction id est de type $\forall$t.t$\to$t, et que appliquer id à int consiste à instancier t avec int, ainsi le type de id[int] est int $\to$ int, c'est-à-dire le type de l'identité sur les entiers.

D'autre part, le polymorphisme est l'interprétation habituelle ("à la Curry-Howard") des règles de déduction logique du second ordre, il est donc intéressant d'appliquer ce principe au calcul de Gentzen.

Le travail de ce stage a consisté à étendre le λ-calcul typé avec motifs au polymorphisme en se basant cette fois sur le calcul de Gentzen du 2^{nd} ordre. Plus exactement, sur le calcul du 1^{er} ordre augmenté des règles traitant le quantificateur universel.

Plan du rapport Nous commençons dans la section 2 par rappeler la syntaxe du λ-calcul typé avec motifs au premier ordre, ainsi que le système de typage, les deux évaluateurs et le système de réduction associés au calcul dans [BTKP93]. Dans la section 3, nous reprenons successivement la syntaxe, le système de typage, les deux évaluateurs et le système de réduction pour les étendre au polymorphisme et redémontrer les propriétés de cette extension du calcul.

1.2 Travaux dans le domaine

Samson Abramsky [Abr93] et Jean Gallier [Gal93] proposent chacun une interprétation du calcul de Gentzen, ils remarquent déjà que les règles gauches de ce calcul peuvent correspondre à la construction de motifs, mais leurs syntaxes ne permettent pas la construction de motifs imbriqués, et ils ne traitent pas le polymorphisme.

Vincent Van Oostrom [Oos90] travaille aussi sur la notion de motif, mais en tant que terme arbitraire du λ-calcul, ce qui permet la réduction à l'intérieur des motifs. Cette interprétation est plus proche de la symétrie inhérente du calcul de Gentzen, mais s'éloigne de l'esprit des langages fonctionnels. Il ne s'intéresse pas non plus au polymorphisme.

Toujours dans un cadre monomorphe, Simon Peyton Jones et Philippe Wadler [Jon87] ont étudié la généralisation de la λ-abstraction classique à l'abstraction sur les motifs, mais ils ne définissent qu'une syntaxe abstraite pour les langages fonctionnels.

Yves Lafont [Laf89] propose une interprétation du calcul des séquents de la logique propositionnelle intuitionniste. Les règles gauches sont interprétées comme règles de construction de motifs imbriqués. Malgré plusieurs différences techniques, son *clausal calculus* est le système le plus proche de celui de Delia Kesner, Laurence Puel et Val Breazu-Tannen [BTKP93] sur lequel porte ce stage. Mais lui non plus ne s'intéresse pas au polymorphisme.

Enfin, Jerzy Tiuryn [Tiu96] propose une interprétation du calcul de Gentzen au second ordre traitant le polymorphisme, mais il n'interprète pas les règles gauches, et ne construit pas de motif.

2 Le λ-calcul typé avec motifs au 1^{er} ordre

2.1 Syntaxe

La particularité de ce λ-calcul typé avec motifs réside dans la nature de la λ-abstraction. Elle ne se fait plus seulement sur les variables comme en λ-calcul, mais sur les motifs, ce qui constitue une sorte de généralisation puisque un motif peut être soit une variable, soit des expressions plus complexes. On peut ainsi voir une fonction prendre explicitement en argument une paire:
λ <x,y>. ...
Ou une définition par cas:
λ (0|n). ...

Alors que les termes habituels du λ-calcul typé sont typés grâce aux règles droites, cette particularité du calcul est traitée grâce à l'interprétation des règles gauches du calcul des séquents de Gentzen, comme nous le verrons dans les règles de typage.

Le calcul utilise un outil classique des langages fonctionnels qui généralise la substitution: le filtrage (ou pattern matching). Le filtrage de P par M, M match P, est la substitution qui remplace les variables de P par les sous-termes de M correspondants. Par exemple, $< \lambda$x.M, y> match (a,b) doit être évalué en la substitution $\{$ a $\leftarrow \lambda$x.M, b $\leftarrow$ y$\}$.

Voici la syntaxe du λ-calcul typé avec motifs monomorphe de [BTKP93]:

- Variables usuelles: x, y, z ...
- Variables de communication: ξ, ρ,ψ ...

L'ensemble des types contient un ensemble dénombrable de types atomiques ou de base, et il est clos par les opérations de $\times$(produit), $+$(somme) et $\rightarrow$ (type fonctionnel)

On a trois sortes d'expressions, les types, les termes, et les motifs :

Types:

$$A ::= \iota \mid A \times A \mid A + A \mid A \to A$$

Où ι contient les types de base

Motifs:

$$P ::= \_ \mid x \mid \sharp x \mid < P, P > \mid (P|_\xi P) \mid P@P$$

Termes de communication:

$$T ::= \xi \mid L \mid R$$

Termes:

$$\begin{array}{lcl} M & ::= & x \mid < M, M > \mid inl_A(M) \mid inr_A(M) \mid [M|_T M] \mid \lambda P : A.M \\ & & \mid \text{let M be P : A in M} \mid \text{Z of M is P:A in M} \\ \\ Z & ::= & x \mid \lambda P : A.M \end{array}$$

Le motif $\sharp$x correspond aux variables de type A $\to$ B, on peut ainsi prendre explicitement en argument une variable qui représente une fonction. Cette notation permettra de traiter au cours de l'évaluation les types fonctionnels de manière analogue aux type sommes ou produits.

Le motif _ est le motif anonyme comme on le connaît dans les langages fonctionnels.

Le motif $(P|_\xi P)$ est le motif correspondant à la définition par cas. Il est utilisé de pair avec l'expression $[M|_T M]$ de la manière suivante :
λ (P1 $|_\xi$ P2): A+B.[M1 $|_\xi$ M2].
La sémantique étant que si on applique cette expression à un argument de la forme P1 (respectivement P2), ξ prendra au cours de l'évaluation la valeur L (left) (respectivement R (right)) dans l'expression [M1 $|_\xi$ M2] qui se réduira alors en M1 (respectivement M2).

L'expression Z of M is P:A in N est une version complexe de l'application du λ-calcul pur. Intuitivement, par abus de notation, on peut interpréter l'expression Z of M is P:A in N comme une expression de la forme "(λP.N)(Z M)", c'est à dire l'expression N dans laquelle le motif P a subi un filtrage avec l'application de Z à M.

L'expression let M be P:A in N correspond à une construction courante dans les langages fonctionnels: let M be P:A in N est l'expression N dans laquelle le motif P a été filtré par M, ce qui pourrait s'exprimer informellement par "(λP.N)M".

Les deux construction inl_A et inr_A servent à donner le type somme A+B à un terme de type A ou B.

On remarque la présence des annotations de type dans les expressions. On gardera cette caractéristique dans l'extension au second ordre, ce qui permettra d'obtenir un calcul dont le typage est décidable. En effet, notre calcul est au second ordre, et le typage du λ-calcul du second ordre sans annotation de type est indécidable.

On dira qu'un terme est clos si aucune occurrence d'une variable n'est libre. On étend également la notion classique de contexte au λ-calcul typé avec motifs : un contexte C[] est un pseudo-terme construit avec les mêmes alphabets de types et de motifs que le λ-calcul avec motifs, avec un élément en plus dans l'alphabet des termes : $\diamond$. On définit l'expression C[t], où t est un terme, comme (C[])[t/$\diamond$].

Enfin on note par Var(P) l'ensemble des variables du motif P, et par FV(M) l'ensemble des variables libres du terme M.

On donne ci-dessous la traduction du λ-calcul typé en λ-calcul typé avec motifs. Pour chaque terme M du premier, on donne sa traduction M* dans le second.

$$
\begin{array}{lcl}
(x)^* & \stackrel{def}{=} & x \\
<M,N>^* & \stackrel{def}{=} & <M^*,N^*> \\
(\lambda x{:}A.M)^* & \stackrel{def}{=} & (\lambda x{:}A.M^*) \\
\mathrm{inl}_B(M)^* & \stackrel{def}{=} & \mathrm{inl}_B(M^*) \\
\mathrm{inr}_B(M)^* & \stackrel{def}{=} & \mathrm{inr}_B(M^*) \\
(\pi_i(M))^* & \stackrel{def}{=} & \text{let } M^* \text{ be } <x_1,x_2>: A \times B \text{ in } x_i \\
(MN)^* & \stackrel{def}{=} & \text{let } M^* \text{ be } \sharp z{:}A \to B \text{ in } (z \text{ of } N^* \text{ is } y \text{ in } y)
\end{array}
$$

2.2 Exemples de fonctions

On donne ici quelques exemples de fonctions écrites en λ-calcul typé avec motifs. On admet un ensemble de types de base où le type int est défini par les constructeurs 0 et succ(), et le type bool par les deux constructeurs true et false. Chaque fonction est suivie d'un exemple d'application dans lequel M(N) est une abréviation de M of N is Q in Q.

IDNAT		identité sur les entiers
IDNAT	=	λx:nat.x
IDNAT(succ(0))	=	succ(0)

ZERO?		test si l'argument est un entier égal à 0
ZERO?	=	$\lambda(0 \mid_\xi \mathrm{succ}(\_)):\mathrm{nat}.[\mathrm{true} \mid_\xi \mathrm{false}]$
ZERO?(0)	=	true

Lors de l'évaluation de ZERO?(0), le filtrage de $(0 \mid_\xi \mathrm{succ}(\_))$ par 0 remplace ξ par L (left), l'expression se réduit donc a $[\mathrm{true} \mid_L \mathrm{false}]$ et le résultat est finalement true. Le méchanisme se généralise aux définitions par cas plus complexes en imbriquant les disjonctions:

AND		conjonction sur les booléens
AND	=	$\lambda <(\mathrm{true} \mid_\xi \mathrm{false}),(\mathrm{true} \mid_\zeta \mathrm{false})>:\mathrm{bool} \times \mathrm{bool}.$
		$[[\mathrm{true} \mid_\zeta \mathrm{false}\ ] \mid_\xi [\mathrm{false} \mid_\zeta \mathrm{false}]]$
$\mathrm{AND}(<\mathrm{true},\mathrm{false}>)$	=	false

PLUS		Addition sur les entiers
PLUS	=	$\lambda(<0,m> \mid_\xi <\mathrm{succ}(n),m>):(\mathrm{nat} \times \mathrm{nat}).$
		$[m \mid_\xi \mathrm{succ}(\mathrm{PLUS}\ <n,m>)]$
$\mathrm{PLUS}(<\mathrm{succ}(0),\mathrm{succ}(0)>)$	=	succ(succ(0))

On peut définir des fonctions sur les listes (on a omis les annotations de type pour plus de clarté), les listes sont définies avec les deux constructeurs nil et cons(,):

APPEND		effectue la concaténation de liste :
APPEND	=	$\lambda <(\mathrm{nil} \mid_\xi \mathrm{cons}(M,L)),(\mathrm{nil} \mid_\zeta \mathrm{cons}(P,W))>.$
		$[[\mathrm{nil} \mid_\zeta \mathrm{cons}(P,W)] \mid_\xi [\mathrm{cons}(M,L)$
		$\mid_\zeta \mathrm{cons}(M, \mathrm{APPEND}(L,\mathrm{cons}(P,W))]]$

SUFFIXLIST		rend la liste de tous les suffixes d'une liste :
SUFFIXLIST	=	λ(z@(nil $\|_\xi$ cons(_,l))).
		[cons(nil, nil) $\|_\xi$ cons(z, SUFFIXLIST l)]
FLATTEN		aplanit un niveau d'imbrication dans une liste :
FLATTEN	=	λ(nil $\|_\xi$ cons((nil $\|_\zeta$ cons(x,l)),L)).
		[[nil $\|_\zeta$ nil] $\|_\xi$ [FLATTEN L $\|_\zeta$ cons(x, FLATTEN (cons(l, L)))]]
LENGTH		rend la longueur d'une liste :
LENGTH	=	λ(nil $\|_\xi$ cons(x,l)):list nat. [0 $\|_\xi$ succ(LENGTH l)]

On constate que les définitions par cas s'écrivent de manière très intuitive, comme dans les langages fonctionnels.

2.3 Le typage

2.3.1 Les règles de typage

Les jugements de typage sont de la forme $\Gamma \rhd M : A$, où M est un terme, A un type, et Γ un multi-ensemble d'affirmations de la forme P:B, où P est un motif et B un type. Les règles de typage apparaissent sous forme de fractions dont le numérateur contient un ensemble de jugements de typage, et le dénominateur un seul jugement de typage. Une règle se lit comme suit: si tous les jugements du numérateur sont valides, alors celui du dénominateur est valide.

$$\frac{\Gamma_1 \rhd M_1 : A_1 \quad \Gamma_2 \rhd M_2 : A_2}{\Gamma' \rhd M' : A'}$$

On appelle une dérivation un enchaînement de règles dont toutes les branches commençent par des axiomes, et qui aboutit à un jugement unique.

Dans l'esprit de l'isomorphisme de Curry-Howard, on interprète chaque règle du calcul des séquents de Gentzen par une règle de typage. De manière classique, les règles droites correspondent aux règles de typage des termes du λ-calcul. L'originalité et l'intérêt théorique de ce système de typage réside dans l'interprétation des règles gauches: elles servent à typer les motifs. L'environnement de typage contient des motifs (et non des variables comme dans les systèmes classiques), qui sont construits récursivement par les règles gauches puis "injectés" à droite dans les termes par les règles (let), ($\rightarrow$-right)et ($\rightarrow$-left).

Règles de typage au premier ordre:

Calcul de Gentzen:

$$A_1, \ldots, A_n \vdash A_i$$

$$(\wedge\text{-}right)\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$(\vee\text{-}right1)\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$$

$$(\vee\text{-}right2)\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$(\supset\text{-}right)\frac{A, \Gamma \vdash B}{\Gamma \vdash A \supset B}$$

$$(cut)\frac{\Gamma \vdash A \quad A, \Gamma \vdash B}{\Gamma \vdash B}$$

$$(contraction)\frac{A, A, \Gamma \vdash C}{A, \Gamma \vdash C}$$

$$(\wedge\text{-}left)\frac{A, B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C}$$

$$(affaiblissement)\frac{\Gamma \vdash B}{A, \Gamma \vdash B}$$

$$(\supset\text{-}left)\frac{\Gamma \vdash A \quad B, \Gamma \vdash C}{A \supset B, \Gamma \vdash C}$$

$$(\vee\text{-}left)\frac{A, \Gamma \vdash C \quad B, \Gamma \vdash C}{A \vee B, \Gamma \vdash C}$$

Calcul avec motifs:

$$x_1 : A_1, \ldots, x_n : A_n \rhd x_i : A_i$$

où les x_j sont des variables distinctes

$$\frac{\Gamma \rhd M : A \quad \Gamma \rhd N : B}{\Gamma \rhd < M, N >: A \times B}(\times\text{-}right)$$

$$\frac{\Gamma \rhd M : A}{\Gamma \rhd inl_B(M) : A + B}(+\text{-}right1)$$

$$\frac{\Gamma \rhd N : B}{\Gamma \rhd inr_A(N) : A + B}(+\text{-}right2)$$

$$\frac{P : A, \Gamma \rhd M : B}{\Gamma \rhd \lambda P : A.M : A \to B}(\to\text{-}right)$$

$$\frac{\Gamma \rhd M : A \quad P : A, \Gamma \rhd N : B}{\Gamma \rhd \text{let M be P:A in N:B}}(let)$$

$$\frac{P : A, Q : A, \Gamma \rhd M : B}{P@Q : A, \Gamma \rhd M : B}(layered)$$

$$\frac{P : A, Q : B, \Gamma \rhd M : C}{< P, Q >: A \times B, \Gamma \rhd M : C}(\times\text{-}left)$$

$$\frac{\Gamma \rhd M : B}{\_ : A, \Gamma \rhd M : B}(wildcard)$$

$$\frac{\Gamma \rhd N : A \quad Q : B, \Gamma \rhd M : C}{\sharp z : A \to B, \Gamma \rhd \text{z of N is Q:B in M} : C}(\to\text{-}left)$$

$$\frac{P : A, \Gamma \rhd M : C \quad Q : B, \Gamma \rhd N : C}{(P|_\xi Q) : A + B, \Gamma \rhd [M|_\xi N] : C}(+\text{-}left)$$

Pour rendre le typage stable par évaluation, on ajoute quelques règles de typage étendues qui ne correspondent pas à des règles du calcul de Gentzen. Elles correspondent aux expressions intermédiaires engendrées, au cours de l'évaluation, par l'application d'une substitution, et qui ne sont pas typables avec les règles précédentes:

$$\frac{\Gamma \rhd M : C}{\Gamma \rhd [M|_L N] : C}(L)$$

$$\frac{\Gamma \rhd M : C}{\Gamma \rhd [N|_R M] : C}(R)$$

$$\frac{\Gamma \rhd \lambda P : A.L : A \to B \quad \sharp z : A \to B, \Gamma \rhd \text{z of N is Q:B in M:C}}{\Gamma \rhd (\lambda P : A.L) \text{ of N is Q:B in M:C}}(app\text{-}of)$$

2.3.2 Exemples de typage

L'identité sur les naturels se type de la même manière qu'en λ-calcul pur:

$$\frac{x : nat \rhd x : nat}{\rhd (\lambda x : nat.x) : nat \to nat}(\to\text{-}right)$$

L'application de l'identité sur l'entier succ(0) se type grâce à la règle de typage étendue (app-of) et à la règle ($\to$ -left):

$$\frac{\dfrac{x : nat \rhd x : nat}{\rhd \lambda x : nat.x : nat \to nat}(\to\text{-}right) \quad \dfrac{\rhd succ(0) : nat \quad y : nat \rhd y : nat}{\sharp z : nat \to nat \rhd \text{z of succ(0) is y:nat in y:nat}}(\to\text{-}left)}{\rhd (\lambda x : nat.x) \text{ of succ(0) is y:A in y:A : nat}}(app\text{-}of)$$

La fonction AND et sa définition par cas se type simplement par la règle (+-left):

$$
\dfrac{\dfrac{\dfrac{\dfrac{true : bool \rhd true : bool \quad false : bool \rhd false : bool \quad true : bool \rhd false : bool \quad false : bool \rhd false : bool}{true : bool, (true|_{\varsigma} false) : bool \rhd [true|_{\varsigma} false] : bool \quad false : bool, (true|_{\varsigma} false) : bool \rhd [false|_{\varsigma} false] : bool}\,(+\text{-}left)}{(true|_{\xi} false) : bool, (true|_{\varsigma} false) : bool \rhd [[true|_{\varsigma} false]|_{\xi}[false|_{\varsigma} false]] : bool}\,(+\text{-}left)}{< (true|_{\xi} false), (true|_{\varsigma} false) >: bool \times bool \rhd [[true|_{\varsigma} false]|_{\xi}[false|_{\varsigma} false]] : bool}\,(\times\text{-}left)}{\rhd \lambda < (true|_{\xi} false), (true|_{\varsigma} false) >: bool \times bool.[[true|_{\varsigma} false]|_{\xi}[false|_{\varsigma} false]] : (bool \times bool) \rightarrow bool}\,(\rightarrow\text{-}right)
$$

2.4 Les évaluateurs

On donne deux évaluateurs pour ce λ-calcul. L'un est paresseux, et l'autre strict. La particularité de l'évaluateur paresseux est qu'il retarde l'évaluation des arguments d'une fonction tant qu'il n'en a pas besoin pour obtenir un résultat. L'évaluateur strict, au contraire, évalue complètement tous ses arguments avant d'évaluer une fonction.

Les deux évaluateurs sont décrits en sémantique naturelle[Kah87] où les assertions sont de la forme M $\Downarrow$ K, où M et K sont des termes clos et K est le résultat final de l'évaluation de M (sa forme canonique).

L'utilisation des motifs oblige à introduire l'opération de filtrage (ou pattern matching) dans les évaluateurs à travers l'opérateur match qui prend en paramètres un terme M et un motif K, et renvoie une substitution.

Évaluateur paresseux:

Du fait de l'évaluation paresseuse, certaines expressions ne seront pas évaluées tout de suite, et l'opération de filtrage sur ces expressions devra être retardée également. L'opérateur Match est donc évalué lui aussi de manière paresseuse et son évaluation dépend de celle des termes.

$$\frac{\text{Match N on P} \Downarrow_l \sigma \quad \text{let } L[\sigma] \text{ be Q:A in M} \Downarrow_l K}{(\lambda P.L) \text{ of N is Q:A in M} \Downarrow_l K}$$

$$\frac{\text{Match M on P} \Downarrow_l \sigma \quad N[\sigma] \Downarrow_l K}{\text{let M be P in N} \Downarrow_l K}$$

$$\frac{}{< M, N > \Downarrow_l < M, N >} \quad \frac{}{\lambda P.M \Downarrow_l \lambda P.M}$$

$$\frac{}{inl(M) \Downarrow_l inl(M)} \quad \frac{}{inr(M) \Downarrow_l inr(M)}$$

$$\frac{M \Downarrow_l K}{[M|_L N] \Downarrow_l K} \quad \frac{N \Downarrow_l K}{[M|_R N] \Downarrow_l K}$$

$$\frac{L \Downarrow_l < M, N > \quad \text{Match M on P} \Downarrow_l \sigma \quad \text{Match N on Q} \Downarrow_l \theta \quad Dom(\sigma) \cap Dom(\theta) = \emptyset}{\text{Match L on } < \text{P,Q} > \Downarrow_l \sigma \cup \theta}$$

$$\frac{\text{Match L on P} \Downarrow_l \sigma \quad \text{Match L on Q} \Downarrow_l \theta \quad Dom(\sigma) \cap Dom(\theta) = \emptyset}{\text{Match L on P @ Q} \Downarrow_l \sigma \cup \theta} \quad \frac{}{\text{Match L on \_} \Downarrow_l \emptyset}$$

$$\frac{L \Downarrow_l inl(M) \quad \text{Match M on P} \Downarrow_l \sigma \quad \xi \notin Dom(\sigma)}{\text{Match L on (P } |_\xi \text{ Q)} \Downarrow_l [\mathbf{L}/\xi] \cup \sigma} \quad \frac{}{\text{Match L on x} \Downarrow_l [L/x]}$$

$$\frac{L \Downarrow_l inr(N) \quad \text{Match N on Q} \Downarrow_l \sigma \quad \xi \notin Dom(\sigma)}{\text{Match L on (P } |_\xi \text{ Q)} \Downarrow_l [\mathbf{R}/\xi] \cup \sigma} \quad \frac{L \Downarrow_l \lambda P.M}{\text{Match L on } \sharp \text{ z} \Downarrow_l [\lambda P.M/z]}$$

Évaluateur strict:

Dans l'évaluateur strict, le filtrage est en revanche indépendant de l'évaluation des termes.

$$\frac{M \Downarrow_e K \quad Match(K,P)=\sigma \quad N[\sigma] \Downarrow_e L}{\text{let M be P in N} \Downarrow_e L}$$

$$\frac{N \Downarrow_e K \quad Match(K,P)=\sigma \quad \text{let L}[\sigma] \text{ be Q:A in M} \Downarrow_e L}{(\lambda P.L) \text{ of N is Q:A in M} \Downarrow_e L}$$

$$\frac{M \Downarrow_e K}{[M|_L N] \Downarrow_e K} \qquad \frac{N \Downarrow_e K}{[M|_R N] \Downarrow_e K}$$

$$\frac{M \Downarrow_e K \quad N \Downarrow_e L}{< M,N > \Downarrow_e < K,L >} \qquad \frac{M \Downarrow_e K}{inl(M) \Downarrow_e inl(K)}$$

$$\frac{N \Downarrow_e L}{inr(N) \Downarrow_e inr(L)} \qquad \frac{}{\lambda P.M \Downarrow_e \lambda P.M}$$

$$\begin{array}{lll}
Match(< M_1,M_2 >,< P_1,P_2 >) & = \sigma_1 \cup \sigma_2 & où \begin{cases} Match(M_1,P_1)=\sigma_1 \\ Match(M_2,P_2)=\sigma_2 \\ Dom(\sigma_1) \cap Dom(\sigma_2) = \emptyset \end{cases} \\
Match(M, P_1 @ P_2) & = \sigma_1 \cup \sigma_2 & où \begin{cases} Match(M,P_1)=\sigma_1 \\ Match(M,P_2)=\sigma_2 \\ Dom(\sigma_1) \cap Dom(\sigma_2) = \emptyset \end{cases} \\
Match(inl(M),(P|_\xi Q)) & = [\mathbf{L}/\xi] \cup \sigma & où \begin{cases} Match(M,P)=\sigma \\ \xi \notin Dom(\sigma) \end{cases} \\
Match(inr(M),(P|_\xi Q)) & = [\mathbf{R}/\xi] \cup \sigma & où \begin{cases} Match(M,Q)=\sigma \\ \xi \notin Dom(\sigma) \end{cases} \\
Match(\lambda P.M, \sharp z) & = [\lambda P.M/z] & \\
Match(M,x) & = [M/x] & \\
Match(M,_) & = \emptyset & \\
Match(M,P) & = \mathbf{fail} & autrement
\end{array}$$

Ces deux évaluateurs sont déterministes et ils préservent les types, c'est à dire que si $\triangleright M : A$ et $M \downarrow K$, alors $\triangleright K : A$. Ils sont également confluents (pour les preuves de ces propriétés voir [BTKP93].

2.5 Un système général de réduction

On donne un système de réécriture sur tous les termes purs. Ce mécanisme de réduction non déterministe est plus général que les deux évaluateurs, qui ne sont en fait que deux stratégies particulières pour ce système. La relation d'un pas de réduction $\longrightarrow$ est obtenue en orientant l'ensemble des équations suivantes de gauche à droite, où Match(M,P) est définie de la même manière que dans l'évaluateur strict.

$$\frac{Match(M,P) = \sigma}{\text{let M be P in N} = N[\sigma]}$$

$$\frac{Match(M,P) = \sigma}{(\lambda P : A.L) \text{ of M is Q:B in N} = \text{let L}[\sigma] \text{ be Q:B in N}}$$

$$[M|_L N] = M \qquad [M|_R N] = N$$

$$\text{let M be } P_1 @ P_2 \text{ in N} = \text{let } < M, M > \text{ be } < P_1, P_2 > \text{ in N}$$

$$(\lambda P_1 @ P_2 .L) \text{ of M is Q:A in N} = (\lambda < P_1, P_2 > .L) \text{ of } < M, M > \text{ is Q:A in N}$$

La deuxième règle généralise la β-réduction du λ-calcul pur.

La relation de réduction $\Rightarrow$ entre termes purs s'obtient par la fermeture sous tous contextes de la relation $\longrightarrow$, c'est à dire que si M $\longrightarrow$ M', alors pour tout contexte C[], on a C[M] $\Rightarrow$ C[M'].

La relation $\Rightarrow$ est stable par substitution, préserve les types, et est confluente (preuve dans [BTKP93]).

3 Extension du calcul au 2^{nd} ordre

3.1 Syntaxe

Il s'agit d'enrichir la syntaxe pour inclure le polymorphisme **paramétrique** et **explixicite**. On ajoute pour cela aux variables usuelles et de communication les variables de type. On introduit également un nouveau motif correspondant aux expressions polymorphes, et enfin on ajoute l'abstraction et l'instanciation de type.

syntaxe du λ-calcul typé avec motif polymorphe :

- Variables usuelles : x, y, z ...
- Variables de communication : ξ, ρ,ψ ...
- On ajoute les variables de type : $\alpha, \beta \ldots$

L'ensemble des types contient un ensemble dénombrable de types atomiques ou de base et un ensemble dénombrable de variables de type, et il est clos par les opérations de $\times$(produit), $+$(somme), $\rightarrow$ (type fonctionnel) et $\forall$ (type polymorphe).
Types:

$$A ::= \iota \mid \tau \mid A \times A \mid A + A \mid A \rightarrow A \mid \forall \alpha . A$$

Où ι contient les types de base et τ les variables de type.

Motifs: on ajoute le motif Δx, qui correspond au type de la forme $\forall\alpha$.A, de la même manière que le motif $\sharp$x correspond au type A $\rightarrow$ B :

$$P ::= \_ \mid x \mid \sharp x \mid \Delta x \mid < P, P > \mid (P|_{\xi} P) \mid P@P$$

Termes de communication:

$$T ::= \xi \mid L \mid R$$

Termes: On ajoute une abstraction sur les variables de type : $\Lambda\alpha.M$. On ajoute également l'instanciation de type correspondante: Z'[A] is P:A in M est l'expression M dans laquelle le motif P est "filtré" par l'expression polymorphe Z' instanciée avec le type A.

$$\begin{array}{lcl} M & ::= & x \mid < M, M > \mid inl_A(M) \mid inr_A(M) \mid [M|_T M] \\ & & \mid \lambda P : A.M \mid \Lambda\alpha.M \mid \text{let M be P : A in M} \\ & & \mid \text{Z of M is P:A in M} \mid \text{Z'[A] is P:A in M} \end{array}$$

$$\begin{array}{lcl} Z & ::= & x \mid \lambda P : A.M \\ Z' & ::= & x \mid \Lambda\alpha.M \end{array}$$

Exemples de fonctions:

Voici quelques exemples de fonctions écrites dans le nouveau calcul, on reprend notemment l'identité polymorphe Id de l'introduction.

Identité polymorphe	$\Lambda\alpha.\lambda x : \alpha.x$
Id. poly. instanciée avec int	$(\Lambda\alpha.\lambda x : \alpha.x)[int]$ is $id : int \to int$ in id
Id. poly. instanciée par int	let $(\Lambda\alpha.\lambda x : \alpha.x)$ be $\Delta x : \forall\alpha.\alpha \to \alpha$ in $x[int]$ is
avec le let	$id : int \to int$ in id
test liste vide polymorphe	$\Lambda\alpha.\lambda(cons(x, \_)|_\xi nil) : list\alpha.[false|_\xi true]$

3.2 Le typage

3.2.1 Les règles de typage

Il s'agit donc d'ajouter les règles du calcul des séquents de Gentzen qui traitent le quantificateur universel et d'en donner une interprétation dans le λ-calcul typé avec motifs. Les deux règles du second ordre du calcul de Gentzen sont les suivantes:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall\alpha.A} \text{ où } \alpha \text{ n'est pas libre dans } \Gamma \text{, et :} \qquad \frac{A[t/x], \Gamma \vdash C}{\forall x.A, \Gamma \vdash C}$$

L'interprétation de la première règle est assez immédiate, et comme on a choisi le polymorphisme paramétrique, la nouvelle règle de typage sera :

$$\frac{\Gamma \rhd M : A}{\Gamma \rhd \Lambda\alpha.M : \forall\alpha.A} \qquad (\forall\text{-}right)$$

où α n'est pas libre dans Γ.

Pour la deuxième, on s'inspire de la règle déjà existante ($\rightarrow$-left) :

$$\frac{\Gamma \rhd N : A \quad Q : B, \Gamma \rhd M : C}{\sharp z : A \rightarrow B, \Gamma \rhd \text{z of N is Q:B in M} : C}(\rightarrow \text{-}left)$$

et par analogie entre l'abstraction de type et la λ-abstraction, on obtient la règle suivante:

$$\frac{Q : A[t/\alpha], \Gamma \rhd M : C}{\Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in M:C}}(\forall\text{-}left)$$

où la substitution $[t/\alpha]$ porte sur les variables de type.

L'enrichissement de la syntaxe a introduit une nouvelle expression étendue: ($\Lambda\alpha$.L)[t] is Q:A in M. On ajoute donc une nouvelle règle de typage étendue (on remarque encore l'analogie avec la règle (app-of)):

$$\frac{\Gamma \rhd \Lambda\alpha.L : \forall\alpha.A \quad \Delta z : \forall\alpha.A, \Gamma \rhd \text{z[t] is Q:A}[t/\alpha] \text{ in M:C}}{\Gamma(\Lambda\alpha.L)[t]\text{is Q} : A[t/\alpha]\text{in M:C}}(inst)$$

On donne à la page suivante le système de typage complet, chaque règle est associée à une règle du calcul de Gentzen:

Règles de typage au premier ordre:

Calcul de Gentzen:	Calcul avec motifs:
$A_1, \ldots, A_n \vdash A_i$	$x_1 : A_1, \ldots, x_n : A_n \rhd x_i : A_i$ où les x_j sont des variables distinctes
$(\wedge\text{-}right)\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$	$\dfrac{\Gamma \rhd M : A \quad \Gamma \rhd N : B}{\Gamma \rhd < M, N >: A \times B}(\times\text{-}right)$
$(\vee\text{-}right1)\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$	$\dfrac{\Gamma \rhd M : A}{\Gamma \rhd inl_B(M) : A + B}(+\text{-}right1)$
$(\vee\text{-}right2)\dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$	$\dfrac{\Gamma \rhd N : B}{\Gamma \rhd inr_A(N) : A + B}(+\text{-}right2)$
$(\supset\text{-}right)\dfrac{A, \Gamma \vdash B}{\Gamma \vdash A \supset B}$	$\dfrac{P : A, \Gamma \rhd M : B}{\Gamma \rhd \lambda P : A.M : A \to B}(\to\text{-}right)$
$(cut)\dfrac{\Gamma \vdash A \quad A, \Gamma \vdash B}{\Gamma \vdash B}$	$\dfrac{\Gamma \rhd M : A \quad P : A, \Gamma \rhd N : B}{\Gamma \rhd \text{let M be P:A in N:B}}(let)$
$(contraction)\dfrac{A, A, \Gamma \vdash C}{A, \Gamma \vdash C}$	$\dfrac{P : A, Q : A, \Gamma \rhd M : B}{P@Q : A, \Gamma \rhd M : B}(layered)$
$(\wedge\text{-}left)\dfrac{A, B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C}$	$\dfrac{P : A, Q : B, \Gamma \rhd M : C}{< P, Q >: A \times B, \Gamma \rhd M : C}(\times\text{-}left)$
$(affaiblissement)\dfrac{\Gamma \vdash B}{A, \Gamma \vdash B}$	$\dfrac{\Gamma \rhd M : B}{\_ : A, \Gamma \rhd M : B}(wildcard)$
$(\supset\text{-}left)\dfrac{\Gamma \vdash A \quad B, \Gamma \vdash C}{A \supset B, \Gamma \vdash C}$	$\dfrac{\Gamma \rhd N : A \quad Q : B, \Gamma \rhd M : C}{\sharp z : A \to B, \Gamma \rhd \text{z of N is Q:B in M} : C}(\to\text{-}left)$
$(\vee\text{-}left)\dfrac{A, \Gamma \vdash C \quad B, \Gamma \vdash C}{A \vee B, \Gamma \vdash C}$	$\dfrac{P : A, \Gamma \rhd M : C \quad Q : B, \Gamma \rhd N : C}{(P\vert_\xi Q) : A + B, \Gamma \rhd [M\vert_\xi N] : C}(+\text{-}left)$

Règles de typage du second ordre:

Calcul de Gentzen:

$$(\forall - left)\frac{A[t/x], \Gamma \rhd C}{\forall x.A, \Gamma \rhd C}$$

$$(\forall - right)\ \frac{\Gamma \rhd \Delta, A}{\Gamma \rhd \Delta, \forall\alpha.A}$$

avec α non libre dans Γ.

Calcul avec motifs:

$$\frac{Q : A[t/\alpha], \Gamma \rhd M : C}{\Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in M:C}}(\forall - left)$$

$$\frac{\Gamma \rhd M : A}{\Gamma \rhd \Lambda\alpha.M : \forall\alpha.A}(\forall - right)$$

avec α non libre dans Γ.

Règles étendues:

$$\frac{\Gamma \rhd M : C}{\Gamma \rhd [M|_L N] : C}(L)$$

$$\frac{\Gamma \rhd M : C}{\Gamma \rhd [N|_R M] : C}(R)$$

$$\frac{\Gamma \rhd \lambda P : A.L : A \to B \quad \sharp z : A \to B, \Gamma \rhd \text{z of N is Q:B in M:C}}{\Gamma \rhd (\lambda P : A.L) \text{ of N is Q:B in M:C}}(\mathit{app\text{-}of})$$

$$\frac{\Gamma \rhd \Lambda\alpha.L : \forall\alpha.A \quad \Delta z : \forall\alpha.A, \Gamma \rhd \text{z[t] is Q:A}[t/\alpha] \text{ in M:C}}{\Gamma(\Lambda\alpha.L)[t]\text{is Q} : A[t/\alpha]\text{in M:C}}(inst)$$

3.2.2 Exemples de typage

L'identité polymorphe se type simplement:

$$\cfrac{\cfrac{x : \alpha \rhd x : \alpha}{\rhd \lambda x : \alpha.x : \alpha \to \alpha}(\to\text{-}right)}{\rhd \Lambda\alpha.\lambda x : \alpha.x : \forall\alpha.\alpha \to \alpha}(\forall\text{-}right)$$

L'identité polymorphe peut être instanciée par un autre type:

$$
\dfrac{(\forall\text{-}right)\dfrac{(\to\text{-}right)\dfrac{x:\alpha \rhd x:\alpha}{\rhd \lambda x:\alpha.x:\alpha\to\alpha}}{\rhd \Lambda\alpha.\lambda x:\alpha.x:\forall\alpha.\alpha\to\alpha} \qquad \dfrac{id:int\to int \rhd id:int\to int}{\Delta z:\forall\alpha.\alpha\to\alpha \rhd z[int] \text{ is } id:int\to int \text{ in } id:int\to int}(\forall\text{-}left)}{(\Lambda\alpha.\lambda x:\alpha.x)[int] \text{ is } id:int\to int \text{ in } id:int\to int}(inst)
$$

ou encore:

$$
\dfrac{(\forall\text{-}right)\dfrac{(\to\text{-}right)\dfrac{x:\alpha \rhd x:\alpha}{\rhd \lambda x:\alpha.x:\alpha\to\alpha}}{\rhd \Lambda\alpha.\lambda x:\alpha.x:\forall\alpha.\alpha\to\alpha} \qquad \dfrac{id:int\to int \rhd id:int\to int}{\Delta x:\forall\alpha.\alpha\to\alpha \rhd x[int] \text{ is } id:int\to int \text{ in } id:int\to int}(\forall\text{-}left)}{let\ (\Lambda\alpha.\lambda x:\alpha.x) \text{ be } \Delta x:\forall\alpha.\alpha\to\alpha \text{ in } x[int] \text{ is } id:int\to int \text{ in } id:int\to int}(inst)
$$

3.2.3 Décidabilité du typage

Montrons que le système de typage est décidable.

On commence par démontrer une propriété utile pour le reste des démonstrations : les règles (×-left), (layered) et (wildcard) commutent avec toutes les autres règles, à condition qu'elles n'agissent pas sur le même motif.

Propriété 3.2.1 (Commutation) *Si une dérivation se termine par l'application de 2 règles de la forme:*

$$
(II)\dfrac{(I)\dfrac{\Theta,\Gamma,\Sigma \rhd M:A}{\Theta',\Gamma,\Sigma \rhd M':A'}}{\Theta',\Gamma,\Sigma' \rhd M'':A''} \quad \textit{resp. de la forme:} \quad \dfrac{\dfrac{\Theta_1,\Gamma,\Sigma \rhd M_1:A_1 \quad \Theta_2,\Gamma,\Sigma \rhd M_2:A_2}{\Theta',\Gamma,\Sigma \rhd M':A'}(I)}{\Theta',\Gamma,\Sigma' \rhd M'':A''}(II)
$$

où l'une des 2 règles (I) ou (II) est (×-left), (layered) ou (wildcard), alors il existe une dérivation, aboutissant au même résultat, qui se termine par:

$$(II)\frac{(I)\dfrac{\Theta,\Gamma,\Sigma \rhd M : A}{\Theta,\Gamma,\Sigma' \rhd N : B}}{\Theta',\Gamma,\Sigma' \rhd M'' : A''} \quad \textit{resp. de la forme:} \quad \frac{\dfrac{\Theta_1,\Gamma,\Sigma \rhd M_1 : A_1 \quad \Theta_2,\Gamma,\Sigma \rhd M_2 : A_2}{\Theta_1,\Gamma,\Sigma' \rhd M_1' : A_1' \quad \Theta_2,\Gamma,\Sigma' \rhd M_2' : A_2'}(I)}{\Theta',\Gamma,\Sigma' \rhd M'' : A''}(II)$$

La preuve est par analyse des cas, on montre ici les cas supplémentaires induits par la règle (∀-left), les deux autres règles (∀-right) et (inst) ne posent pas plus de difficulté :

Soit une dérivation se terminant par:

$$\frac{\dfrac{P : B, Q : C, R : A[t/\alpha], \Gamma \rhd M : D}{< P, Q >: (B \times C), R : A[t/\alpha], \Gamma \rhd M : D}(\times - left)}{\Delta x : \forall\alpha.A, < P, Q >: (B \times C), \Gamma \rhd \text{ x[t]is R:A}[t/\alpha] \text{ in M: D}}(\forall - left)$$

Montrons que la propriété s'applique en posant $\left\{\begin{array}{lcl} \Theta & = & \text{P:B,Q:C} \\ \Theta' & = & \text{<P,Q>:(B} \times \text{C)} \\ \Sigma & = & \text{R:A[t/}\alpha] \\ \Sigma' & = & \Delta \text{ x: } \forall\alpha \text{ .A} \end{array}\right.$.

On peut construire la même dérivation en inversant les deux dernières règles de la manière suivante:

$$\frac{\dfrac{P : B, Q : C, R : A[t/\alpha], \Gamma \rhd M : D}{\Delta x : \forall\alpha.A, P : B, Q : C, \Gamma \rhd \text{ x[t] is R:A}[t/\alpha] \text{ in M: D}}(\forall - left)}{\Delta x : \forall\alpha.A, < P, Q >: (B \times C), \Gamma \rhd \text{ x[t] is R:A}[t/\alpha] \text{ in M: D}}(\times - left)$$

On procède de même pour les deux autres cas:

$$\frac{\dfrac{\Gamma, Q : A[t/\alpha] \rhd N : C}{\_ : B, \Gamma, Q : A[t/\alpha] \rhd N : C}(wildcard)}{\_ : B, \Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in N: C}}(\forall - left)$$

s'inverse en:

$$\cfrac{\cfrac{\Gamma, Q : A[t/\alpha] \rhd N : C}{\Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in N : C}}\forall - left}{\_ : B, \Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in N : C}}(wildcard)$$

$$\cfrac{\cfrac{P : A, Q : A, R : B[t/\alpha], \Gamma \rhd M : D}{P@Q : A, R : B[t/\alpha], \Gamma \rhd M : D}(layered)}{\Delta x : \forall\alpha.B, P@Q : A, \Gamma \rhd \text{ x[t] is R:B}[t/\alpha] \text{ in M : D}}(\forall - left)$$

s'inverse en:

$$\cfrac{\cfrac{P : A, Q : A, R : B[t/\alpha], \Gamma \rhd M : D}{\Delta x : \forall\alpha.B, P : A, Q : A, \Gamma \rhd \text{ x[t] is R:B}[t/\alpha] \text{ in M : D}}(\forall - left)}{\Delta x : \forall\alpha.B, P@Q : A, \Gamma \rhd \text{ x[t] is R:B}[t/\alpha] \text{ in M : D}}(layered)$$

$\square$

Corollaire 3.2.2 *Soit une dérivation de la forme:*

$$\cfrac{\vdots}{Q : A, \Gamma \rhd M : C}$$

Il existe une dérivation de même longueur de la forme:

$$\cfrac{\cfrac{\vdots}{Q_1 : A_1, \ldots Q_n : A_n, \Gamma \rhd M : C}}{\cfrac{(layered, wildcard, \times\text{-}left)}{Q : A, \Gamma \rhd M : C}}$$

où les Q_i sont des motifs de la forme x, $(P \mid_\xi R)$, $\sharp x$, et Δx.

Preuve: Par récurrence sur la profondeur de la dérivation.

- Si la dérivation est un axiome, Q est une variable et la propriété est vérifiée.

- Soit une dérivation de longueur n+1 de la forme:

$$\frac{n \left\{ \begin{array}{c} \cdot \\ \cdot \\ \cdot \end{array} \right.}{Q : A, \Gamma \rhd M : C}$$

  - Si Q est déjà de la forme x, ♯x, Δx, ou (P $|_\xi$ R), la propriété est vérifiée.
  - Si Q est de la forme <P,R>, P@R, ou _ , alors on a au cours de l'évaluation (dans chaque branche), un enchaînement de 2 règles de la forme :

$$\frac{\dfrac{\dfrac{\vdots}{\Theta, \Gamma'' \rhd M'' : C''}}{Q : A, \Gamma'' \rhd M'' : C''}(I)}{\dfrac{Q : A, \Gamma' \rhd M' : C'}{\vdots}}$$

    où (I) est l'une des trois règles (wildcard),(layered) ou (×-left). En appliquant la propriété de commutation, on sait qu'il existe la même dérivation mais dans laquelle l'enchaînement est remplacé par:

$$\frac{\dfrac{\dfrac{\vdots}{\Theta, \Gamma'' \rhd M'' : C''}}{\Theta, \Gamma' \rhd M' : C'}}{\dfrac{Q : A, \Gamma' \rhd M' : C'}{\vdots}}(I)$$

    En itérant cette propriété dans chaque branche, on obtient à la fin une dérivatiom où la règle (I) est appliquée en dernier :

$$n \left\{ \begin{array}{c} \cdot \\ \cdot \\ \cdot \\ \Theta, \Gamma \rhd M : C \end{array} \right.$$
$$\overline{Q : A, \Gamma \rhd M : C}$$

Comme Q est de la forme _ , <P,R>, ou P@R, Θ est soit vide et la propriété est vérifiée, soit de la forme P:A,Q:B. Dans ce cas on applique l'hypothèse de récurrence successivement sur les deux motifs P et R dans la dérivation de longueur n:

$$n-1 \left\{ \begin{array}{c} \cdot \\ \cdot \\ \cdot \end{array} \right.$$
$$\overline{P : A, R : B, \Gamma \rhd M : C}$$

Et la propriété est vérifiée.

$\square$

On définit maintenant la destruction d'un environnement de typage, noté Decon(Γ), par induction sur les motifs :

Decon(_:A, Γ)	=	Decon(Γ)
Decon(x:A, Γ)	=	x:A, Decon(Γ)
Decon(♯ x, Γ)	=	♯ x, Decon(Γ)
Decon(<P,Q>:A × B, Γ)	=	Decon(P:A), Decon(Q:B), Decon(Γ)
Decon(P@Q:A, Γ)	=	Decon(P:A), Decon(Q:A) Decon(Γ)
Decon((P $\vert_\xi$ Q) : A, Γ)	=	(P $\vert_\xi$ Q) : A, Decon(Γ)

On ajoute le cas suivant par rapport à la définition de [BTKP93]

Decon(Δx:A, Γ) = Δx:A, Decon(Γ)

Lemme 3.2.3 *Pour n'importe quelle dérivation D qui termine en $\Gamma \rhd M : C$, il y a une dérivation de profondeur au plus celle de D, qui termine avec* $Decon(\Gamma) \rhd M : C$.

Preuve:

Par récurrence sur la profondeur de la dérivation $\Gamma \rhd M : C$, puis par cas selon la dernière règle utilisée.
Cas supplémentaires par rapport à [BTKP93]

- La dernière règle est la règle ($\forall$-$left$)

$$\dfrac{n\left\{\begin{array}{c} \vdots \\ Q : A[t/\alpha], \Gamma \rhd M : C \quad (I) \end{array}\right.}{\Delta x : \forall\alpha.A, \Gamma \rhd x[t] \text{ is Q:A}[t/\alpha] \text{ in M:C} \quad (II)}(\forall\text{-}left)$$

Il existe (par le corollaire 3.2.2) une dérivation de la forme :

$$\dfrac{n\left\{\begin{array}{c} \dfrac{p\left\{\begin{array}{c} \vdots \\ Q_1 : A_1, Q_2 : A_2 \ldots Q_n : A_n, \Gamma \rhd M : C \end{array}\right.}{n-p\left\{\begin{array}{c} (layered, wildcard, \times - left) \quad (III) \\ \hline Q : A[t/\alpha], \Gamma \rhd M : C \quad (I) \end{array}\right.} \end{array}\right.}{\Delta x : \forall\alpha.A, \Gamma \rhd x[t] \text{ is Q:A}[t/\alpha] \text{ in M:C} \quad (II)}\forall\text{-}left$$

où les Q_i sont des motifs de la forme : x, $(P|_{\xi}Q)$, $\sharp z$ et Δz.

D'après l'hypothèse de récurrence, il existe une dérivation de longueur au plus p qui termine avec $Decon(Q_1 : A_1, \ldots Q_n : A_n, \Gamma) \rhd M : C$, or $Decon(Q_1 : A_1, \ldots Q_n : A_n, \Gamma) = Q_1 : A_1, \ldots Q_n : A_n, Decon(\Gamma)$. Donc il existe une dérivation de la forme:

$$\dfrac{\leq p\left\{\begin{array}{c} \vdots \\ Q_1 : A_1, Q_2 : A_2 \ldots Q_n : A_n, Decon(\Gamma) \rhd M : C \end{array}\right.}{n-p\left\{\begin{array}{c} \dfrac{(III)}{Q : A[t/\alpha], Decon(\Gamma) \rhd M : C} \\ \end{array}\right.}$$

Ici on peut appliquer la règle $(\forall - left)$:

$$\dfrac{\leq p \left\{ \begin{array}{c} \vdots \\ Q_1 : A_1, Q_2 : A_2 \ldots Q_n : A_n, Decon(\Gamma) \rhd M : C \end{array} \right. \qquad n - p \left\{ \begin{array}{c} (III) \\ \hline Q : A[t/\alpha], Decon(\Gamma) \rhd M : C \end{array} \right.}{\Delta x : \forall\alpha.A, Decon(\Gamma) \rhd x[t] \text{ is Q:A}[t/\alpha] \text{ in M:C}}(\forall - left)$$

or $\Delta x : \forall\alpha.A, Decon(\Gamma) = Decon(\Delta x : \forall\alpha.A, \Gamma)$. Donc il existe une dérivation de profondeur au plus n+1 de la forme:

$$\dfrac{\vdots}{Decon(\Delta x : \forall\alpha.A, \Gamma) \rhd x[t] \text{ is Q:A}[t/\alpha] \text{ in M:C}}(\forall - left)$$

- La dernière règle est la règle $(\forall - right)$

$$\dfrac{n \left\{ \begin{array}{c} \vdots \\ \Gamma \rhd M : A \end{array} \right.}{\Gamma \rhd \Lambda\alpha.M : \forall\alpha.A}(\forall - right)$$

Par hypothèse de récurrence : il existe une dérivation de longueur au plus n telle que :

$$\dfrac{\vdots}{Decon(\Gamma) \rhd M : A}$$

Ici on applique la règle $(\forall - right)$:

$$\dfrac{\begin{array}{c}\vdots\\ Decon(\Gamma) \rhd M : A\end{array}}{Decon(\Gamma) \rhd \Lambda\alpha.M : \forall\alpha.A}(\forall - right)$$

Et on obtient une dérivation de longueur au plus n+1, c'est à dire au plus celle de la dérivation initiale et la propriété est vérifiée.
On vérifie de la même manière la propriété si la dernière règle est (inst).

$\square$

On démontre deux lemmes établissant la correspondance entre un jugement de la forme $Decon(\Gamma) \rhd M : C$ et le jugement $\Gamma \rhd M : C$.

Lemme 3.2.4 *Pour n'importe quelle dérivation qui termine en $Decon(\Gamma) \rhd M : C$ il y a une dérivation qui termine en $\Gamma \rhd M : C$.*

Preuve: Par application des règles ($\times$-left), (layered) et (wildcard).

$\square$

Pour montrer qu'un terme a un type unique, on associe à chaque paire (environnement de typage Γ, terme M) un ensemble fini de types Types(Γ, M), défini par induction sur les termes de la façon suivante :

$$
\begin{array}{lcl}
\text{Types}(\Gamma,\ \text{x}) & \overset{def}{=} & \{\ \text{A} \mid \text{x:A} \in \text{Decon}(\Gamma) \text{ et Decon}(\Gamma) \text{ est axiomatique }\} \\
\text{Types}(\Gamma,\ \text{inl}_B(\text{M})) & \overset{def}{=} & \{\ \text{A+B} \mid \text{A} \in \text{Types}(\Gamma,\ \text{M})\ \} \\
\text{Types}(\Gamma,\text{inr}_A(\text{M})) & \overset{def}{=} & \{\ \text{A+B} \mid \text{B} \in \text{Types}(\Gamma,\ \text{N})\ \} \\
\text{Types}(\Gamma,[\text{M} \mid_{\mathbf{L}} \text{N}]) & \overset{def}{=} & \text{Types}(\Gamma,\ \text{M}) \\
\text{Types}(\Gamma,[\text{M} \mid_{\mathbf{R}} \text{N}]) & \overset{def}{=} & \text{Types}(\Gamma,\ \text{N}) \\
\text{Types}(\Gamma,\lambda\text{P:A.M}) & \overset{def}{=} & \{\ \text{A} \to \text{B} \mid \text{B} \in \text{Types}((\text{P:A},\ \Gamma),\ \text{M})\ \} \\
\text{Types}(\Gamma,<\text{M,N}>) & \overset{def}{=} & \{\ \text{A} \times \text{B} \mid \text{A} \in \text{Types}(\Gamma,\ \text{M}) \text{ et B} \in \text{Types}(\Gamma,\ \text{N})\} \\
\text{Types}(\Gamma,\text{let M be P : A in N}) & \overset{def}{=} & \{\ \text{B} \mid \text{A} \in \text{Types}(\Gamma,\ \text{M}) \text{ et B} \in \text{Types}((\text{P:A},\ \Gamma),\ \text{N})\} \\
\text{Types}(\Gamma,(\lambda\text{P:A.L}) \text{ of N is Q in M}) & \overset{def}{=} & \{\ \text{C} \mid \exists\ \text{B,z},\Gamma',\ \text{Decon}(\Gamma)=\Gamma' \text{ et} \\
 & & \text{A} \to \text{B} \in \text{Types}(\Gamma',\ \lambda\text{P:A.L}) \text{ et} \\
 & & \text{C} \in \text{Types}((\sharp\text{z:A} \to \text{B},\ \Gamma'),\text{z of N is Q in M})\} \\
\text{Types}(\Gamma,\text{z of N is Q:A in M}) & \overset{def}{=} & \{\ \text{C} \mid \exists\ \text{A},\ \Gamma',\ \text{Decon}(\Gamma)=\sharp\text{z:A} \to \text{B},\ \Gamma' \text{ et} \\
 & & \text{A} \in \text{Types}(\Gamma',\text{N}) \text{ et C} \in \text{Types}((\text{Q:B},\Gamma'),\text{M})\ \} \\
\text{Types}(\Gamma,[\text{M} \mid_{\xi} \text{N}]) & \overset{def}{=} & \{\ \text{C} \mid \exists\ \text{P,Q,A,B},\Gamma' \text{ Decon}(\Gamma)=(\text{P} \mid_{\xi} \text{Q})\text{: A + B},\ \Gamma' \text{ et} \\
 & & \text{C} \in \text{Types}((\text{P:A},\Gamma'),\ \text{M}) \cap \text{Types}((\text{Q:B},\Gamma'),\text{N})\ \}
\end{array}
$$

On ajoute les trois cas suivantes à ceux de [BTKP93] :

$$
\begin{array}{lcl}
\text{Types}(\Gamma,\ \text{z[t] is Q:B}[\text{t}/\alpha] \text{ in M}) & \overset{def}{=} & \{C/\exists\Gamma', Decon(\Gamma) = \Delta z : \forall\alpha.B, \Gamma' \text{ et} \\
 & & C \in Types((Q : B[t/\alpha], \Gamma'), M)\} \\
\text{Types}(\Gamma,(\Lambda\alpha\ .\text{L})[\text{t}] \text{ is Q in M}) & \overset{def}{=} & \{C/\exists\Gamma', B, Decon(\Gamma) = \Gamma', \\
 & & \forall\alpha.B \in Types(\Gamma', \Lambda\alpha.L), \\
 & & C \in Types((\Delta z : \forall\alpha.B, \Gamma'), \text{ z[t] is Q:A in M})\} \\
\text{Types}(\Gamma, \Lambda\alpha.M) & \overset{def}{=} & \{\forall\alpha.A/A \in Types(\Gamma, M)\}
\end{array}
$$

La définition est bien fondée, et si Γ est linéaire, alors Types(Γ,M) contient au plus un élément et qu'elle est calculable en temps linéaire(preuve dans [BTKP93], les cas ajoutés ne change rien). La correction de l'algorithme vient du lemme suivant :

Lemme 3.2.5 *$\Gamma \rhd M : A$ est dérivable si et seulement si Types(Γ, M) = A.*

- Montrons que si $Types(\Gamma, M) = A$, alors Γ est dérivable.
  La démonstration se fait par induction sur M et on ne détaille ici que les nouveaux cas:

- M = z[t] is Q:B[t/α] in L,
  si Types(Γ,z[t] is Q:B[t/α] in L) $=\{C\}$ alors par définition de Types :
  $\exists \Gamma', Decon(\Gamma) = \Delta z : \forall \alpha.B, \Gamma'$ et $C \in Types((Q : B[t/\alpha], \Gamma'), L)\}$
  Par hypothèse d'induction, $Q : B[t/\alpha], \Gamma' \rhd M : C$, et on applique la règle $(\forall - left)$. Et on obtient finalement que
  $\Gamma \rhd$ z[t] is $Q : B[t/\alpha]$ in M : C.
- M = $\Lambda\alpha$.M, par hypothèse d'induction et application de la règle ($\forall$-right), la propriété est également vérifiée.

- Montrons maintenant que si Γ est dérivable, alors $Types(\Gamma, M) = A$, par récurrence sur la profondeur de la dérivation :
  - si $\Gamma \rhd M : A$ est un axiome, alors Γ est axiomatique et M est une variable x, donc $Types(\Gamma, x) = A$.
  - si $\Gamma \rhd M : A$ n'est pas un axiome, par le lemme 3.2.3, on sait que $Decon(\Gamma) \rhd M : A$. On procède ensuite par cas selon la dernière règle utilisée, on ne présente que les cas supplémentaires par rapport à [BTKP93]

    * Si la dernière règle utilisée est $(\forall - left)$:

$$\dfrac{n\left\{\begin{array}{c} \vdots \\ Q : A[t/\alpha], \Gamma \rhd M : C \quad (I) \end{array}\right.}{\Delta x : \forall \alpha.A, \Gamma \rhd x[t] \text{ is Q:A}[t/\alpha] \text{ in M:C} \quad (II)}(\forall - left)$$

      D'après l'hypothèse de récurrence, on sait que $Types((Q : A[t/\alpha], \Gamma), M) = C$. D'après la définition de Types, on en déduit directement que
      $Types(\Delta x : \forall \alpha.A, \Gamma, x[t]$ is Q:A$[t/\alpha]$ in M$) = C$.
    * On procède de la même manière pour les règle ($\forall$-right) et (inst).

$\square$

Corollaire 3.2.6 (Décidabilité du typage) *Étant donné* Γ *et* M*, il est décidable en temps linéaire de savoir s'il existe un type* A *tel que* $\Gamma \rhd M : A$ *est dérivable. Si un tel type existe, il est unique et calculable en temps linéaire.*

3.3 Les évaluateurs

3.3.1 Extension des deux évaluateurs

On étend les deux évaluateurs aux nouvelles expressions. Toujours par analogie entre l'abstraction de type et la λ-abstraction, on construit la règle traitant l'instanciation de type de la même manière que la règle existante pour l'application:

$$\frac{\text{let M}[t/\alpha] \text{ be Q:A in N} \downarrow L}{(\Lambda\alpha.M)[t] \text{ is Q:A in N} \downarrow L}$$

Cette règle est la même pour les deux évaluateurs.

Pour les deux évaluateurs, on n'évalue pas l'expression sous une abstraction de type (de la même façon qu'on n'évalue pas sous une λ-abstraction), comme c'est l'usage dans les langages fonctionnels. Ce qui donne la règle suivante:

$$\frac{}{\Lambda\alpha.M \downarrow \Lambda\alpha.M}$$

Reste à étendre l'évaluation de l'opération de filtrage, qui diffère d'un évaluateurs à l'autre :

- Évaluateur paresseux:

$$\frac{L \downarrow \Lambda t.M}{\text{match L on } \Delta z \downarrow [\Lambda t.M/z]}$$

- Évaluateur strict:

$$Match(\Lambda t.M, \Delta z) = [\Lambda t.M/z]$$

3.3.2 Propriétés des évaluateurs

Il s'agit de montrer que les deux évaluateurs sont déterministes, la preuve est une extension de celle de [BTKP93], on ne détaille donc que les cas nouveaux.

L'évaluateur paresseux

On définit une forme canonique pour un évaluateur comme la forme du résultat final d'une évaluation. Par exemple, l'expression let M be Q in N ne peut pas être le résultat d'une évaluation.

$$K ::= < M, M > \mid inl(M) \mid inr(M) \mid \lambda P : A.M \mid \Lambda A.M$$

où P,M et A varient dans les ensembles des motifs, des termes et des types respectivement.

Propriété 3.3.1 *Si $M \Downarrow_l K$, alors K est une forme paresseuse canonique et, si K est une forme paresseuse canonique, alors $K \Downarrow_l K$.*

Preuve: par récurrence sur la profondeur de la définition de M $\Downarrow_l$ K.

□

Théorème 3.3.2 *L'évaluateur paresseux est déterministe.*

Preuve par récurrence sur la profondeur d'une évaluation paresseuse:

- Cas de base : M $\Downarrow_l$ M, alors $M \equiv inl(N)$, $M \equiv inr(N)$, $M \equiv \lambda P : A.M$, $M \equiv < M_1, M_2 >$, ou $M \equiv \Lambda A.N$, et la propriété est triviale.
- Si $M \equiv (\Lambda A.M)[t]$ is Q:B in $L \Downarrow_l K_1$, alors let M[t/A] be Q:B in $L \Downarrow_l K_1$. Or par hypothèse de récurrence, K1 est unique pour let M[t/A] be Q:B in L, donc K_1 est unique pour $(\Lambda A.M)[t]$ is Q:B in L.
- Si M $\equiv$ x[t] is Q:B in L, où x est une variable, alors M n'est pas clos, et l'évaluateur ne s'applique pas.

□

L'évaluateur strict

On définit la forme stricte canonique de la manière suivante:

$$K ::= < K, K > \mid inl(K) \mid inr(K) \mid \lambda P : A.M \mid \Lambda A.M$$

où P,M et A varient dans les ensembles des motifs, des termes et des types respectivement.

Propriété 3.3.3 *Si $M \Downarrow_e K$, alors K est une forme stricte canonique et, si K est une forme stricte canonique, alors $K \Downarrow_e K$.*

Preuve: par récurrence sur la profondeur de la définition de M $\Downarrow_e$ K.

$\square$

Lemme 3.3.4 *Le résultat de Match(M,P) est unique.*

Preuve : par récurrence sur la structure du motif P.

$\square$

Théorème 3.3.5 *L'évaluateur strict est déterministe.*

Preuve : par récurrence sur la profondeur d'une évaluation stricte:

- Cas de base : M $\Downarrow_e$ M, alors $M \equiv \lambda P : A.M$, ou $M \equiv \Lambda A.N$, et la propriété est triviale.

- Cas supplémentaire par rapport à la preuve de [BTKP93] : Si $M \equiv (\Lambda A.M)[t]$ is Q:B in $L \Downarrow_e K_1$, alors let M[t/A] be Q:B in $L \Downarrow_e K_1$. Or par hypothèse de récurrence, K1 est unique pour let M[t/A] be Q:B in L, donc K_1 est unique pour $(\Lambda A.M)[t]$ is Q:B in L.

- Si M $\equiv$ x[t] is Q:B in L, où x est une variable, alors M n'est pas clos, et l'évaluateur ne s'applique pas.

$\square$

Les deux évaluateurs préservent les types, et sont convergents. Ces propriétés découlent de celles du système de réduction non déterministe défini ci-dessous car les évaluateurs sont simplement des stratégies déterministes pour ce système. On conclut que notre calcul possède les mêmes propriétés que le λ-calcul typé, le λ-calcul polymorphe de Girard[Gir72], etc.

3.4 Le système de réduction

3.4.1 Extension du système

Pour traiter l'instanciation de type, on ajoute l'équation suivante(On remarque encore l'analogie avec la β-réduction):

$$(\Lambda\alpha.L)[t] \text{ is Q:B in N} = \text{let L}[t/\alpha] \text{ be Q:B in N}$$

On étend donc aussi la fermeture sous les contexte: Si M $\Rightarrow$ M', alors:

$$\begin{array}{lcl} \text{M[t] is Q:A in N} & \Rightarrow & \text{M'[t] of Q:A in N} \\ \text{N[t] is Q:A in M} & \Rightarrow & \text{N[t] of Q:A in M'} \end{array}$$

Voici donc le système de réduction complet:

$$\frac{Match(M, P) = \sigma}{\text{let M be P in N} = N[\sigma]}$$

$$\frac{Match(M, P) = \sigma}{(\lambda P : A.L) \text{ of M is Q:B in N} = \text{let L}[\sigma] \text{ be Q:B in N}}$$

$$[M|_L N] = M \qquad [M|_R N] = N$$

$$\text{let M be } P_1@P_2 \text{ in N} = \text{let} < M, M > \text{be} < P_1, P_2 > \text{in N}$$

$$(\lambda P_1@P_2.L) \text{ of M is Q:A in N} = (\lambda < P_1, P_2 > .L) \text{ of} < M, M > \text{is Q:A in N}$$

$$(\Lambda\alpha.L)[t] \text{ is Q:B in N} = \text{let L}[t/\alpha] \text{ be Q:B in N}$$

3.4.2 Propriétés du système

Stabilité par substitution

Montrons que $\Rightarrow$ est stable par substitution. Dans la de'monstration, on ne présentera que les cas engendrés par les motifs ou les termes ajoutés au système initial.

Remarque 3.4.1 *Soient L, M et N trois termes et soient P et Q deux motifs tel que Var(P)* $\cap$ *Var(Q)* $= \emptyset$, *Var(P)* $\cap$ *FV(N)* $= \emptyset$, *Match(N,Q)* $\neq$ **fail** *et Match(M,P)* $\neq \emptyset$. *Alors:*

$L[Match(N,Q)][Match(M[Match(N,Q)],P)] = L[Match(M,P)][Match(N,Q)]$

Propriété 3.4.2 *Si Match(M,P)=*θ *et M* $\Rightarrow$ *M', alors Match(M',P)=*θ' *et pour toute variable usuelle* x *dans Dom(*θ*) on a* $\theta(x) \Rightarrow^* \theta'(x)$. *On remarque qu'il existe au moins une variable usuelle* x *dans Dom(*θ*) telle que* $\theta(x) \Rightarrow^+ \theta'(x)$.

Preuve: Par induction sur le motif P.

- P=Δz, alors M = ΛA.N, et M'=ΛA.N', avec $N \Rightarrow N'$.
  Par définition de Match, on a :
  Match(ΛA.N, Δz)=[ΛA.N/z]
  Match(ΛA.N', Δz)=[ΛA.N'/z]
  Or ΛA.N ⇒ ΛA.N'. Donc la propriété est vraie.

□

Propriété 3.4.3 *Si N* $\Rightarrow$ *N' et la substitution Match(N,P) est définie, alors le terme M[Match(N,P)] se réduit en M[Match(N',P)].*
Même plus, si $\emptyset \neq$ *Var(P)* $\subseteq$ *FV(M), alors M[Match(N,P)]* $\Rightarrow^+$ *M[Match(N',P)].*

Preuve: Soit Match(N,P) = θ. Par la propriété 3.6.2 il existe une substitution θ' telle que Match(N',P)= θ'. On montre la propriété par récurrence sur la structure de M.

- M=ΛA.L, par hypothèse de récurrence : L[Match(N,P)] ⇒ L[Match(N',P)]. Comme ⇒ est fermée sous tous les contextes, on a aussi : ΛA.L[Match(N,P)] ⇒ ΛA.L[Match(N',P)]. La propriété est donc vérifiée.
- M=L[t] is Q:A in K. Par hypothèse de récurrence :
  L[Match(N,P)] ⇒ L[Match(N',P)], et K[Match(N,P)] ⇒ K[Match(N',P)].
  Et donc, par la définition de ⇒ (fermeture sous tous contextes), la propriété est vrai:
  (L[t] is Q:A in K)[Match(N,P)] ⇒ (L[t] is Q:A in K)[Match(N',P)].

$\square$

Propriété 3.4.4 *Soit M $\Rightarrow$ M' et Match(N,P) $\neq$* **fail**. *Alors, M[Match(N,P)] $\Rightarrow$ M'[Match(N,P)].*

Preuve: Soit Match(N,P) = θ. On montre la propriété par récurrence sur la structure de M.

- Si M = ($\Lambda\alpha$.J)[A] is Q:t in L $\Rightarrow$ M'= let J[A/α] be Q:t in L (le cas où la réduction est interne est déjà traiter dans le système initial)
  alors (($\Lambda\alpha$.J)[A] is Q:t in L)[θ] = ($\Lambda\alpha$.J)[θ][A] is Q:t in L[θ] $\Rightarrow$
  let J[θ][A/α] be Q:t in L[θ] = (let J[A/α] be Q:t in L)[θ]
  Donc :
  (($\Lambda\alpha$.J)[A] is Q:t in L)[Match(N,P)] = (let J[A/α] be Q:t in L)[Match(N,P)]
  et la propriété est vérifiée.

$\square$

Corollaire 3.4.5 (Stabilité par substitution) *Si M $\Rightarrow$ M', et N $\Rightarrow$ N', alors M[Match(N,P)] $\Rightarrow^*$ M'[Match(N,P)].*

Préservation des types

On commence par montrer deux lemmes techniques:

Lemme 3.4.6 *Si $P : B, \Gamma \rhd M : A$ et Var(P) $\cap$ FV(M) = $\emptyset$, alors $\Gamma \rhd M : A$.*

Preuve: Par récurrence sur la profondeur de la dérivation de $P : B, \Gamma \rhd M : A$. Cas supplémentaire par rapport à [BTKP93] :

- Si la dernière règle est ($\forall$-left):

$$\frac{P : B, Q : A[t/\alpha], \Omega \rhd N : C}{P : B, \Delta x : \forall\alpha.A, \Omega \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in N:C}}(\forall - left)$$

  Si Var(P) $\cap$ FV(x[t] is Q:A [t/α] in N:C) = $\emptyset$, alors Var(P) $\cap$ FV(N) = $\emptyset$.
  Par hypothèse de récurrence:

$Q : A[t/\alpha], \Omega \rhd N : C$

et on applique la règle (∀-left):

$$\frac{}{\Delta x : \forall x.A, \Omega \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in N:C}} \forall - left$$

De même pour les deux autres règles (∀-right) et (inst).

□

Lemme 3.4.7 (lemme de substitution) *Soient* $P : B, \Gamma \rhd M : A$, $\Gamma \rhd N : B$ *et Match(N,P)=* θ*. Alors* $\Gamma \rhd M[\theta] : A$.

Preuve :Par récurrence sur la profondeur de la dérivation $P : B, \Gamma \rhd M : A$, et par cas selon la dernière règle utilisées.

- (∀-left):

$$\frac{Q : A[t/\alpha], \Gamma \rhd L : C}{\Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in L:C}} (\forall - left)$$

Soit $M =$ x[t] is Q:A$[t/\alpha]$ in L.

Si P=Δx, alors par définition de Match, N=$\Lambda\alpha$.K,
et Match(N,P)= [$\Lambda\alpha$.K/x].
Donc M[θ]=$(\Lambda\alpha.K)[t]$ is Q:A$[t/\alpha]$ in M:C).
Par hypothèse, on a :$\Gamma \rhd \Lambda\alpha.K : \forall\alpha.A$,
de plus : $\Delta x : \forall\alpha.A, \Gamma \rhd$ x[t] is Q:A$[t/\alpha]$ in L:C.
On peut donc appliquer la règle (inst) :

$$\frac{\Gamma \rhd \Lambda\alpha.K : \forall\alpha.A \qquad \Delta x : \forall\alpha.A, \Gamma \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in L:C}}{\Gamma \rhd (\Lambda\alpha.K)[t] \text{ is Q:A}[t/\alpha] \text{ in M:C)}} (inst)$$

Et la propriété est vérifiée.

Si P $\neq$ Δx, on a:

$$\frac{P : B, Q : A[t/\alpha], \Omega \rhd L : C}{P : B, \Delta x : \forall\alpha.A, \Omega \rhd \text{x[t] is Q:A}[t/\alpha] \text{ in L:C}} (\forall - left)$$

Par hypothèse de récurrence on a :$Q : A[t/\alpha], \Omega \rhd L[\theta] : C$.
On peut alors appliquer la règle ($\forall$-left):

$$\frac{Q : A[t/\alpha], \Omega \rhd L[\theta] : C}{\Delta x : \forall\alpha.A, \Omega \rhd (\text{x[t] is Q:A}[t/\alpha] \text{ in L})[\theta] : C}(\forall - left)$$

Et la propriété est vérifiée.

- ($\forall$-right):

$$\frac{P : B, \Gamma \rhd M : A}{P : B, \Gamma \rhd \Lambda\alpha.M : \forall\alpha.A} \qquad (\forall - right)$$

α non libre dans $\{ \Gamma, P : B\}$.
Par hypothèse de récurrence: $\Gamma \rhd M[\theta] : A$
Donc on peut appliquer la règle ($\forall$-right), car α non libre dans Γ:

$$\frac{\Gamma \rhd M[\theta] : A}{\Gamma \rhd \Lambda\alpha.M[\theta] : \forall\alpha.A}(\forall - right)$$

Et la propriété est vérifiée.

- (inst):

$$\frac{P : B, \Gamma \rhd \Lambda\alpha.L : \forall\alpha.A \quad P : B, \Delta z : \forall\alpha.A, \Gamma \rhd \text{z[t] is Q:A}[t/\alpha] \text{ in M:C}}{P : B, \Gamma \rhd (\Lambda\alpha.L)[t]\text{is Q} : A[t/\alpha]\text{in M:C}}(inst)$$

Par hypothèse de récurrence : $\Gamma \rhd \Lambda\alpha.L[\theta] : \forall\alpha.A$
et: $\Delta z : \forall\alpha.A, \Gamma \rhd \text{z[t] is Q:A}[t/\alpha] \text{ in M}[\theta] : C$
Et on applique (inst):

$$\frac{\Gamma \rhd \Lambda\alpha.L[\theta] : \forall\alpha.A \qquad \Delta z : \forall\alpha.A, \Gamma \rhd \text{z[t] is Q:A}[t/\alpha] \text{ in M}[\theta] : C}{\Gamma \rhd (\Lambda\alpha.L)[\theta][t]\text{is Q} : A[t/\alpha]\text{in M}[\theta] : C}(inst)$$

Or :

$$(\Lambda\alpha.L)[\theta][t]\text{is Q} : A[t/\alpha]\text{in M}[\theta] = ((\Lambda\alpha.L)[t]\text{is Q} : A[t/\alpha]\text{in M})[\theta]$$

Donc la propriété est vérifiée.

□

On a besoin ici d'un lemme supplémentaire sur les substitutions de variables de type:

Lemme 3.4.8 *Si* $\Gamma \rhd L : B$*, alors* $\Gamma[t/\alpha] \rhd L[t/\alpha] : B[t/\alpha]$

Preuve : Par récurrence sur la profondeur de la dérivation de $\Gamma \rhd L : B$.

- $\Gamma \rhd L : B$ est un axiome de la forme :
  $x_1 : B_1, \ldots, x_n : B_n \rhd x_i : B_i$, donc :
  $\Gamma[t/\alpha] = x_1 : B_1[t/\alpha], \ldots, x_n : B_n[t/\alpha]$.
  D'où:
  $\Gamma[t/\alpha] \rhd x_i : B_i[t/\alpha]$
  Or $x_i = x_i[t/\alpha]$, donc finalement:
  $\Gamma[t/\alpha] \rhd x_i[t/\alpha] : B[t/\alpha]$
  et la propriété est vérifiée.

- Par cas selon la dernière règle utilisée, on ne montre ici que deux cas:

  – ($\forall$-right):

  $$\frac{\Gamma \rhd M : A}{\Gamma \rhd \Lambda\alpha'.M : \forall\alpha'.A}(\forall\text{-}right)$$

  où α' n'est pas libre dans Γ. On observe deux cas:

  * Si $\alpha' = \alpha$ alors α n'est pas libre dans Γ et donc $\Gamma[\mathrm{t}/\alpha] = \Gamma$, et comme α n'est pas libre non plus dans $\Lambda\alpha'$.M et $\forall\alpha'$.A, on a aussi $\Lambda\alpha'$.M = $\Lambda\alpha'$.M[t/α] et $\forall\alpha'$.A = $\forall\alpha'$.A[t/α]. et donc on a $\Gamma \rhd \Lambda\alpha'.M : \forall\alpha'.A$.
  * Si $\alpha' \neq \alpha$, alors par hypothèse de récurrence et par application de la règle ($\forall$-right), la propriété est aussi vérifiée.

  Par hypothèse de récurrence,

  – ($\forall$-left):

  $$\frac{Q : A[t'/\alpha'], \Gamma \rhd M : C}{\Delta x : \forall\alpha'.A, \Gamma \rhd \mathrm{x[t'] \ is \ Q{:}A}[t'/\alpha'] \mathrm{\ in\ M{:}C}}(\forall - left)$$

Par hypothèse de récurrence:
$Q : A[t'/\alpha'][t/\alpha], \Gamma[t/\alpha] \rhd M[t/\alpha] : C[t/\alpha]$.

Or on peut supposer sans perte de généralité que :
$\alpha' \notin Var(t)$, donc, d'après la remarque 3.4.1: $A[t'/\alpha'][t/\alpha] = A[t/\alpha][t'[t/\alpha]/\alpha']$
On peut donc appliquer la règle ($\forall$ -left):

$$\frac{Q : A[t/\alpha][t'[t/\alpha]/\alpha'], \Gamma[t/\alpha] \rhd M[t/\alpha] : C[t/\alpha]}{\Delta x : \forall\alpha'.A[t/\alpha], \Gamma[t/\alpha] \rhd x[t'[t/\alpha]] \text{ is Q:A}[t'[t/\alpha]/\alpha'] \text{ in } M[t/\alpha] : C[t/\alpha]}(\forall - left)$$

Or :
$x[t'[t/\alpha]]$ is Q:A$[t'[t/\alpha]/\alpha']$ in $M[t/\alpha] =$ (x[t'] is Q:A [t'/α'] in M:C)[t/α]
Et la propriété est donc vérifiée.

$\square$

Lemme 3.4.9 (Préservation de type) *Si* $\Gamma \rhd M : A$ *et* $M \Rightarrow M'$*, alors* $\Gamma \rhd M' : A$.

Preuve: Par récurrence sur la profondeur de la dérivation. On ne montre que les cas qui correspondent aux réductions externes, les autres cas étant traités par récurrence sur la structure du termes M.
Cas supplémentaire par rapport à la preuve de [BTKP93]:

($\Lambda\alpha$.L)[t] is Q:B[t/α] in N $\rightarrow$ let L[t/α] be Q:B[t/α] in N
M = ($\Lambda\alpha$.L)[t] is Q:B[t/α] in N

On a : $\Gamma \rhd M : A$ donc par la propriété de commutation, il existe une dérivation qui se termine par :

$$\frac{\Gamma \rhd \Lambda\alpha.L : \forall\alpha.B \qquad \Delta z : \forall\alpha.B, \Gamma \rhd z[t] \text{ is Q} : B[t/\alpha] \text{ in N:A}}{\Gamma \rhd (\Lambda\alpha.L)[t] \text{ is Q} : B[t/\alpha] \text{ in N:A}}(inst)$$

Donc, toujours par la propriété de commutation, il existe deux dérivations qui se terminent par :

$$\frac{\Gamma \rhd L : B}{\Gamma \rhd \Lambda\alpha.L : \forall\alpha.B}(\forall - right)$$

et:

$$\frac{Q : B[t/\alpha], \Gamma \rhd N : A}{\Delta z : \forall\alpha.B, \Gamma \rhd z[t] \text{ is Q} : B[t/\alpha] \text{ in N:A}}(\forall - left)$$

Comme $\Gamma \rhd L : B$ et α non libre dans Γ, on a, par le lemme 3.4.8: $\Gamma \rhd L[t/\alpha] : B[t/\alpha]$
On peut donc appliquer la règle (let):

$$\frac{\Gamma \rhd L[t/\alpha] : B[t/\alpha] \qquad Q : B[t/\alpha], \Gamma \rhd N : A}{\Gamma \rhd \text{let L}[t/\alpha] \text{ be Q} : B[t/\alpha] \text{ in N:A}}(let)$$

Et la propriété est vérifiée.

$\square$

Confluence

Montrons que la relation $\Rightarrow$ est confluente sur les termes purs. La preuve est une extension de celle de [BTKP93], qui utilise la méthode de Tait et Martin-Löf. Il s'agit de définir une réduction parallèle $\blacktriangleright$ dont la fermeture reflexive et transitive soit égale à celle de $\Rightarrow$, et vérifiant la propriété du diamant définie ci-dessous. On pourra alors en déduire par la propriété 3.4.11 donnée ci-dessous, que $\blacktriangleright$ est confluente et donc $\Rightarrow$ aussi.

Définition 3.4.10 (propriété du diamant) *La relation $\rightarrow$ a la propriété du diamant si pour tout a, b, $c \in A$ tels que $a \rightarrow b$ et $a \rightarrow c$, il existe $d \in A$ tel que $b \rightarrow d$ et $c \rightarrow d$.*

Propriété 3.4.11 *Si $\rightarrow$ possède la propriété du diamant, alors $\rightarrow$ est confluente.*

La relation de réduction parallèle de [BTKP93] est donnée à la page suivante:

- M $\blacktriangleright$ M
- Si M $\blacktriangleright$ M', alors
  - [M $|_{\mathbf{L}}$ N] $\blacktriangleright$ M'
  - [N $|_{\mathbf{R}}$ M] $\blacktriangleright$ M'
  - λP.M $\blacktriangleright$ λP.M'
  - inl(M) $\blacktriangleright$ inl(M')
  - inr(M) $\blacktriangleright$ inr(M')
- Si M $\blacktriangleright$ M' et N $\blacktriangleright$ N', alors
  - [M $|_{T}$ N] $\blacktriangleright$ [M' $|_{T}$ N']
  - <M,N> $\blacktriangleright$ <M',N'>
  - let M be P:A in N $\blacktriangleright$ let M' be P:A in N'
  - Si M $\blacktriangleright$ M", alors let M be P_1@P_2 in N $\blacktriangleright$ let <M',M"> be <P_1,P_2 > in N'.
  - Si Match(M,P) $\neq$ **fail**, alors let M be P:A in N $\blacktriangleright$ N'[Match(M',P)]
- Si M $\blacktriangleright$ M', N $\blacktriangleright$ N' et L $\blacktriangleright$ L', alors
  - M of N is Q:A in L $\blacktriangleright$ M' of N' is Q:A in L'
  - Si N $\blacktriangleright$ N", alors (λP_1@P_2.M) of N is Q:A in L $\blacktriangleright$ (λ <P_1,P_2 >.M) of <N',N"> is Q:A in L('?))
  - Si Match(N,P) $\neq$ **fail**, alors (λ P.M) of N is Q:A in L $\blacktriangleright$ let M'[Match(N',P)] be Q:A in L'.

On ajoute les cas suivants:

- Si M $\blacktriangleright$ M', alors
  - $\Lambda\alpha$.M $\blacktriangleright$ $\Lambda\alpha$.M'
- Si M $\blacktriangleright$ M' et N $\blacktriangleright$ N'
  - M[t] is Q:A in N $\blacktriangleright$ M'[t] is Q:A in N'
  - ($\Lambda\alpha$.M)[t] is Q:A in N $\blacktriangleright$ let M'[t/α] is Q:A in N'

Montrons tout d'abord que les deux relation $\blacktriangleright$ et $\Rightarrow$ ont des fermetures réflexives et transitives $\blacktriangleright^*$ et $\Rightarrow^*$ identiques.

Pour cela on commence par montrer que $\blacktriangleright$ est stable par substitution.

Lemme 3.4.12 *Si M $\blacktriangleright$ M', N $\blacktriangleright$ N' et Match(N,P) est défini, alors M[Match(N,P)] $\blacktriangleright$ M'[Match(N',P)].*

Preuve: Par récurrence sur la structure de M, on procède de la même manière que pour la relation $\Rightarrow$ (voir propriétés 3.4.2 et 3.4.3)

$\square$

Lemme 3.4.13 *Les fermetures réflexives et transitives de $\blacktriangleright$ et $\Rightarrow$ sont des relations identiques.*

Preuve: On montre d'abord l'inclusion $\blacktriangleright^* \subseteq \Rightarrow^*$ puis l'inclusion $\Rightarrow^* \subseteq \blacktriangleright^*$.

- Pour montrer $\blacktriangleright^* \subseteq \Rightarrow^*$, il suffit de montrer $\blacktriangleright \subseteq \Rightarrow^*$, par récurrence sur la définition de $\blacktriangleright$. On ne traite que les cas supplémentaires par rapport à [BTKP93].
  - Si M $\blacktriangleright$ M' implique $\Lambda\alpha$.M $\blacktriangleright$ $\Lambda\alpha$.M', alors par hypothèse de récurrence M $\Rightarrow^*$ M' et comme $\Rightarrow$ est fermée sous tout contexte, $\Lambda\alpha$.M$\Rightarrow^*$ $\Lambda\alpha$.M'.
  - Si M $\blacktriangleright$ M' et N $\blacktriangleright$ N' impliquent M[t] is Q:A in N $\blacktriangleright$ M'[t] is Q:A in N', alors par hypothèse de récurrence M $\Rightarrow^*$ M' et N $\Rightarrow^*$ N', et comme $\Rightarrow$ est fermée sous tout contexte, M[t] is Q:A in N $\Rightarrow^*$ M'[t] is Q:A in N'.
  - De la même manière, si M $\blacktriangleright$ M' et N $\blacktriangleright$ N' impliquent ($\Lambda\alpha$.M)[t] is Q:A in N $\blacktriangleright$ let M'[t/α] is Q:A in N', la propriété se montre par l'hypothèse de récurrence et le fait que $\Rightarrow$ est fermée sous tout contexte.
- Pour montrer l'inclusion $\Rightarrow^* \subseteq \blacktriangleright^*$, il suffit de montrer $\Rightarrow \subseteq \blacktriangleright$, par récurrence sur la structure des termes, puis par cas selon la définition de $\Rightarrow$.

$\square$

Montrons enfin que la propriété du diamant est vérifiée par $\blacktriangleright$.

Propriété 3.4.14 *La relation $\blacktriangleright$ satisfait la propriété du diamant. C'est à dire que pour chaque terme L tel que $L \blacktriangleright L_1$ et $L \blacktriangleright L_2$, il existe un terme L_3 tel que $L_1 \blacktriangleright L_3$ et $L_2 \blacktriangleright L_3$.*

Preuve:Par récurrence sur la structure de L. On montre les cas supplémentaires par rapport à [BTKP93]:

- Si $\mathrm{L} = \Lambda\alpha.\mathrm{M}$, alors:
  $$\begin{array}{ccc} \mathrm{L} & \blacktriangleright & \Lambda\alpha.\mathrm{M}_1 \\ \blacktriangledown & & \\ \Lambda\alpha.\mathrm{M}_2 & & \end{array}$$
  avec $\mathrm{M} \blacktriangleright \mathrm{M}_1$ et $\mathrm{M} \blacktriangleright \mathrm{M}_2$. Par hypothèse de récurrence, il existe M_3 tel que $\mathrm{M}_1 \blacktriangleright \mathrm{M}_3$ et $\mathrm{M}_2 \blacktriangleright \mathrm{M}_3$. Comme $\blacktriangleright$ est fermée sous tout contexte, on a aussi :
  $$\begin{array}{ccc} & & \Lambda\alpha.\mathrm{M}_1 \\ & & \blacktriangledown \\ \Lambda\alpha.\mathrm{M}_2 & \blacktriangleright & \Lambda\alpha.\mathrm{M}_3 \end{array}$$
  Donc la propriété est vérifiée.

- Si L = M[t] is P:A in N, on procède de la même manière que le cas précédent.

- Si L = ($\Lambda\alpha$.M)[t] is P:A in N, on procède comme ci-dessus.

$\square$

$\blacktriangleright$ vérifie donc bien la propriété du diamant, elle est donc confluente par la propriété 3.4.11. Comme $\blacktriangleright$ et $\Rightarrow$ ont la même fermeture reflexive et transitive, $\Rightarrow$ est aussi confluente.

Théorème 3.4.15 (Confluence) $\Rightarrow$ *est confluente.*

4 Conclusion

Nous avons étendu le λ-calcul avec motifs au polymorphisme en utilisant les règles du second ordre du calcul des séquents de Gentzen et en nous inspirant de l'isomorphisme de Curry-Howard. Ce modèle de calcul se rapproche ainsi plus des langages fonctionnels actuels, et offre un traitement rigoureux des motifs et du polymorphisme. Entre autre la détection de définitions de fonctions par cas non exhaustives ou redondantes est possible sans recourir aux solutions *ad-hoc* de certains langages.

L'interaction motifs-termes semble être une interprétation pertinente de la symétrie entre les règles gauches et les règles droite du calcul de Gentzen. Le passage au second ordre se fait naturellement, ce qui montre l'importance et l'envergure de l'isomorphisme de Curry-Howard. Cette correspondance nous amène à nous interroger sur l'impact des propriétés du calcul de Gentzen sur notre calcul, par exemple dans [BTKP93] les questions suivantes émergent : quelle est l'interprétation de l'élimination de la règle de coupure dans notre calcul? Les relations entre le calcul de Gentzen et la déduction naturelle sont elles transposables entre notre calcul et le λ-calcul typé?

En revanche, la syntaxe du calcul reste lourde et est principalement la cause de l'existence des règles de typage étendues. Un objectif peut être de définir un calcul semblable mais dans lequel la règle de coupure serait remplacée par des règles interprétant l'élimination de coupure, et dont la syntaxe soit plus uniforme. Pour ce deuxième point, il semble intéressant d'introduire les substitutions explicites [ACCL90] dans la syntaxe pour obtenir un système de réduction plus régulier. Par exemple si on introduit dans la syntaxe la notation [M match P] pour représenter la substitution explicite permettant le filtrage du motif P par le terme M, on pourrait obtenir des règles de réduction de la forme:

$$\begin{array}{lcl}
\text{let M be P in N} & \rightarrow & \text{N[M match P]} \\
\text{x[N match x]} & \rightarrow & \text{N} \\
\text{y[N match x]} & \rightarrow & \text{x} \\
\text{[inl(M) match (P }|_\xi\text{ Q)]} & \rightarrow & \text{[L match }\xi\text{][M match P]}
\end{array}$$

On remarque alors que le traitement des substitutions se fait dans le calcul lui même, ce qui présente l'avantage d'être plus simple et plus proche des méchanismes des langages fonctionnels.

Références

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[ACCL90] Martin Abadi, Luca Cardelli, Pierre Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. Technical Report 54, DEC Systems Research Center, 1990.

[BTKP93] Val Breazu-Tannen, Delia Kesner, and Laurence Puel. A typed pattern calculus. Technical Report MS-CIS-93-40, Logic & Computation 62, University of Pennsylvania, 1993.

[Gal93] Jean Gallier. Constructive logics part i : A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:249–339, 1993.

[Gir72] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieure*. Thèse de doctorat d'état, Université Paris VII, 1972.

[HPJe92] Paul Hudak, Simon Peyton-Jones, and Philip Wadler (editors). Report on the programming language haskell, a non-strict, purely functional language (version 1.2). Sigplan Notices, 1992.

[Jon87] Simon Peyton Jones. *The implementation of functional programming languages*. Prentice-Hall, 1987.

[Kah87] Gilles Kahn. Natural semantics. In *Proceedings Symposium Theoretical Aspects of Computer Science*, pages 22–39. Springer-Verlag, 1987. Lecture Notes in Computer Science 247.

[Kes93] Delia Kesner. *La définition de fonctions par cas à l'aide de motifs dans des langages applicatifs*. Thèse de doctorat, Université de Paris XI, Orsay, december 1993.

[Laf89] Yves Lafont. Functional programming and linear logic. Lecture notes for the Summer School on Functional Programming and Constructive Logic, 1989. Glasgow.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *The definition of Standard ML*. MIT Press, 1990.

[Oos90] Vincent Van Oostrom. Lambda calculus with patterns. Technical Report IR 228, Vrije Universieit, Amsterdam, 1990.

[Tiu96] Jerzy Tiuryn. A sequent calculus for subtyping polymorphic types. to appear in ...?, 1996.

[Tur85] D. A. Turner. Miranda: A non-strict functional language with polymorphic types. In Jean Pierre Jouannaud, editor, *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, pages 1–16. Springer-Verlag, 1985. Lecture Notes in Computer Science 201.