Languages et Modèles pour la Formalisation et l'Automation de la Mathématique et de l'Informatique

Fairouz Kamareddine (Université de Heriot-Watt, Edimbourgh)

11 Mai 2009

Chambéry, le 11 Mai 2009

Le language de Mathématique

D'habitude, le mathématicien ignore la logique formelle. Les mathématiciens écrivent la mathématique avec un language (style) commun qu'on appelle le CML. Les avantages de CML:

- *Expressivité:* On peut exprimer toutes genres de notions.
- *Acceptabilité:* CML est accepté par la plupart des mathématicien.
- *traditionalité:* CML existe depuis très longtemps et a été raffiné avec le temps.
- *Universalité:* CML est utilisé partout dans le monde.
- *Flexibilité:* Avec CML on peut décrire plusieures branches de mathématiques.

Les désavantages de CML:

- *Informel et ambigu:* CML est basé sur le language naturelle.
- *Incomplet:* De choses implicites, l'écrivain compte sur l'intuition du lecteur.
- *Pas facile à automatiser* CML
- Au 19ème ciècle, les problèmes en Analyse créaient le besoin d'un style *précis*.
- Plusieures de ces problèmes ont été resolu par le travail de Cauchy (par example par sa définition précise de convergence dans son Cours d'Analyse).
- Les systèmes des nombres sont devenus plus précis avec la définition exacte des nombres réel de Dedekind.
- Cantor commençait la formalisation de la théorie des ensembles et contribuait à la théorie des nombres.

Logique, fonctions, λ-calcul et theories des Types

- Frege était frustré par les informalités de $\textsc{Cml}$.
- *La definition Genérale de la fonction* était la clef de sa *formalisation de la logique* (1879).
- *L'application d'une fonction à elle-même* $f(x) = \neg x(x)$ était la clef du *paradox de Russell* (1902). Voir [Kamareddine et al., 2002].
- Pour éliminer les paradox, Russell controllait l'application d'une function à un argument par la *theorie des types*.
- Russell (1908) donnait $\textsc{rtt}$, la 1ère theorie des types. Russell and Whitehead utilisaient $\textsc{rtt}$ dans *Principia Mathematica* (1910–1912).
- Les *types* ont *toujours existés* en mathématiques, mais pas explicitment avant 1879. Euclid a éliminé les situations *impossibles* (e.g., deux points parallels) par l'utilisations des classes/*types*.

- *theorie simple des types* (STT): Ramsey (1926), Hilbert et Ackermann (1928).
- En 1928, Church voulait décrire un language de *fonctions et de logique*:
  $\Lambda ::= \mathcal{V}|(\Lambda\Lambda)|\lambda\mathcal{V}.\Lambda|\neg\mathcal{V}|\forall\mathcal{V}.\Lambda(\mathcal{V})$
- La combinaison des 2 (fonctions et logique) etait paradoxique.
  Le problème n'est pas avec $\neg$, mais avec $\forall$.
- En 1932, il elimine la logique et son λ-calcul devient un calcul des fonctions.
- En 1940, il rajoute la logique mais aussi les types pour eliminer les paradoxes.
  Le λ-calcul simplement typé $\lambda\!\!\rightarrow$ = λ-calcul + STT (1940).

Mes premières recherches au λ-calcul

- On sait bien que $|A| < |A \mapsto A|$ et même $|A| < |A \mapsto \{0,1\}|$ (Cantor).
- En 1969, Dana Scott voulait démontrer la non-existence des modèles du λ-calcul.
- Contrairement à ce qu'il voulait, il construit un modèle D du λ-calcul où $D \sim (D \longrightarrow D)$, $(D \longrightarrow D)$ est l'ensemble des fonctions continues de D à D et D est le point fixe d'un processu continu de la construction.
- Comme les domains de Scott ne permettent pas la logique (ils sont des modèles du système des fonctions, le λ calcul), Peter Aczel en 1980 introduit les "Frege structures" qui sont des modèles et des fonctions et de la logique. Frege structures sont aussi construit par le theorème du point fixe.
- Ma thèse de doctorat "Semantics in a Frege structure (1988)" proposait un λ-calcul basé sur Frege structure, étudiait des propriétés comme l'adequacy et la complétude, et l'appliquait à la semantique des phrases paradoxiques et "self-referentiels".

Le pas suivant

- En 1987, j'ai commencé mon travail à Glasgow dans un département d'informatique. Alors, en apprenant les differents languages et styles de programmation je me posait la question si les théories des λ-calculs (avec/sans logique/typage) sont assez bons pour les languages d'informatique (même si on peut exactement exprimer dans le λ-calcul les fonctions computables).

- J'avais aussi l'interet de la formalisation et de l'automatisation de la mathématique.

- Répetons l'essence du λ-calcul:
  Syntax: $\Lambda ::= \mathcal{V}|(\Lambda\Lambda)|\lambda\mathcal{V}.\Lambda$.
  Règle: $(\lambda x.A)B \rightarrow_\beta A[x := B]$

Le lambda Calcul à la de Bruijn (item notation) [Kamareddine and Nederpelt, 1995, 1996]

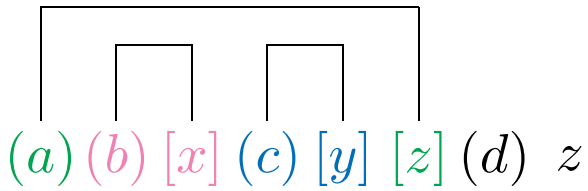
$'$: Notation classique	$\mapsto$	Notation de de Bruijn
x	$\mapsto$	x
$\lambda x.B$	$\mapsto$	$[x]B'$
AB	$\mapsto$	$(B')A'$

Example: $(\lambda x.\lambda y.xy)z \mapsto (z)[x][y](y)x$

- Dans le *train* $(z)[x][y](y)$, les *wagons* sont (z), $[x]$, $[y]$ and (y).
- Le dernier x dans $(z)[x][y](y)x$ est le *coeur* du terme.
- Le *wagon d'application* (z) et le *wagon d'abstraction* $[x]$ se trouvent l'un *à coté* de l'autre.
- La règle β $\quad (\lambda x.A)B \to_\beta A[x := B]$ devient: $(B)[x]A \to_\beta [x := B]A$

Réduction dans la notation de de Bruijn

Notation classique	Notation de de Bruijn
$\underline{((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a}$	$(a)\underline{(b)[x]}(c)[y][z](d)z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{((\lambda_y.\lambda_z.zd)c)a}$	$(a)\underline{(c)[y]}[z](d)z$
$\downarrow_\beta$	$\downarrow_\beta$
$\underline{(\lambda_z.zd)a}$	$\underline{(a)[z]}(d)z$
$\downarrow_\beta$	$\downarrow_\beta$
ad	$(d)a$



Chaque wagon a un *partenaire* sauf (d) qui est *célibataire*.
Les crochets de $((\lambda_x.(\lambda_y.\lambda_z.\ -\ -)c)b)a)$, sont '$[_1\ [_2\ [_3\ ]_2\ ]_1\ ]_3$', où '$[_i$' and '$]_i$' vont ensemble.
Les crochets de $(a)(b)[x](c)[y][z]$ sont plus simple: $[[\,][\,]]$.

Réductions generalisées

- $((\lambda_x.(\underline{\lambda_y.\lambda_z.zd)c})b)a \rightarrow_\beta ((\lambda_x.\{\lambda_z.zd\}[y := c])b)a$

  $(a)(b)[x](c)[y][z](d)z \rightarrow_\beta (a)(b)[x][y := c][z](d)z$

- $(\underline{(\lambda_x.(\lambda_y.\lambda_z.zd)c)b})a \rightarrow_\beta \{(\lambda_y.\lambda_z.zd)c\}[x := b]a$

  $(a)(b)[x](c)[y][z](d)z \rightarrow_\beta (a)[x:=b](c)[y][z](d)z$

- $(a)(b)[x](c)[y][z](d)z \hookrightarrow_\beta (b)[x](c)[y][z := a](d)z$

Quelques notions de réduction dans la literature

Name	In Classical Notation	In de Bruijn's notation
(θ)	$((\lambda_x.N)P)Q$ $\downarrow$ $(\lambda_x.NQ)P$	$(Q)(P)[x]N$ $\downarrow$ $(P)[x](Q)N$
(γ)	$(\lambda_x.\lambda_y.N)P$ $\downarrow$ $\lambda_y.(\lambda_x.N)P$	$(P)[x][y]N$ $\downarrow$ $[y](P)[x]N$
(γ_C)	$((\lambda_x.\lambda_y.N)P)Q$ $\downarrow$ $(\lambda_y.(\lambda_x.N)P)Q$	$(Q)(P)[x][y]N$ $\downarrow$ $(Q)[y](P)[x]N$
(g)	$((\lambda_x.\lambda_y.N)P)Q$ $\downarrow$ $(\lambda_x.N[y := Q])P$	$(Q)(P)[x][y]N$ $\downarrow$ $(P)[x][y := Q]N$

Quelques usages de ces réductions/"term reshuffling"

- Regnier [1992] introduit θ and γ et les utilise pour analyzer la strategy perpetuelle de réduction.
- [Kfoury et al., 1994; Kfoury and Wells, 1994] utilisent term reshuffling pour analyzer des problèmes de typages (comme (non)décidibilité).
- [Nederpelt, 1973; de Groote, 1993; Kfoury and Wells, 1995] les utitlisent pour réduire le problème de normalisation forte à celui de normalisation faible.
- [Ariola et al., 1995] montre que term-reshuffling peut bien representer l'évaluation fonctionnelle paresseuse.
- [Kamareddine and Nederpelt, 1995; Kamareddine et al., 1999, 1998; Bloo et al., 1996] montrent que les réductions géneralisés sont plus efficaces (espace et temps).
- [Kamareddine, 2000] établit le théorème de conservation pour les réductions generalisées.

Formes Canoniques [Kamareddine et al., 2001b]

-

des [ ] célibataires	des ()[ ]-paires	des ()s célibataires	coeur
$[x_1]\ldots[x_n]$	$(A_1)[y_1]\ldots(A_m)[y_m]$	$(B_1)\ldots(B_p)$	x

- Dans [Regnier, 1994] et [Kfoury and Wells, 1995]

$$\lambda x_1 \cdots \lambda x_n.(\lambda y_1.(\lambda y_2.\cdots(\lambda y_m.xB_p\cdots B_1)A_m\cdots)A_2)A_1$$

- Par example, la forme canonique de:

$$[x][y](a)[z][x'](b)(c)(d)[y'][z'](e)x$$

est

$$[x][y][x'](a)[z](d)[y'](c)[z'](b)(e)x$$

Comment arriver aux formes canoniques?

For $M \equiv [x][y](a)[z][x'](b)(c)(d)[y'][z'](e)x$:

$\theta(M)$:	bach. $[\,]$s $[x][y]$	$()[\,]$-pairs mixed with bach. $[\,]$s $(a)[z][x'](d)[y'](c)[z']$	bach. $()$s $(b)(e)$	end var x
$\gamma(M)$:	bach. $[\,]$s $[x][y][x']$	$()[\,]$-pairs mixed with bach. $()$s $(a)[z](b)(c)[z'](d)[y']$	bach. $()$s (e)	end var x
$\theta(\gamma(M))$:	bach. $[\,]$s $[x][y][x']$	$()[\,]$-pairs $(a)[z](c)[z'](d)[y']$	bach. $()$s $(b)(e)$	end var x
$\gamma(\theta(M))$:	bach. $[\,]$s $[x][y][x']$	$()[\,]$-pairs $(a)[z](d)[y'](c)[z']$	bach. $()$s $(b)(e)$	end var x

$\to_\theta$ and $\to_\gamma$ sont SN et CR. Alors les θ-nf and γ-nf sont unique.

$\theta(\gamma(A))$ et $\gamma(\theta(A))$ sont les deux en *forme canonique*

Notons que: $\theta(\gamma(A)) =_p \gamma(\theta(A))$ où $\to_p$ est la règle

$$(A_1)[y_1](A_2)[y_2]B \to_p (A_2)[y_2](A_1)[y_1]B \qquad \text{si } y_1 \notin \mathrm{FV}(A_2)$$

Réduction basée sur les classes de formes canoniques [Kamareddine and Bloo, 2002]

- Définissons $[A] = \{B \mid \theta(\gamma(A)) =_p \theta(\gamma(B))\}$.
- Quand $B \in [A]$,on écrit $B \approx_{\mathtt{equi}} A$.
- $\rightarrow_\theta, \rightarrow_\gamma, =_\gamma, =_\theta, =_p \subset \approx_{\mathtt{equi}} \subset =_\beta$ (inclusions strictes).
- $A \rightsquigarrow_\beta B$ iff $\exists A' \in [A].\exists B' \in [B].\ A' \rightarrow_\beta B'$ (avec compatibilité)
- Si $A \rightsquigarrow_\beta B$ alors $\forall A' \in [A].\forall B' \in [B].\ A' \rightsquigarrow_\beta B'$.
- $\rightarrow_\beta \subset \rightarrow_g \subset \rightsquigarrow_\beta \subset =_\beta = \approx_\beta$.
- $\twoheadrightsquigarrow_\beta$ *est CR:* Si $A \twoheadrightsquigarrow_\beta B$ et $A \twoheadrightsquigarrow_\beta C$, alors $\exists D$: $B \twoheadrightsquigarrow_\beta D$ et $C \twoheadrightsquigarrow_\beta D$.
- Soit $r \in \{\rightarrow_\beta, \rightsquigarrow_\beta\}$. Si $A \in SN_r$ et $A' \in [A]$ alors $A' \in SN_r$.
- $A \in SN_{\rightsquigarrow_\beta}$ ssi $A \in SN_{\rightarrow_\beta}$.

Les types simples

- Soient $f : \mathbb{N} \to \mathbb{N}$ et $g_f : \mathbb{N} \to \mathbb{N}$ tel que $g_f(x) = f(f(x))$.
  Soit $F_{\mathbb{N}} : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ tel que $F_{\mathbb{N}}(f)(x) = g_f(x) = f(f(x))$.

- Dans le lambda calcul typé de Church on écrit la fonction $F_{\mathbb{N}}$ comme suit:

$$\lambda_{f:\mathbb{N}\to\mathbb{N}}.\lambda_{x:\mathbb{N}}.f(f(x))$$

- Si on veut la même fonction sur les Booléens $\mathcal{B}$, on écrit:

la fonction $F_{\mathcal{B}}$ est	$\lambda_{f:\mathcal{B}\to\mathcal{B}}.\lambda_{x:\mathcal{B}}.f(f(x))$
le type de la fonction $F_{\mathcal{B}}$ est	$(\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})$

- *Problèmes* dans RTT et STT. Alors la naissance des *systèmes de typages differents*, chacun avec son *pouvoir d'abstraire des fonctions*.

- *8 λ-calculs typés importants* 1940–1988 ont été unifié dans le *cube de Barendregt*.

Polymorphisme: le λ-calcul typé après Church

- A la place de répéter le travail, on prend $\alpha : *$ (α est un type quelconque) et on définit une fonction polymorphique F comme suit:

$$\lambda_{\alpha:*}.\lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x))$$

  On donne à F le type:

$$\Pi_{\alpha:*}.(\alpha \to \alpha) \to (\alpha \to \alpha)$$

- Comme ça, $F(\alpha) = \lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x)) : (\alpha \to \alpha) \to (\alpha \to \alpha)$

- On peut instancier α selon notre besoin:
  - $F(\mathbb{N}) = \lambda_{f:\mathbb{N}\to\mathbb{N}}.\lambda_{x:\mathbb{N}}.f(f(x)) : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$
  - $F(\mathcal{B}) = \lambda_{f:\mathcal{B}\to\mathcal{B}}.\lambda_{x:\mathcal{B}}.f(f(x)) : (\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})$
  - $F(\mathcal{B} \to \mathcal{B}) = \lambda_{f:(\mathcal{B}\to\mathcal{B})\to(\mathcal{B}\to\mathcal{B})}.\lambda_{x:(\mathcal{B}\to\mathcal{B})}.f(f(x)) :$
    $((\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})) \to ((\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B}))$

Le cube de Barendregt

- Syntax: $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$
- Formation rule: $$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \quad \text{si} (s_1, s_2) \in \boldsymbol{R}$$

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda\!\rightarrow$	$(*,*)$				λ^τ	[Church, 1940; B
$\lambda 2$	$(*,*)$	$(\Box,*)$			F	[Girard, 1972; Re
λP	$(*,*)$		$(*,\Box)$		AUT-QE, LF	[Bruijn, 1968; Ha
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$	POLYREC	[Renardel de Lava
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$			[Longo and Mogg
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$	Fω	[Girard, 1972]
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$		
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$	CC	[Coquand and Hu

Les règles de typage

(axiom) $$\langle\rangle \vdash * : \Box$$

(start) $$\frac{\Gamma \vdash A : s \quad x \notin \mathrm{DOM}(\Gamma)}{\Gamma, x{:}A \vdash x : A}$$

(weak) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad x \notin \mathrm{DOM}(\Gamma)}{\Gamma, x{:}C \vdash A : B}$$

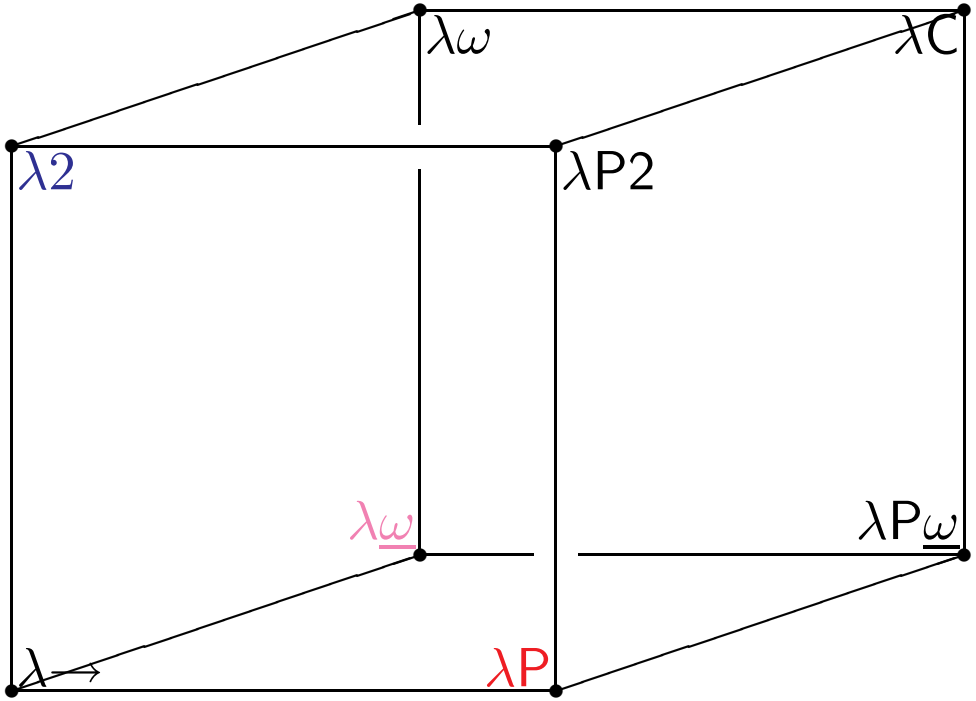
(Π) $$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \Pi_{x:A}.B : s_2}$$

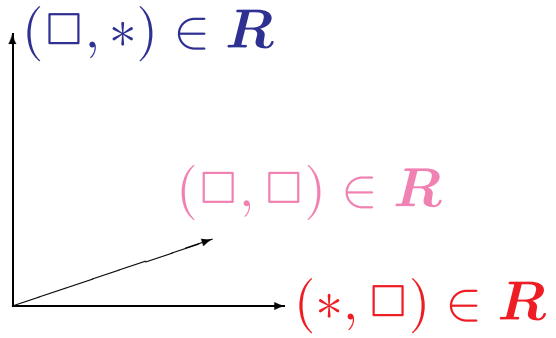
(λ) $$\frac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$$

(conv$_\beta$) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$$

(appl) $$\frac{\Gamma \vdash F : \Pi_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

Le cube de Barendregt





Notre example dans le système F de Girard

- Si $x \notin FV(B)$ on écrit $A \to B$ à la place de $\Pi_{x:A}.B$.
- $\alpha : *, f : \alpha \to \alpha \vdash \lambda_{x:\alpha}.f(f(x)) : \alpha \to \alpha : *$
  (besoin de la règle $(*, *)$).
- $\alpha : * \vdash \lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x)) : (\alpha \to \alpha) \to (\alpha \to \alpha) : *$
  (besoin de la règle $(*, *)$).
- $\vdash \lambda_{\alpha:*}.\lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x)) : \Pi_{\alpha:*}.(\alpha \to \alpha) \to (\alpha \to \alpha) : *$
  (besoin de la règle $(\Box, *)$).

- Fonctions de Frege $\neq$ fonctions du Principia $\neq$ fonctions du *λ-calcul* (1932).
- Frege, Russell et Church écrivait $x \mapsto x + 3$ comme $x + 3$, $\hat{x} + 3$ et $\lambda x.x + 3$.
- Church traitait chaque fonction comme un citoyen de 1ère classe. Mais *les fonctions qui ne sont pas de 1ère classe* nous permettent de rester sur un niveau de typage plus bas (guardant la décidabilité) sans perdre la flexibilité.
- Il n'est pas nécessaires d'abstraire chaque fonction comme dans le λ-calcul. Historiquement, les functions ont toujours été traité comme meta-objets.
- Les $valeurs$ des fonctions etaient le plus important, pas les fonctions abstraites.
- La fonction sinus, est toujours exprimée avec une valeur: $\sin(\pi)$, $\sin(x)$ et des propriétés comme: $\sin(2x) = 2\sin(x)\cos(x)$.
- [Kamareddine et al., 2003, 2001a] étend le cube avec la notion de fonctions qui ne sont pas 1ère classe, et explique que cette extension pertmette bien de placer des systèmes importants comme le ML de Milner, l'Automath et le LF d'Edimbourgh précisement dans la hierarchie des types.

ML

- ML traite let val id $=$ (fn $x \Rightarrow x$) in (id id) end comme le terme
  $(\lambda \mathsf{id}{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ \mathsf{id}(\beta \to \beta)(\mathsf{id}\,\beta))(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$

- On a besoin de la règle $(\Box, *)$ (i.e., $\lambda 2$) pour pouvoir typer ce terme.

- Les règles de typage de ML ne permettent pas le terme:
  let val id $=$ (fn $x \Rightarrow x$) in (fn $y \Rightarrow y\,y$)(id id) end
  Mais, ce terme là est equivalent à un terme de $\lambda 2$ qui est bien-typé:
  $(\lambda \mathsf{id} : (\Pi\alpha{:}*.\ \alpha \to \alpha).$
  $\quad(\lambda y{:}(\Pi\alpha{:}*.\ \alpha \to \alpha).\ y(\beta \to \beta)(y\,\beta))$
  $\quad(\lambda\alpha{:}*.\ \mathsf{id}(\alpha \to \alpha)(\mathsf{id}\,\alpha)))$
  $(\lambda\alpha{:}*.\ \lambda x{:}\alpha.\ x)$

- ML ne doit pas avoir tout le pouvoir de la règle de Π-formation $(\Box, *)$.

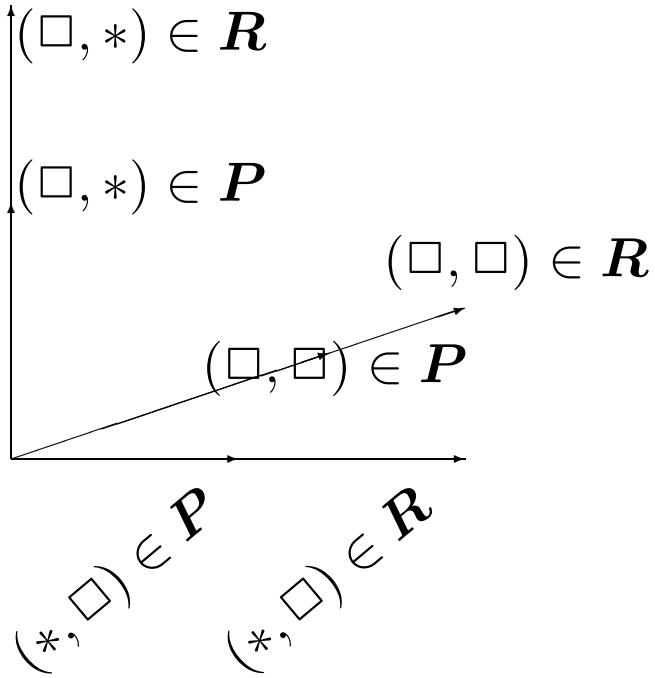
LF

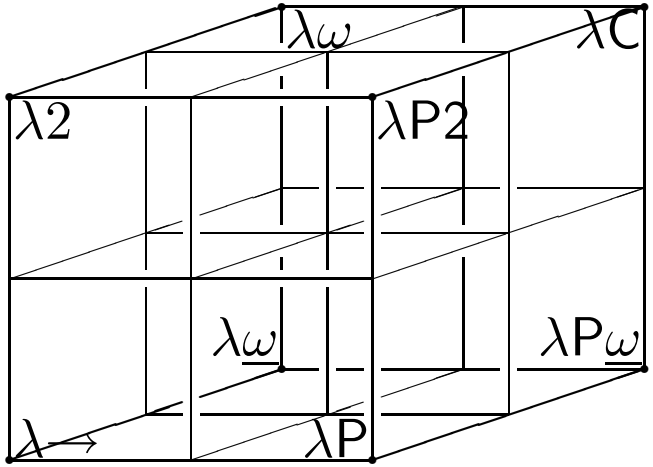
- LF [Harper et al., 1987] a un peu le pouvoir de la règle $(*, \Box)$, mais n'est pas equivalent au système λP du cube.
- l'usage de règle $(*, \Box)$ est très rare.
- On a besoin d'un type $\Pi x{:}A.B : \Box$ seulement lorceque le principe: "Propositions-As-Types" PAT s'applique pendant la construction du type $\Pi\alpha{:}\mathtt{prop}.*$ de l'operateur Prf qui pour une proposition Σ, donne Prf(Σ) le type des pereuves de Σ.

$$\frac{\mathtt{prop}{:}* \vdash \mathtt{prop}{:}* \qquad \mathtt{prop}{:}*, \alpha{:}\mathtt{prop} \vdash *{:}\Box}{\mathtt{prop}{:}* \vdash \Pi\alpha{:}\mathtt{prop}.* : \Box}.$$

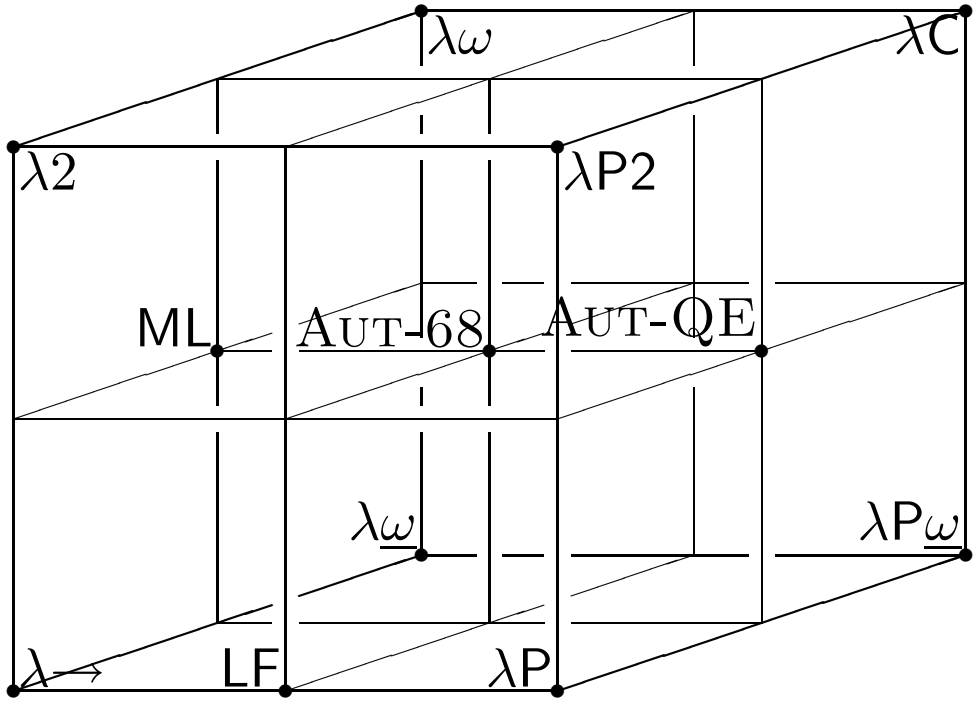
- Dans LF, c'est l'unique place où on utilise règle $(*, \Box)$.
- Mais on utilise Prf seulement avec un $\Sigma{:}\mathtt{prop}$. On n'utilise jamais le Prf seul.
- On va précisement placer LF sure le Cube (entre $\lambda{\to}$ et λP).

Une version raffinée du cube [Kamareddine et al., 2003]





LF, ML, Aut-68, et Aut-QE dans la version raffinée du cube



Le logicien, le mathématicien et l'induction sur les nombres

- Un Logicien utilise `ind: Ind`. Le type `Ind` peut seulement être décrit dans un $\lambda\boldsymbol{R}$ avec $\boldsymbol{R} = \{(*,*),(*,\Box),(\Box,*)\}$:

$$\texttt{Ind} = \Pi p{:}(\mathbb{N}\to *).p0\to(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn\to Snm\to pm)\to\Pi n{:}\mathbb{N}.pn \quad (1)$$

- Un Mathématicien utilise `ind` seulement avec
  $P : \mathbb{N}\to *$, $Q : P0$ et $R : (\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.Pn\to Snm\to Pm)$
  pour former un terme $(\texttt{ind}PQR){:}(\Pi n{:}\mathbb{N}.Pn)$.

- L'usage de l'axiom d'induction par le mathématician peut être décrit par le schema (p, q et r sont les *parameters* du schema):

$$\texttt{ind}(p{:}\mathbb{N}\to *, q{:}p0, r{:}(\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}.pn\to Snm\to pm)) : \Pi n{:}\mathbb{N}.pn \quad (2)$$

- Le mathématicien n'a pas besoin du type `Ind` du logicien. Les types dans 2 peuvent être construit dans $\lambda\boldsymbol{R}$ avec $\boldsymbol{R} = \{(*,*)(*,\Box)\}$.

- Un Mathématicien: *applique* l'axiom d'induction et n'a pas besoin de la théorie des preuves derrière.
- Un logicien developpe l'axiom d'induction ou étudie ses propriétés.
- Le mathématicien n'a pas besoin de $(\Box, *)$ tandis que le logicien en a besoin pour pouvoir former l'abstraction $\Pi p{:}(\mathbb{N} \to *).\cdots$.
- Alors le système de types utilisé pour décrire l'usage du mathématicien est plus faible que celui pour le logicien.
- Ce nouveau cube est basé sur deux notions de mathématiques: Les définitions et les paramètres.

Etendre le Cube avec des constantes parametriques

- On ajoute des des constantes parametriques de la forme $c(b_1, \ldots, b_n)$ où $b_1, \ldots, b_n$ sont des termes de certains types et $c \in \mathcal{C}$.
- $b_1, \ldots, b_n$ sont les *paramèters* de $c(b_1, \ldots, b_n)$.
- $\mathbf{R}$ permet plusieures sortes de Π-constructs. On utilise aussi un ensemble $\boldsymbol{P}$ of (s_1, s_2) où $s_1, s_2 \in \{*, \Box\}$. Ceci permet plusieures sortes de constantes parametriques.
- $(s_1, s_2) \in \boldsymbol{P}$ veut dire que permet des constantes parametriques $c(b_1, \ldots, b_n)$: A où $b_1, \ldots, b_n$ ont les types $B_1, \ldots, B_n$ de sorte s_1, et$A$ est de type s_2.

Le Cube avec les paramètres

- soient $(*,*) \subseteq \mathbf{R}, \boldsymbol{P} \subseteq \{(*,*),(*,\Box),(\Box,*),(\Box,\Box)\}$.

- $\lambda\mathbf{R}\boldsymbol{P} = \lambda\mathbf{R}$ et les deux règles **($\vec{\mathbf{C}}$-weak)** et **($\vec{\mathbf{C}}$-app)**:

$$\frac{\Gamma \vdash b : B \quad \Gamma, \Delta_i \vdash B_i : s_i \quad \Gamma, \Delta \vdash A : s}{\Gamma, c(\Delta) : A \vdash b : B} \; (s_i, s) \in \boldsymbol{P}, c \text{ est } \Gamma\text{-fraiche}$$

$$\frac{\begin{array}{lcll} \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \ldots, n) \\ \Gamma_1, c(\Delta){:}A, \Gamma_2 & \vdash & A : s & (\text{si } n = 0) \end{array}}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$$

$\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n.$
$\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$

Propriétés du Cube Raffiné

- (Correctnesse des types) Si $\Gamma \vdash A : B$ alors $(B \equiv \Box$ or $\Gamma \vdash B : S)$.
- (Réduction du Sujet SR) Si $\Gamma \vdash A : B$ et $A \twoheadrightarrow_\beta A'$ alors $\Gamma \vdash A' : B$
- (Normalisation forte) Pour chaque terme M qui est $\vdash$-legal, on a $\mathrm{SN}_{\twoheadrightarrow_\beta}(M)$.
- On a aussi Unicité des types et typabilité des sous-terms, etc.
- $\lambda\mathbf{R}\boldsymbol{P}$ est le system qui a les règles: de Π-formation $\boldsymbol{R}$ et de paramètre $\boldsymbol{P}$.
- Soit $\lambda\mathbf{R}\boldsymbol{P}$ tel que $(s_1, s_2) \in \boldsymbol{P}$ implique $(s_1, s_2) \in \boldsymbol{R}$.
  - Le système sans paramètre $\lambda\mathbf{R}$ est le plus petit système qui a au moin le pouvoir de $\lambda\mathbf{R}\boldsymbol{P}$.
  - Si $\Gamma \vdash_{\mathbf{R}\boldsymbol{P}} a : A$ alors $\{\Gamma\} \vdash_{\boldsymbol{R}} \{a\} : \{A\}$.

La notation Item dans les systèmes de type

- On écrit les items dans $()$ en plce de $()$ et $[]$.
- On écrit $(\lambda_x.x)y$ comme: $(y\delta)(\lambda_x)x$ en place de $(y)[x]x$.
- On écrit $\Pi_{z:*}.(\lambda_{x:z}.x)y$ comme: $(*\Pi_z)(y\delta)(z\lambda_x)x$.

Le cube de Barendregt en notation item avec réduction de classe

- Même formulation sauf que les termes sont écrits en notation item:
- $\mathcal{T} = * \mid \Box \mid V \mid (\mathcal{T}\,\delta)\mathcal{T} \mid (\mathcal{T}\,\lambda_V)\mathcal{T} \mid (\mathcal{T}\,\Pi_V)\mathcal{T}$.
- Les règles de typages ne changent même si on utilise $\rightsquigarrow_\beta$ en place de $\rightarrow_\beta$ (devinez pourquoi).

Le “Subject Reduction” n’ait plus valable

- La plupart des propriétés (inclusif SN) sont valable pour le cube étendu avec la réduction modulo les classes.
- *MAIS* SN n’est pas valable que dans $\lambda_{\rightarrow}$ $(*, *)$ et $\lambda\underline{\omega}$ $(\Box, \Box)$.
- SR ne marche plus dans λP $(*, \Box)$ (et alors dans $\lambda P2, \lambda P\underline{\omega}$ et λC).
- SR ne marche plus dans $\lambda 2$ $(\Box, *)$ (et alors dans $\lambda P2, \lambda\omega$ et λC):

Pourquoi on perd le SR?

- $(y'\delta)(\beta\delta)(*\lambda_\alpha)(\alpha\lambda_y)(y\delta)(\alpha\lambda_x)x \rightsquigarrow_\beta (\beta\delta)(*\lambda_\alpha)(y'\delta)(\alpha\lambda_x)x$.
- $(\lambda_{\alpha:*}.\lambda_{y:\alpha}.(\lambda_{x:\alpha}.x)y)\beta y' \rightsquigarrow_\beta (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta$
- $\beta : *, y' : \beta \vdash_{\lambda 2} (\lambda_{\alpha:*}.\lambda_{y:\alpha}.(\lambda_{x:\alpha}.x)y)\beta y' : \beta$
- Mais, $\beta : *, y' : \beta \not\vdash_{\lambda 2} (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta : \tau$ pour tout τ.
- On a perrdu l'information que $y' : \beta$ a remplacé $y' : \alpha$ $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta$.
- On a besoin de $y' : \alpha$ pour typer le sousterm $(\lambda_{x:\alpha}.x)y'$ de $(\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta$ et alors pour typer $\beta : *, y' : \beta \vdash (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta : \beta$.

Solution de SR: Utilise "let expressions/définitions"

- Définitions/let expressions ont la forme: let $x : A = B$. On les ajoute aux contextes exactement comme les déclarations $y : C$.
- (def rule) $$\frac{\Gamma, \text{let } x : A = B \vdash^{\mathsf{c}} C : D}{\Gamma \vdash^{\mathsf{c}} (\lambda_{x:A}.C)B : D[x := A]}$$
- On définit $\Gamma \vdash^{\mathsf{c}} \cdot =_{\mathtt{def}} \cdot$ comme la relation d' equivalence generée par
  - si $A =_\beta B$ alors $\Gamma \vdash^{\mathsf{c}} A =_{\mathtt{def}} B$
  - si let $x : M = N$ est dans Γ et si B vient de A en substituant une occurrence de x dans A par N, alors $\Gamma \vdash^{\mathsf{c}} A =_{\mathtt{def}} B$.

Le Cube (simplifiée) avec définitions et réduction de classes [Kamareddine and Bloo, 2002]

(axiom) (app) (abs) et (form) ne changent pas.

(start) $\dfrac{\Gamma \vdash^{c} A : s}{\Gamma, x{:}A \vdash^{c} x : A}$ $\dfrac{\Gamma \vdash^{c} A : s \quad \Gamma \vdash^{c} B : A}{\Gamma,\ \mathsf{let}\ x : A\ = B \vdash^{c} x : A}$ x fraic

(weak) $\dfrac{\Gamma \vdash^{c} D : E \quad \Gamma \vdash^{c} A : s}{\Gamma, x{:}A \vdash^{c} D : E}$ $\dfrac{\Gamma \vdash^{c} A : s \quad \Gamma \vdash^{c} B : A \quad \Gamma \vdash^{c} D : E}{\Gamma,\ \mathsf{let}\ x : A\ = B \vdash^{c} D : E}$ x fraic

(conv) $\dfrac{\Gamma \vdash^{c} A : B \qquad \Gamma \vdash^{c} B' : S \qquad \Gamma \vdash^{c} B =_{\mathtt{def}} B'}{\Gamma \vdash^{c} A : B'}$

(def) $\dfrac{\Gamma, \mathsf{let}\ x : A\ = B \vdash^{c} C : D}{\Gamma \vdash^{c} (\lambda_{x:A}.C)B : D[x := A]}$

l'usage des définitions résoud le problème de subject reduction

1. $\beta : *, y' : \beta,\ \mathsf{let}\ \alpha : * = \beta \qquad \vdash^{\mathsf{c}} y' : \beta$
2. $\beta : *, y' : \beta,\ \mathsf{let}\ \alpha : * = \beta \qquad \vdash^{\mathsf{c}} \alpha =_{\mathtt{def}} \beta$
3. $\beta : *, y' : \beta,\ \mathsf{let}\ \alpha : * = \beta \qquad \vdash^{\mathsf{c}} y' : \alpha$ (de 1 and 2)
4. $\beta : *, y' : \beta,\ \mathsf{let}\ \alpha : * = \beta,\ \mathsf{let}\ x : \alpha = y' \qquad \vdash^{\mathsf{c}} x : \alpha$
5. $\beta : *, y' : \beta,\ \mathsf{let}\ \alpha : * = \beta \qquad \vdash^{\mathsf{c}} (\lambda_{x:\alpha}.x)y' : \alpha[x := y'] = \alpha$

$$\beta : *, y' : \beta \qquad \vdash^{\mathsf{c}} \qquad (\lambda_{\alpha:*}.(\lambda_{x:\alpha}.x)y')\beta : \alpha[\alpha := \beta] = \beta$$

Les propriétés du Cube avec définitions et réduction de classes

- $\vdash^{\mathsf{c}}$ est une generalisation of $\vdash$: Si $\Gamma \vdash A : B$ alors $\Gamma \vdash^{\mathsf{c}} A : B$.
- Equivalent terms have same types:
  Si $\Gamma \vdash^{\mathsf{c}} A : B$ et $A' \in [A]$, $B' \in [B]$ alors $\Gamma \vdash^{\mathsf{c}} A' : B'$.
- Subject Reduction for $\vdash^{\mathsf{c}}$ and $\twoheadrightarrow_\beta$:
  Si $\Gamma \vdash^{\mathsf{c}} A : B$ et $A \twoheadrightarrow_\beta A'$ alors $\Gamma \vdash^{\mathsf{c}} A' : B$.
- Unicity of Types for $\vdash^{\mathsf{c}}$:
  - Si $\Gamma \vdash^{\mathsf{c}} A : B$ et $\Gamma \vdash^{\mathsf{c}} A : B'$ alors $\Gamma \vdash^{\mathsf{c}} B =_{\mathtt{def}} B'$
  - Si $\Gamma \vdash^{\mathsf{c}} A : B$ et $\Gamma \vdash^{\mathsf{c}} A' : B'$ et $\Gamma \vdash^{\mathsf{c}} A =_\beta A'$ alors $\Gamma \vdash^{\mathsf{c}} B =_{\mathtt{def}} B'$.
- Strong Normalisation of $\twoheadrightarrow_\beta$:
  Chaque terme légal est fortement normalisable par rapport à $\twoheadrightarrow_\beta$.

Les indices de de Bruijn

- λ-calcul classique: $A ::= x \mid (\lambda x.B) \mid (BC)$
  $(\lambda x.A)B \to_\beta A[x := B]$
- $(\lambda x.\lambda y.xy)y \to_\beta (\lambda y.xy)[x := y] \neq \lambda y.yy$
- $(\lambda x.\lambda y.xy)y \to_\beta (\lambda y.xy)[x := y] =_\alpha (\lambda z.xz)[x := y] = \lambda z.yz$
- $\lambda x.x$ and $\lambda y.y$ sont la même fonction. Soit $\lambda 1$.
- Supposons une liste de variables libres (disons $x, y, z, \ldots$).
- $(\lambda\lambda 2\ 1)2 \to_\beta (\lambda 2\ 1)[1 := 2] = \lambda(2[2 := 3])(1[2 := 3]) = \lambda 3\ 1$

Le λ-calcul classique avec les indices de de Bruijn

- Soient $i, n \geq 1$ et $k \geq 0$
- $A \ ::= \ n \mid (\lambda B) \mid (BC)$
  $(\lambda A)B \rightarrow_\beta A\{\!\{1 \leftarrow B\}\!\}$
- $$\begin{array}{l} U_k^i(AB) = U_k^i(A)\, U_k^i(B) \\ U_k^i(\lambda A) = \lambda(U_{k+1}^i(A)) \end{array} \qquad U_k^i(\mathtt{n}) = \begin{cases} \mathtt{n+i-1} & \text{si} \quad n > k \\ \mathtt{n} & \text{si} \quad n \leq k\,. \end{cases}$$
- $$\begin{array}{lcl} (A_1 A_2)\{\!\{\mathtt{i} \leftarrow B\}\!\} & = & (A_1\{\!\{\mathtt{i} \leftarrow B\}\!\})\,(A_2\{\!\{\mathtt{i} \leftarrow B\}\!\}) \\ (\lambda A)\{\!\{\mathtt{i} \leftarrow B\}\!\} & = & \lambda(A\{\!\{\mathtt{i+1} \leftarrow B\}\!\}) \end{array}$$
  $$\mathtt{n}\{\!\{\mathtt{i} \leftarrow B\}\!\} = \begin{cases} \mathtt{n-1} & \text{si} \quad n > i \\ U_0^i(B) & \text{si} \quad n = i \\ \mathtt{n} & \text{si} \quad n < i\,. \end{cases}$$
- Plusieures implantations des systèmes de preuves et des languages de programmation sont basées sur les indices de de Bruijn.

Les wagons de substitutions [Kamareddine and Nederpelt, 1993]

- $(B)[x]A \rightarrow_\beta [x := B]A$ devient $(B\delta)(\lambda_x)A \rightarrow (B\sigma^x)A$
- Si on utilise les indices de de Bruijn, ca devient: $(B\delta)(\lambda)A \rightarrow (B\sigma^1)A$
- Les σ-wagons propagent les λ- et δ-wagons:

$$\begin{array}{lrcl} \sigma\delta\text{-}transition & (C\sigma^i)(B\delta)A & \rightarrow & ((C\sigma^i)B\delta)(C\sigma^i)A \\ \sigma\lambda\text{-}transition & (C\sigma^i)(\lambda)A & \rightarrow & (\lambda)(C\sigma^{i+1})A \end{array}$$

- On a aussi besoin des φ-wagons qui changent les variables. Ces wagons, eux aussi propagent les λ- et δ-wagons:

$$\begin{array}{lrcl} \varphi\delta\text{-}transition & (\varphi^i_k)(B\delta)A & \rightarrow & ((\varphi^i_k)B\delta)(\varphi^i_k)A \\ \varphi\lambda\text{-}transition & (\varphi^i_k)(\lambda)A & \rightarrow & (\lambda)(\varphi^i_{k+1})A \end{array}$$

Le λs_e- calcul [Kamareddine and Ríos, 1995, 1997]

- Il y avait aussi les règles de destruction des nouveaux wagons, et des règles permettant un σ-wagon (resp., un δ-wagon) à propager les σ- et δ-wagons.
- [Kamareddine and Ríos, 1995] a démontré qu'une restriction λs de ce calcul (qui reflète le calcul lui même de de Bruijn), satisfait les bonnes propriétés sur les termes clos.
- Pour la confluence de λs étendu avec les termes ouverts, [Kamareddine and Ríos, 1997] ajoutait des règles qui reflètent six lemmes sur les propriétés du calcul de de Bruijn donnat λs_e.
- Comme $\lambda\sigma$, λs_e est confluent et n'a pas PSN (René David et Bruno Guillaume).
- Mais, $\lambda\sigma$ n'as pas une restriction sur les termes clos satisfaisant PSN, simulation de beta et confluence tandis que λs_e a le λs.

From classical λ-calculus with de Bruijn indices to substitution calculus λs [Kamareddine and Ríos, 1995]

- Ecrivons $A\{\!\{n \leftarrow B\}\!\}$ comme $A\sigma^n\, B$ et $U_k^i(A)$ comme $\varphi_k^i A$.
- $A \;::=\; n \mid (\lambda B) \mid (BC) \mid (A\sigma^i B) \mid (\varphi_k^i B) \qquad where \quad i, n \geq 1\,, \quad k \geq 0\,.$

σ-*generation*	$(\lambda A)\, B$	$\longrightarrow$	$A\, \sigma^1\, B$
σ-λ-*transition*	$(\lambda A)\, \sigma^i B$	$\longrightarrow$	$\lambda(A\, \sigma^{i+1}\, B)$
σ-*app-transition*	$(A_1\, A_2)\, \sigma^i B$	$\longrightarrow$	$(A_1\, \sigma^i B)\, (A_2\, \sigma^i B)$
σ-*destruction*	$\mathtt{n}\, \sigma^i B$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \mathsf{si} \quad n > i \\ \varphi_0^i\, B & \mathsf{si} \quad n = i \\ \mathtt{n} & \mathsf{si} \quad n < i \end{cases}$
φ-λ-*transition*	$\varphi_k^i(\lambda A)$	$\longrightarrow$	$\lambda(\varphi_{k+1}^i\, A)$
φ-*app-transition*	$\varphi_k^i(A_1\, A_2)$	$\longrightarrow$	$(\varphi_k^i\, A_1)\, (\varphi_k^i\, A_2)$
φ-*destruction*	$\varphi_k^i\, \mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+i-1} & \mathsf{si} \quad n > k \\ \mathtt{n} & \mathsf{si} \quad n \leq k \end{cases}$

1. Le s-calcul (i.e., λs moins σ-generation) est fortement normalisable.

2. Le λs-calcul est confluent and simule (en petit pas) la β-reduction

3. Le λs-calculus preserve la normalisation forte.

4. Le λs-calculus a une extension avec les termes ouverts qui est confluente.

- Pierre Lesanne a posé la question de l'existence d'un calcul qui satisfait ses propriétés mais qui est lui-même confluent sur les termes ouverts.

- Le λs-calculus satisfait les propriétés 1., 2., 3. et 4. Mais ni le λs ni le λs_e n'est confluent sur les termes ouverts (Guillaume).

- David et Guillaume donne un calcul qui a des bonnes propriétés.

$\lambda\upsilon$ [Benaissa et al., 1996]

Termes: $\Lambda\upsilon^t ::= \mathbb{N} \mid \Lambda\upsilon^t\Lambda\upsilon^t \mid \lambda\Lambda\upsilon^t \mid \Lambda\upsilon^t[\Lambda\upsilon^s]$
Substitutions: $\Lambda\upsilon^s ::= \uparrow \mid \Uparrow(\Lambda\upsilon^s) \mid \Lambda\upsilon^t.$

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b/]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(FVar)	$\mathtt{1}\,[a/]$	$\longrightarrow$	a
(RVar)	$\mathtt{n}+\mathtt{1}\,[a/]$	$\longrightarrow$	$\mathtt{n}$
(FVarLift)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(RVarLift)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}\,[s]\,[\uparrow]$
(VarShift)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$

$\lambda\upsilon$ satisfait 1., 2., and 3., mais n'a pas une extension sur les termes ouverts qui est confluente.

$\lambda\sigma_{\Uparrow}$

Terms: $\Lambda\sigma^t_{\Uparrow} ::= \mathbb{N} \mid \Lambda\sigma^t_{\Uparrow}\Lambda\sigma^t_{\Uparrow} \mid \lambda\Lambda\sigma^t_{\Uparrow} \mid \Lambda\sigma^t_{\Uparrow}[\Lambda\sigma^s_{\Uparrow}]$

Substitutions: $\Lambda\sigma^s_{\Uparrow} ::= id \mid \uparrow \mid \Uparrow(\Lambda\sigma^s_{\Uparrow}) \mid \Lambda\sigma^t_{\Uparrow} \cdot \Lambda\sigma^s_{\Uparrow} \mid \Lambda\sigma^s_{\Uparrow} \circ \Lambda\sigma^s_{\Uparrow}.$

(Beta)	$(\lambda a)\, b$	$\longrightarrow$	$a\,[b \cdot id]$
(App)	$(a\, b)[s]$	$\longrightarrow$	$(a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a\,[\Uparrow(s)])$
(Clos)	$(a\,[s])[t]$	$\longrightarrow$	$a\,[s \circ t]$
(Varshift1)	$\mathtt{n}\,[\uparrow]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}$
(Varshift2)	$\mathtt{n}\,[\uparrow \circ s]$	$\longrightarrow$	$\mathtt{n}+\mathtt{1}\,[s]$
(FVarCons)	$\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	a
(RVarCons)	$\mathtt{n}+\mathtt{1}\,[a \cdot s]$	$\longrightarrow$	$\mathtt{n}\,[s]$
(FVarLift1)	$\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{1}$
(FVarLift2)	$\mathtt{1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{1}\,[t]$
(RVarLift1)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s)]$	$\longrightarrow$	$\mathtt{n}[s \circ \uparrow]$
(RVarLift2)	$\mathtt{n}+\mathtt{1}\,[\Uparrow(s) \circ t]$	$\longrightarrow$	$\mathtt{n}[s \circ (\uparrow \circ t)]$

$\lambda\sigma_{\Uparrow}$ rules continued

(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a\,[t] \cdot (s \circ t)$
(Ass)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\longrightarrow$	s
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\longrightarrow$	$s \circ \uparrow$
(ShiftLift2)	$\uparrow \circ (\Uparrow(s) \circ t)$	$\longrightarrow$	$s \circ (\uparrow \circ t)$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\longrightarrow$	$\Uparrow(s \circ t)$
(Lift2)	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\longrightarrow$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\longrightarrow$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
(LiftId)	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a\,[id]$	$\longrightarrow$	a

$\lambda\sigma_{\Uparrow}$ satisfait 1., 2., et 4., mais n'a pas PSN.

Comment on obtient λse de λs?

- $A ::= X \mid n \mid (\lambda B) \mid (BC) \mid (A\sigma^i B) \mid (\varphi_k^i B)$ où $i, n \geq 1\,,\ k \geq 0\,.$
- Ajouter les termes ouverts sans étendre les règles aboutit a la perte de la confluence (même locale):
  $((\lambda X)Y)\sigma^1 1 \to (X\sigma^1 Y)\sigma^1 1 \qquad ((\lambda X)Y)\sigma^1 1 \to ((\lambda X)\sigma^1 1)(Y\sigma^1 1)$
- $(X\sigma^1 Y)\sigma^1 1$ et $((\lambda X)\sigma^1 1)(Y\sigma^1 1)$ n'ont pas un reduct commun.
- Mais , $((\lambda X)\sigma^1 1)(Y\sigma^1 1) \twoheadrightarrow (X\sigma^2 1)\sigma^1(Y\sigma^1 1)$
- La solution est simple: ajoute les lemmes de metasubstitution et distribution aux règles de λs:

σ-σ	$(A\sigma^i B)\,\sigma^j\,C$	$\longrightarrow$	$(A\,\sigma^{j+1}\,C)\,\sigma^i\,(B\,\sigma^{j-i+1}\,C)$	si	$i \leq j$
σ-φ *1*	$(\varphi^i_k\,A)\,\sigma^j\,B$	$\longrightarrow$	$\varphi^{i-1}_k\,A$	si	$k < j < k+i$
σ-φ *2*	$(\varphi^i_k\,A)\,\sigma^j\,B$	$\longrightarrow$	$\varphi^i_k(A\,\sigma^{j-i+1}\,B)$	si	$k+i \leq j$
φ-σ	$\varphi^i_k(A\,\sigma^j\,B)$	$\longrightarrow$	$(\varphi^i_{k+1}\,A)\,\sigma^j\,(\varphi^i_{k+1-j}\,B)$	si	$j \leq k+1$
φ-φ *1*	$\varphi^i_k\,(\varphi^j_l\,A)$	$\longrightarrow$	$\varphi^j_l\,(\varphi^i_{k+1-j}\,A)$	si	$l+j \leq k$
φ-φ *2*	$\varphi^i_k\,(\varphi^j_l\,A)$	$\longrightarrow$	$\varphi^{j+i-1}_l\,A$	si	$l \leq k < l+j$

- Ces nouvelles règles de réecriture sont bien connues: les lemmes de meta-substitution ($\sigma - \sigma$) et de distribution ($\varphi - \sigma$) (et 4 autres règles necessaires pour les demontrer).

D'où venaient ces règles?

Dans le λ-calcul de de Bruijn nous avons les lemmes:

$(\sigma-\varphi\ 1)$ Si $k<j<k+i$ alors: $U_k^{i-1}(A)=U_k^i(A)\{\!\{\mathtt{j}\leftarrow B\}\!\}$.

$(\varphi-\varphi\ 2)$ Si $l\leq k<l+j$ alors: $U_k^i(U_l^j(A))=U_l^{j+i-1}(A)$.

$(\sigma-\varphi\ 2)$ Si $k+i\leq j$ alors: $U_k^i(A)\{\!\{\mathtt{j}\leftarrow B\}\!\}=U_k^i(A\{\!\{\mathtt{j}-\mathtt{i}+\mathtt{1}\leftarrow B\}\!\})$.

$(\sigma-\sigma)$ *[Meta-substitution lemma]* Si $i\leq j$ alors:
$A\{\!\{\mathtt{i}\leftarrow B\}\!\}\{\!\{\mathtt{j}\leftarrow C\}\!\}=A\{\!\{\mathtt{j}+\mathtt{1}\leftarrow C\}\!\}\{\!\{\mathtt{i}\leftarrow B\{\!\{\mathtt{j}-\mathtt{i}+\mathtt{1}\leftarrow C\}\!\}\}\!\}$.

$(\varphi-\varphi\ 1)$ Si $j\leq k+1$ alors:
$U_{k+p}^i(U_p^j(A))=U_p^j(U_{k+p+1-j}^i(A))$.

$(\varphi-\sigma)$ *[Distribution lemma]*
Si $j\leq k+1$ alors: $U_k^i(A\{\!\{\mathtt{j}\leftarrow B\}\!\})=U_{k+1}^i(A)\{\!\{\mathtt{j}\leftarrow U_{k+1-j}^i(B)\}\!\}$.

La preuve de $(\sigma-\sigma)$ utilise $(\sigma-\varphi\ 1)$ et $(\sigma-\varphi\ 2)$ avec $k=0$.
La preuve de $(\sigma-\varphi\ 2)$ utilise $(\varphi-\varphi\ 2)$ avec $l=0$.
Finalement, $(\varphi-\varphi\ 1)$ avec $p=0$ est nécessaire pour démontrer $(\varphi-\sigma)$).

Rappel

- A la place de répéter le travail, on prend $\alpha : *$ (α est un type quelconque) et on définit une fonction polymorphique F comme suit:

  $$\lambda_{\alpha:*}.\lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x))$$

  On donne à F le type:

  $$\Pi_{\alpha:*}.(\alpha \to \alpha) \to (\alpha \to \alpha)$$

- Comme ça, $F(\alpha) = \lambda_{f:\alpha\to\alpha}.\lambda_{x:\alpha}.f(f(x)) : (\alpha \to \alpha) \to (\alpha \to \alpha)$

- On peut instancier α selon notre besoin:
  - $F(\mathbb{N}) = \lambda_{f:\mathbb{N}\to\mathbb{N}}.\lambda_{x:\mathbb{N}}.f(f(x)) : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$
  - $F(\mathcal{B}) = \lambda_{f:\mathcal{B}\to\mathcal{B}}.\lambda_{x:\mathcal{B}}.f(f(x)) : (\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})$
  - $F(\mathcal{B} \to \mathcal{B}) = \lambda_{f:(\mathcal{B}\to\mathcal{B})\to(\mathcal{B}\to\mathcal{B})}.\lambda_{x:(\mathcal{B}\to\mathcal{B})}.f(f(x)) :$
    $((\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B})) \to ((\mathcal{B} \to \mathcal{B}) \to (\mathcal{B} \to \mathcal{B}))$

- Comme ça, les types sont comme les fonctions:
  - On peut les construire par abstraction.
  - On peut les instancier

- Mais, pendant le passage des types simples aux types polymorphiques, on a oublié d'atapter la règle

$$(\beta) \qquad (\lambda_{x:A}.b)C \to b[x := C]$$

  à une règle semblable pour le Π:

$$(\Pi) \qquad (\Pi_{x:A}.B)C \to B[x := C]$$

- D'habitude, si $b : B$, on prend $(\lambda_{x:A}.b)C : B[x := C]$ à la place de prendre $(\lambda_{x:A}.b)C : (\Pi_{x:A}.B)C$

- Dans la literature, il y a plusieures études (comme le Automath de de Bruijn) qui montrent que la règle Π est utile.

- Il parait que le development des théories des types nous amène de plus en plus à adopter des rôle identiques pour les lieurs λ et Π. Est-ce qu'il y avait vraiment le besoin de les séparer? Je crois que la séparation est artificielle.

- Dans Automath, de Bruijn avait déjà identifié les deux lieurs mais dans un système de types beaucoups moins riche que les systèmes du cube. Il ecrivait: $[x : A]B$ pour les deux $\lambda_{x:A}.B$ et $\Pi_{x:A}.B$

- Quelles sont les propriétés des theories des types avec un seul lieur qui represente au même temps le λ et le Π?

- Est ce que le passage des systèmes des types avec deux lieurs à des systèmes avec un seul lieur est assez nécessaire comme le passage des types simples aux types polymorphiques et independents?

- Je crois que dès le début, il fallait pas se limiter aux types simples. Aussi, dès le début, il fallait pas distinguer entre les termes et les types.

Notation [Kamareddine 2005]

- Soit $\mathcal{V} = \mathcal{V}^* \cup \mathcal{V}^\square$ où $\mathcal{V}^* \cap \mathcal{V}^\square = \emptyset$.

  $\mathcal{T} ::= * \mid \square \mid \mathcal{V} \mid \lambda_{\mathcal{V}:\mathcal{T}}.\mathcal{T} \mid \Pi_{\mathcal{V}:\mathcal{T}}.\mathcal{T} \mid \mathcal{T}\mathcal{T}$

  $\mathcal{T}_\flat ::= * \mid \square \mid \mathcal{V} \mid \flat_{\mathcal{V}:\mathcal{T}_\flat}.\mathcal{T}_\flat \mid \mathcal{T}_\flat\mathcal{T}_\flat$

- Si $A \in \mathcal{T}$, on définit $\overline{A} \in \mathcal{T}_\flat$ par: $\overline{s} \equiv s, \overline{x} \equiv x, \overline{AB} \equiv \overline{A}\,\overline{B}, \overline{\lambda_{x:A}.B} \equiv \overline{\Pi_{x:A}.B} \equiv \flat_{x:\overline{A}}.\overline{B}$.

- Si $A \in \mathcal{T}_\flat$, on définit $A^\lambda \in \mathcal{T}$ par: $s^\lambda \equiv s, x^\lambda \equiv x, (AB)^\lambda \equiv A^\lambda B^\lambda$, $(\flat_{x:A}.B)^\lambda \equiv \lambda_{x:A^\lambda}.B^\lambda$

  Aussi, on définit $[A]$ par $\{A' \text{ in } \mathcal{T} \mid \overline{A'} \equiv A\}$.

- Un contexte Γ est de la forme: $x_1 : A_1, \ldots, x_n : A_n$.
  $\mathrm{DOM}(\Gamma) = \{x_1, \ldots x_n\}$.

- Dans $\mathcal{T}$ on définit: $\overline{\langle\rangle} \equiv \langle\rangle \quad \overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : \overline{A}$.

- Dans $\mathcal{T}_\flat$ on définit: $[\Gamma] \equiv \{\Gamma' \mid \overline{\Gamma'} \equiv \Gamma\}$.

- On a trois systèmes $(\mathcal{T}, \to_\beta)$, $(\mathcal{T}, \to_{\beta\Pi})$ et $(\mathcal{T}_\flat, \to_\flat)$ avec les axiomes:
  - $(\lambda_{x:A}.B)C \to_\beta B[x := C]$
  - $(\flat_{x:A}.B)C \to_\flat B[x := C]$
  - $(\Pi_{x:A}.B)C \to_\Pi B[x := C]$
  - Où $\to_{\beta\Pi} = \to_\beta \cup \to_\Pi$

- Church-Rosser: Soit $r \in \{\beta, \beta\Pi, \flat\}$.
  Si $B_1 \,{}_r\!\!\twoheadleftarrow A \twoheadrightarrow_r B_2$ alors $\exists C$ tel que $B_1 \twoheadrightarrow_r C \,{}_r\!\!\twoheadleftarrow B_2$.

Le β-cube: R_β = Règles de typage avec deux lieurs, $\rightarrow_\beta$ et (appl)

(axiom)	$\langle\rangle \vdash * : \Box$
(start)	$\dfrac{\Gamma \vdash A : s \quad x^s \notin \mathrm{DOM}(\Gamma)}{\Gamma, x^s{:}A \vdash x^s : A}$
(weak)	$\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \quad x^s \notin \mathrm{DOM}(\Gamma)}{\Gamma, x^s{:}C \vdash A : B}$
(Π)	$\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \Pi_{x:A}.B : s_2}$
(λ)	$\dfrac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$
(conv$_\beta$)	$\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$
(appl)	$\dfrac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$

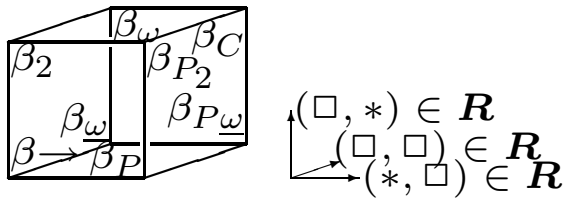


Figure 1: Le β-cube de Barendregt

Le π-cube: $R_\pi = R_\beta \setminus$ (conv$_\beta$) $\cup$ (conv$_{\beta\Pi}$) = Règles de typage avec deux lieurs, $\rightarrow_{\beta\Pi}$ et (appl)

(axiom) (start) (weak) (Π) (λ) (appl)

(conv$_{\beta\Pi}$) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_{\beta\Pi} B'}{\Gamma \vdash A : B'}$$

Le π_i-cube: $R_{\pi_i} = R_\pi \setminus$ (appl) $\cup$ (newappl) = Règles de typage avec deux lieurs, $\rightarrow_{\beta\Pi}$ et (newappl)

(axiom) (start) (weak) (Π) (λ)

$$(\text{conv}_{\beta\Pi}) \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_{\beta\Pi} B'}{\Gamma \vdash A : B'}$$

$$(\text{newappl}) \quad \frac{\Gamma \vdash F : \Pi_{x:A}.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}.B)a}$$

Le $\flat$-cube: $R_\flat$ = Règles de typage avec un seul lieur, $\rightarrow_\flat$ et (appl$\flat$)

(axiom) (start) (weak)

$$(\flat_1)\quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \flat_{x:A}.B : s_2}$$

$$(\flat_2)\quad \frac{\Gamma, x{:}A \vdash b : B \quad \Gamma \vdash \flat_{x:A}.B : s}{\Gamma \vdash \flat_{x:A}.b : \flat_{x:A}.B}$$

$$(\mathsf{conv}_\flat)\quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\flat B'}{\Gamma \vdash A : B'}$$

$$(\mathsf{appl}\flat)\quad \frac{\Gamma \vdash F : \flat x{:}A.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$$

Le $\flat_i$-cube: $R_{\flat_i} = R_\flat \setminus$ (appl$\flat$) $\cup$ (newappl$\flat$) = Règles de typage avec un seul lieur, $\rightarrow_\flat$ et (newappl$\flat$)

(axiom) (start) (weak) ($\flat_1$) ($\flat_2$) (conv$_\flat$)

(newappl$\flat$)
$$\frac{\Gamma \vdash F : \flat x{:}A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\flat x : A.B)a}$$

Six propriétés qu'on désire des systèmes des types

Soit r la relation de réduction dans le système en question.

- *Types correctes (TC)* Si $\Gamma \vdash A : B$ alors $B \equiv \Box$ ou $\Gamma \vdash B : s$ pour $s \in \{*, \Box\}$.
- *Subject reduction (SR)* Si $\Gamma \vdash A : B$ et $A \twoheadrightarrow_r A'$ alors $\Gamma \vdash A' : B$.
- *Préservation des types (PT)* Si $\Gamma \vdash A : B$ et $B \twoheadrightarrow_r B'$ alors $\Gamma \vdash A : B'$.
- *Normalisation forte (NF)* Si $\Gamma \vdash A : B$ alors $\mathsf{SN}_{\rightarrow_r}(A)$ et $\mathsf{SN}_{\rightarrow_r}(B)$.
- *Les sous-termes sont typables (STT)* Si A est $\vdash$-légale et si C est un sous-terme de A alors C est $\vdash$-légale.
- *Unicité des types*
  - *(UT1)* Si $\Gamma \vdash A_1 : B_1$ et $\Gamma \vdash A_2 : B_2$ et $\Gamma \vdash A_1 =_r A_2$, alors $\Gamma \vdash B_1 =_r B_2$.
  - *(UT2)* If $\Gamma \vdash B_1 : s$, $B_1 =_r B_2$ and $\Gamma \vdash A : B_2$ then $\Gamma \vdash B_2 : s$.

Est ce qu'il ya dans les systèmes avec deux lieurs, des Π-redexes dans les termes typables?

- Lemme:
  - Si $\Gamma \vdash_\beta A : B$ ou $\Gamma \vdash_\pi A : B$ alors Γ, A et B ne contiennent pas des Π-redexes.
  - Si $\Gamma \vdash_{\pi_i} A : B$ alors Γ et A ne contiennent pas des Π-redexes et B est le seul Π-redexe possible dans B.

La corespondence entre les β-, π- et π_i-cubes

- Lemme:
  - $\Gamma \vdash_\beta A : B$ si et seulement si $\Gamma \vdash_\pi A : B$
  - Si $\Gamma \vdash_\beta A : B$ alors $\Gamma \vdash_{\pi_i} A : B$.
  - Si $\Gamma \vdash_{\pi_i} A : B$ alors $\Gamma \vdash_\beta A : [B]_\Pi$
    où $[B]_\Pi$ est la Π-forme normale de B.

- Lemme: Le β-cube et le π-cube satisfont les six propriétés qu'on désire des systèmes des types.

Le π_i-cube

- Le π_i-cube perd trois de ces six propriétés:
  Soit $\Gamma = z : *, x : z$. On a que $\Gamma \vdash_{\pi_i} (\lambda_{y:z}.y)x : (\Pi_{y:z}.z)x$.
  - *On n'a pas TC* $(\Pi_{y:z}.z)x \not\equiv \Box$ et $\Gamma \not\vdash_{\pi_i} (\Pi_{y:z}.z)x : s$.
  - *On n'a pas SR* $(\lambda_{y:z}.y)x \rightarrow_{\beta\Pi} x$ mais $\Gamma \not\vdash_{\pi_i} x : (\Pi_{y:z}.z)x$.
  - *On n'a pas UT2* $\vdash_{\pi_i} * : \Box$, $* =_{\beta\Pi} (\Pi_{z:*}.*)\alpha$, $\alpha : * \vdash_{\pi_i} (\lambda_{z:*}.*)\alpha : (\Pi_{z:*}.*)\alpha$ and $\not\vdash_{\pi_i} (\Pi_{z:*}.*)\alpha : \Box$

- Mais on a:
  - *On a UT1*
  - *On a STT*
  - *On a PT*
  - *On a NF*
  - *On a une version faible de TC* Si $\Gamma \vdash_{\pi_i} A : B$ et B *n'est pas un* Π*-redexe* alors ou bien $B \equiv \Box$ ou bien $\Gamma \vdash_{\pi_i} B : s$.
  - *On a une version faible de SR* Si $\Gamma \vdash_{\pi_i} A : B$, B *n'est pas un* Π*-redexe* et $A \twoheadrightarrow_{\beta\Pi} A'$ alors $\Gamma \vdash_{\pi_i} A' : B$.

Si on élimine la distinction entre λ et Π, il restera toujours possible de déduire le rôle de chaque lieur

- Lemme: Supposons que $\Gamma \vdash_\beta A_1 : B_1$ et $\Gamma \vdash_\beta A_2 : B_2$.
  1. Si $\overline{A_1} \equiv \overline{A_2}$ et $B_1 =_\beta B_2$ alors $A_1 \equiv A_2$ et $B_1 \equiv B_2$.
  2. *Dans un terme $\vdash_\beta$-légale de type s, il y a une manière unique de placer les λs et les Πs*
     Si $B_1 \equiv s_1$, $B_2 \equiv s_2$ et $\overline{A_1} \equiv \overline{A_2}$ alors $A_1 \equiv A_2$ et $s_1 \equiv s_2$.
  3. *Dans un contexte $\vdash_\beta$-légale, il y a une manière unique de placer les λs et les Πs*
     Si Γ_1 et Γ_2 sont $\vdash_\beta$-légales et if $\overline{\Gamma_1} \equiv \overline{\Gamma_2}$ alors $\Gamma_1 \equiv \Gamma_2$.
  4. *Dans un type $\vdash_\beta$-légale, il y a une manière unique de placer les λs et les Πs*
     Si $\overline{B_1} \equiv \overline{B_2}$ alors $B_1 \equiv B_2$.
  5. Si $\overline{A_1} \equiv \overline{A_2}$ et $\overline{B_1} \equiv \overline{B_2}$ alors $A_1 \equiv A_2$ et $B_1 \equiv B_2$.
  6. Si $B_1 \equiv s_1$, $B_2 \equiv s_2$, $\overline{A_1} =_\flat \overline{A_2}$ alors $A_1 =_\beta A_2$.

Un algorithme qui transforme un $\flat$-typage en un β-typage

Soit $\Gamma \vdash_\flat A : B$. On définit $(\Gamma \vdash_\flat A : B)^{-1} \in [\Gamma] \times [A] \times [B]$ par:

$$
\begin{array}{ll}
(\langle\rangle \vdash_\flat * : \Box)^{-1} & = (\langle\rangle, *, \Box) \\
(\Gamma, x^s : A \vdash_\flat * : \Box)^{-1} & = (\Gamma', x^s : A', *, \Box) \text{ où } (\Gamma \vdash_\flat A : s)^{-1} = (\Gamma', A', s) \\
(\Gamma \vdash_\flat x^s : C)^{-1} & = (\Gamma', x^s, C') \text{ où } (\Gamma \vdash_\flat C : s)^{-1} = (\Gamma', C', s) \\
(\Gamma \vdash_\flat \flat_{x:A}.B : C)^{-1} & = \begin{cases} (\Gamma', \Pi_{x:A'}.B', C') & \text{si } C =_\flat s_2 \text{ et i.} \\ (\Gamma', \lambda_{x:A'}.B', C') & \text{si } C =_\flat \flat_{x:A}.D \text{ et ii.} \end{cases} \\
(\Gamma \vdash_\flat Fa : C)^{-1} & = (\Gamma', F'a', C') \text{ où iii.}
\end{array}
$$

Où

i, ii, et iii, sont évidents. On écrit simplement la condition i. Pour les autres, voir l'article.

- – $(\Gamma \vdash_\flat A : s_1)^{-1} = (\Gamma', A', s_1)$ pour un s_1 tel que $(s_1, s_2) \in \boldsymbol{R}$,
  i $(\Gamma, x : A \vdash_\flat B : s_2)^{-1} = (\Gamma'', x : A'', B', s_2)$
  – si $C \equiv s_2$ alors $C' \equiv s_2$ sinon
  si $C \not\equiv s_2$ alors $(\Gamma \vdash_\flat C : s)^{-1} = (\Gamma''', C', s)$ pour un certain s.

L'isomorphisme

- Theorème:
  1. si $\Gamma \vdash_\beta A : B$ alors $\overline{\Gamma} \vdash_\flat \overline{A} : \overline{B}$.
  2. si $\Gamma \vdash_\flat A : B$ alors
     - il existe un unique $\Gamma' \in [\Gamma]$ tel que Γ' is $\vdash_\beta$-legal, et
     - il existe un unique $A' \in [A]$, un unique $B' \in [B]$ tel que $\Gamma' \vdash_\beta A' : B'$.

     En plus, Γ', A' et B' sont construits par $^{-1}$ où $(\Gamma \vdash_\flat A : B)^{-1} = (\Gamma', A', B')$.
  3. La vérification des types dans le β-cube est équivalente à la vérification des types dans le $\flat$-cube
  4. La dérivation des types dans le β-cube est équivalente à la dérivation des types dans le $\flat$-cube

Le $\flat$-cube [Kamareddine 2005]

- Le $\flat$-cube a cinq et demi des six propriétés:
  - *On a TC*
  - *On a SR*
  - *On a STT*
  - *On a PT*
  - *On a NF*
  - *On a UT2*
  - *Bien Sûre on n'a pas UT1 (et on ne le veut pas)*.
    En utilisant $(\Box, \Box)$: $\vdash_\beta \lambda_{x:*}.x : \Pi_{x:*}.*$ et $\vdash_\flat \flat_{x:*}.x : \flat_{x:*}.*$.
    En utilisant $(\Box, *)$: $\vdash_\beta \Pi_{x:*}.x : *$ et $\vdash_\flat \flat_{x:*}.x : *$.
    Notez que: $\flat_{x:*}.* \neq_\flat *$.

On a une multiplicité organisée des types dans le $\flat$-cube

- Soient $\mathrm{SN}_{\to_\flat}(B_1)$ et $\mathrm{SN}_{\to_\flat}(B_2)$. On dit que $B_1 \overset{\diamond}{=}_\flat B_2$ ssi $\mathrm{nf}_\flat(B_1) \equiv \flat^{i:1..n_1}_{x_i:F_i}.B$ et $\mathrm{nf}_\flat(B_2) \equiv \flat^{i:1..n_2}_{x_i:F_i}.B$, où $n_1, n_2 \geq 0$.
- *Lemme (MOT)* Si $\Gamma \vdash_\flat A_1 : B_1$ et $\Gamma \vdash_\flat A_2 : B_2$ et $A_1 =_\flat A_2$, alors $B_1 \overset{\diamond}{=}_\flat B_2$.
- Alors le $\flat$-cube est élegant, et a toutes les propriétés (on ne veut plus UT1, MOT suffit.

Example

- Soit $A \equiv \flat_{x_1:*}.\flat_{x_2:\flat_{y:C}.*}.\flat_{x_3:C}.\flat_{x_4:x_2x_3}.x_2x_3$

  Soit $B \in \{\flat_{x_1:*}.\flat_{x_2:\flat_{y:C}.*}.\flat_{x_3:C}.\flat_{x_4:x_2x_3}.*,$
  $\flat_{x_1:*}.\flat_{x_2:\flat_{y:C}.*}.\flat_{x_3:C}.*,$
  $\flat_{x_1:*}.\flat_{x_2:\flat_{y:C}.*}.*,$
  $\flat_{x_1:*}.*,$
  $*\}$.
  On a que $\vdash_\flat A : B$

- Dans le β-cube on a:

$$
\begin{array}{lcll}
\vdash_\beta \lambda_{x_1:*}.\lambda_{x_2:\Pi_{y:C}.*}.\lambda_{x_3:C}.\lambda_{x_4:x_2x_3}.x_2x_3 & : & \Pi_{x_1:*}.\Pi_{x_2:\Pi_{y:C}.*}.\Pi_{x_3:C}.\Pi_{x_4:x_2x_3}. & * \\
\vdash_\beta \lambda_{x_1:*}.\lambda_{x_2:\Pi_{y:C}.*}.\lambda_{x_3:C}.\Pi_{x_4:x_2x_3}.x_2x_3 & : & \Pi_{x_1:*}.\Pi_{x_2:\Pi_{y:C}.*}.\Pi_{x_3:C}. & * \\
\vdash_\beta \lambda_{x_1:*}.\lambda_{x_2:\Pi_{y:C}.*}.\Pi_{x_3:C}.\Pi_{x_4:x_2x_3}.x_2x_3 & : & \Pi_{x_1:*}.\Pi_{x_2:\Pi_{y:C}.*}. & * \\
\vdash_\beta \lambda_{x_1:*}.\Pi_{x_2:\Pi_{y:C}.*}.\Pi_{x_3:C}.\Pi_{x_4:x_2x_3}.x_2x_3 & : & \Pi_{x_1:*}. & * \\
\vdash_\beta \Pi_{x_1:*}.\Pi_{x_2:\Pi_{y:C}.*}.\Pi_{x_3:C}.\Pi_{x_4:x_2x_3}.x_2x_3 & : & & *
\end{array}
$$

  Notons que seulement $\Pi_{x_1:*}.\Pi_{x_2:\Pi_{y:C}.*}.\Pi_{x_3:C}.\Pi_{x_4:x_2x_3}.x_2x_3$ peut être un type.
  Les cinq autres termes ne le peuvent pas.

Merci de votre attention

Bibliography

Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, pages 233–246, 1995.

H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984.

Z.E.A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

Roel Bloo, Fairouz Kamareddine, and Rob Nederpelt. The Barendregt cube with definitions and generalised reduction. *Inform. & Comput.*, 126(2):123–143, May 1996.

N.G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In M. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, pages 29–61, IRIA, Versailles, 1968. Springer Verlag, Berlin, 1970. Lecture Notes in Mathematics **125**; also in [Nederpelt et al., 1994], pages 73–100.

A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

Philippe de Groote. The conservation theorem revisited. In *Proc. Int'l Conf. Typed Lambda Calculi and Applications*, pages 163–178. Springer, 1993.

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, Washington D.C., 1987. IEEE.

J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

F. Kamareddine and R. Bloo. De Bruijn's syntax and reductional behaviour of lambda terms. *Submitted*, 2002.

F. Kamareddine, R. Bloo, and R. Nederpelt. On Π-conversion in the λ-cube and the combination with abbreviations. *Ann. Pure Appl. Logic*, 97(1–3):27–45, 1999.

F. Kamareddine, L. Laan, and R.P. Nederpelt. Refining the Barendregt cube using parameters. In *Proceedings of the Fifth International Symposium on Functional and Logic Programming, FLOPS 2001*, pages 375–389, 2001a.

F. Kamareddine, L. Laan, and R. P. Nederpelt. Types in logic and mathematics before 1940. *Bulletin of Symbolic Logic*, 8(2):185–245, June 2002.

F. Kamareddine, L. Laan, and R. P. Nederpelt. Revisiting the λ-calculus notion of function. *J. Algebraic & Logic Programming*, 54:65–107, 2003.

Fairouz Kamareddine. Postponement, conservation and preservation of strong normalisation for generalised reduction. *J. Logic Comput.*, 10(5):721–738, 2000.

Fairouz Kamareddine and Rob Nederpelt. On stepwise explicit substitution. *Int'l J. Foundations Comput. Sci.*, 4(3): 197–240, 1993.

Fairouz Kamareddine and Rob Nederpelt. Refining reduction in the λ-calculus. *J. Funct. Programming*, 5(4): 637–651, October 1995.

Fairouz Kamareddine and Rob Nederpelt. A useful λ-notation. *Theoret. Comput. Sci.*, 155(1):85–109, 1996.

Fairouz Kamareddine and Alejandro Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *J. Funct. Programming*, 7(4):395–420, 1997.

Fairouz Kamareddine and Alejandro Ríos. A λ-calculus à la de Bruijn with explicit substitution. In *7th Int'l Symp. Prog. Lang.: Implem., Logics & Programs, PLILP '95*, volume 982 of *Lecture Notes in Computer Science*, pages 45–62. Springer, 1995.

Fairouz Kamareddine, Alejandro Ríos, and J. B. Wells. Calculi of generalised β-reduction and explicit substitutions: The type free and simply typed versions. *J. Funct. Logic Programming*, 1998(5), June 1998.

Fairouz Kamareddine, Roel Bloo, and Rob Nederpelt. De Bruijn's syntax and reductional equivalence of lambda terms. In *Proc. 3rd Int'l Conf. Principles & Practice Declarative Programming*, 5–7 September 2001b. ISBN 1-58113-388-X.

A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *Proc. 1994 ACM Conf. LISP Funct. Program.*, pages 196–207, 1994. ISBN 0-89791-643-3.

A. J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of β-strong normalization in typed λ-calculi. In *Proc. 10th Ann. IEEE Symp. Logic in Comput. Sci.*, pages 311–321, 1995. ISBN 0-8186-7050-9. URL `http://www.church-project.org/reports/electronic/Kfo+Wel:LICS-1995.pdf.gz`.

Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *J. ACM*, 41(2):368–398, March 1994.

G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellono University, Pittsburgh, USA, 1988.

Rob Nederpelt. *Strong Normalization in a Typed Lambda Calculus With Lambda Structured Types*. PhD thesis, Eindhoven, 1973.

R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.

L. Regnier. Une équivalence sur les lambda termes. *Theoretical Computer Science*, 126:281–292, 1994.

Laurent Regnier. *Lambda calcul et réseaux*. PhD thesis, University Paris 7, 1992.

G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.

J.C. Reynolds. *Towards a theory of type structure*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.