

TAS

Cours 02 - Lambda-Calcul: Evaluation et Types Simples

Romain Demangeon

51762 (TAS) - M2 STL

27/09/2018

Rappel: λ -calcul

Rappel: Expressivité

Propriétés

Récursion

Stratégies

Types Simples

Section 1

Rappel: λ -calcul

Le λ -calcul

$M, N ::= x \mid \lambda x.M \mid M N$

- ▶ **variables** (ensemble infini de "noms"),
- ▶ **abstraction** (fonctions anonymes, $x \mapsto M$)
- ▶ **application** (comme en ML, M "fonction", N "argument")

α -conversion

- ▶ variable **liée** (par un λ): $\lambda x.((x z) \lambda y.(x y))$
- ▶ variable **libre** (non liée): $\lambda x.((x z) \lambda y.(x y))$
- ▶ renommage des variables **liées** $\lambda x.M = \lambda y.(M[y/x])$ si $y \notin M$.
- ▶ **Convention de Barendregt**: variable liées différentes **deux à deux** et différentes des **variables libres**.

Eviter, par exemple, $(\lambda x.x) (\lambda x.x x)$ ou $(\lambda x.x) x$

Substitution

- ▶ $M[N/x]$: terme M où l'on a remplacé **chaque occurrence** de x par N
- ▶ $(M[N_1/x_1])[N_2/x_2] = (M[N_2/x_2])[N_1[N_2/x_2]/x_1]$
- ▶ opération magique **instantanée**.
 - ▶ **implémentation ?** λ -calcul avec **substitutions explicites**.

$$(\beta) \frac{}{(\lambda x.M) N \longrightarrow M[N/x]}$$

$$\frac{N \longrightarrow N'}{M N \longrightarrow M N'}$$

$$\frac{M \longrightarrow M'}{M N \longrightarrow M' N}$$

$$\frac{M \longrightarrow M'}{\lambda x.M \longrightarrow \lambda x.M'}$$

Premier Exemple

$$\begin{aligned} & (\lambda x.x x) (\lambda y.y) \\ \longrightarrow & (x x)[(\lambda y.y)/x] \\ = & (\lambda y.y) (\lambda y.y) \\ =_{\alpha} & (\lambda x.x) (\lambda y.y) \\ \longrightarrow & (\lambda y.y) \end{aligned}$$

Non-Déterminisme

$$\begin{aligned} & (\lambda x.x x) ((\lambda y.y) (\lambda z.z)) \\ \longrightarrow & ((\lambda y.y) (\lambda z.z)) ((\lambda u.u) (\lambda v.v)) \\ \longrightarrow & \dots \end{aligned}$$

ou

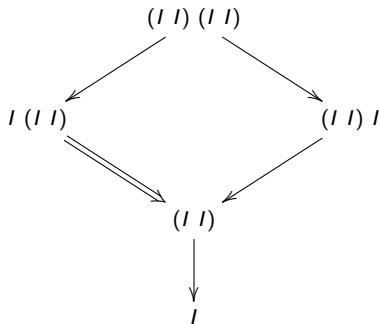
$$\begin{aligned} & (\lambda x.x x) ((\lambda y.y) (\lambda z.z)) \\ \longrightarrow & (\lambda x.x x) (\lambda z.z) \\ \longrightarrow & \dots \end{aligned}$$

Notations

- ▶ $\lambda xy.M$ pour $\lambda x.(\lambda y.M)$ (regroupement des paramètres)
 - ▶ $M_1 M_2 M_3$ pour $(M_1 M_2) M_3$ (parenthésage à gauche)
 - ▶ $\lambda x.M N$ pour $\lambda x.(M N)$ (maximalité du λ)
-
- ▶ $I = \lambda x.x$ (identité)
 - ▶ $K = \lambda xy.x$ (sélection gauche ou T)
 - ▶ $F = \lambda xy.y$ (sélection droite ou \perp)
 - ▶ $S = \lambda xyz.(x z) (y z)$ (application généralisée)
 - ▶ $\delta = \lambda x.(x x)$ (auto-application)
 - ▶ $\Omega = \delta \delta$ (divergence)
 - ▶ $Y = \lambda f.(\lambda x.f (x x)) (\lambda x.f (x x))$ (combinateur paradoxal)

Définition

Un **graphe de réduction** est un multi-graphe dirigé connexe où les **sommets** sont des **termes**. Il y a une **arête** de M vers N pour chaque manière d'obtenir $M \rightarrow N$.



Graphes de Réduction (II)

$$K \text{ I } \Omega \longrightarrow (\lambda y. I) \Omega \longrightarrow I$$

avec $Tr = \lambda x.(x \ x \ x)$

$$Tr \ Tr \longrightarrow Tr \ Tr \ Tr \longrightarrow Tr \ Tr \ Tr \ Tr \longrightarrow \dots$$

$$\begin{array}{ccccccc} (\lambda x. I) (Tr \ Tr) & \longrightarrow & (\lambda x. I) (Tr \ Tr \ Tr) & \longrightarrow & (\lambda x. I) (Tr \ Tr \ Tr \ Tr) & \longrightarrow & \dots \\ \downarrow & & \swarrow & & \swarrow & & \\ I & & \swarrow & & \swarrow & & \end{array}$$

Section 2

Rappel: Expressivité

Encodage

$$(x, y) \stackrel{\text{def}}{=} \lambda f. f \ x \ y$$

- ▶ $(., .) = \lambda a b f. f \ a \ b$ (encouplage)
- ▶ $\Pi_2^1 = \lambda c. (c \ K)$ (projection gauche)
- ▶ $\Pi_2^2 = \lambda c. (c \ F)$ (projection droite)
- ▶ $\leftrightarrow = \lambda c f. f \ (\Pi_2^2 \ c) \ (\Pi_2^1 \ c)$ (échange)

Encodage

$n \stackrel{\text{def}}{=} \lambda fe.f (f \dots (f e)\dots)$ (n applications de f à e)

- ▶ $0 = F$ (zéro)
- ▶ $1 = \lambda fe.(f e)$ (unité)
- ▶ $add = \lambda nmfe.n f (m f e)$ (addition)
- ▶ $mult = \lambda nmfe.n (m f) e$ (multiplication)
- ▶ $power = \lambda nmfe.(m n) f e$ (puissance)
- ▶ pour $pred$ il faut définir $\sigma : (x, y) \mapsto (x + 1, x)$ en utilisant les couples.

Encodage

$$[e_1; e_2; \dots; e_k] \stackrel{\text{def}}{=} \lambda cn. c e_1 (c e_2 (\dots (c e_k n) \dots))$$

- ▶ $[] = 0$ (liste vide)
- ▶ $[.] = \lambda ecn. c e n$ (encapsulation)
- ▶ $cons = \lambda elcn. c e (l c n)$ (construction)
- ▶ $append = \lambda l_1 l_2 cn. (l_1 c (l_2 c n))$ (concaténation)
- ▶ $hd = \lambda L. L K 0$ (tête)
- ▶ pour tl (queue) il faut définir la fonction $\sigma : a, (x, y) \mapsto ((cons a y), y)$
- ▶ $map = \lambda flcn. l (\lambda as. (c (f a) s)) n$
- ▶ $filter = \lambda plcn. l (\lambda as. (p a) (c a s) s) n$
- ▶ $reduce = \lambda fal. l f a$

arbres

$$(e, A_g, A_d) \stackrel{\text{def}}{=} \lambda cn. c e A_g A_d$$

Modèles de **calcul** au 20ème siècle:

- ▶ **Machines de Turing**: automate + rubans + lecture/écriture,
- ▶ **Fonctions récursives**: schéma de définition de fonctions,
- ▶ **λ -calcul**: syntaxe épurée fonctionnelle.

Forme physique (prouvée)

Le **λ -calcul**, les **machines de Turing** et les **fonctions récursives** ont la même expressivité.

Forme psychologique (conjecturée)

Le **cerveau humain** aussi.

Substitutions

- ▶ écrire $M[N/x]$ c'est **triché**.
- ▶ $\lambda_{\sigma\uparrow}$: calcul **implémentant** les substitutions de manière explicite
 - ▶ on descend dans le terme avec l'argument et on remplace au bon endroit.

α -conversion

- ▶ le λ -calcul est trop **syntaxique**.
 - ▶ la notion de variable est **décevante**.
 - ▶ difficulté d'**implémentation**
- ▶ λ avec **indices** de De Bruijn: $M, N ::= n \mid \lambda.M \mid M N$
 - ▶ le **nombre** de λ à traverser pour trouver le λ liant.
 - ▶ $K = \lambda\lambda.1$ et $S = \lambda\lambda\lambda.(2\ 0) (1\ 0)$
- ▶ Réseaux d'interaction.

Section 3

Propriétés

Un modèle de calcul est **non-déterministe** quand **deux réductions** aboutissant à deux termes **différents** sont possibles depuis un **même** terme.

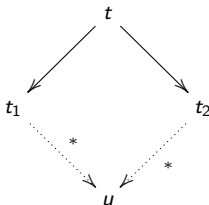
$$\blacktriangleright \exists t, t_1, t_2. (t \longrightarrow t_1) \wedge (t \longrightarrow t_2) \wedge (t_1 \neq t_2)$$

- ▶ langages **usuels**: déterministes,
- ▶ **machines** de Turing: non-déterministes (cf. P & NP),
- ▶ **réseaux de Petri** et **CCS**: non-déterministes.
- ▶ **λ -calcul**: non-déterministe
 - ▶ $(I I) (I I) \longrightarrow I (I I)$
 - ▶ $(I I) (I I) \longrightarrow (I I) I$
- ▶ un "endroit où l'on peut réduire" est appelé **redex**.
- ▶ on a parfois le choix entre **plusieurs** redex.

\longrightarrow^* : cloture **réflexive transitive** de \longrightarrow (i.e. \longrightarrow^n pour un certain $n \geq 0$)

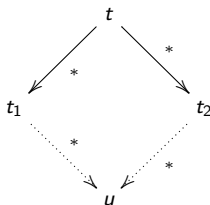
Un modèle de calcul est **localement confluente** quand deux réductions aboutissant à deux termes **différents** sont **joignables** en 0 ou plus réductions.

► $\forall t, t_1, t_2. (t \longrightarrow t_1) \wedge (t \longrightarrow t_2) \Rightarrow \exists u. (t_1 \longrightarrow^* u) \wedge (t_2 \longrightarrow^* u)$



Un modèle de calcul est **confluent** quand deux séries de réductions aboutissant à deux termes **différents** sont **joignables** en 0 ou plus réductions.

- ▶ $\forall t, t_1, t_2. (t \longrightarrow^* t_1) \wedge (t \longrightarrow^* t_2) \Rightarrow \exists u. (t_1 \longrightarrow^* u) \wedge (t_2 \longrightarrow^* u)$



Différence entre les deux notions

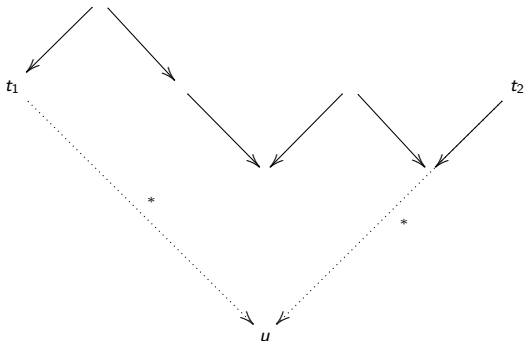


Church-Rosser

- ▶ $u \equiv t$ (équivalence) quand il existe une suite $(u_i)_{0 \leq i \leq k}$ avec $u_0 = u$ et $u_k = t$ telle que, $\forall i < k, (u_i \longrightarrow u_{i+1}) \vee (u_{i+1} \longrightarrow u_i)$.
- ▶ $u \equiv t$ quand on peut utiliser \longrightarrow dans les deux sens pour les relier.

Un modèle de calcul est **Church-Rosser** quand deux termes **équivalents** sont **joignables** en **0 ou plus** réductions.

- ▶ $\forall t, t_1, t_2. (t_1 \equiv t_2) \Rightarrow \exists u. (t_1 \longrightarrow^* u) \wedge (t_2 \longrightarrow^* u)$



- ▶ le λ -calcul est Church-Rosser.
- ▶ implication pratique: pas de concurrence.
 - ▶ "aucun choix n'est définitif"
- ▶ nombre de réductions:
 - ▶ $K I$ (power 10 2 I I) $\longrightarrow \longrightarrow I$,
 - ▶ $K I$ (power 10 2 I I) $\longrightarrow^{(N)} I$ (avec N très grand).
- ▶ "vu qu'on est confluent, autant réduire au plus rapide".

Une **forme normale** n d'un terme t est un **réduit** de t qui **ne** se réduit **pas**

▶ $(t \longrightarrow^* n) \wedge \neg(\exists s, n \longrightarrow s)$

▶ I est une forme normale de $S K K$.

Un terme est **faiblement normalisant** s'il **admet** une forme normale.

- ▶ $S K K$ est faiblement normalisant.
- ▶ $K I \Omega$ est faiblement normalisant.
- ▶ Ω n'est pas faiblement normalisant.

Un modèle de calcul est **faiblement normalisant** si tous les termes sont **faiblement normalisants**.

▶ le λ -calcul n'est **pas** faiblement normalisant.

- ▶ soit un terme t qui a deux formes normales n_1 et n_2 ,
- ▶ comme $t \longrightarrow^* n_1$ et $t \longrightarrow^* n_2$, par confluence, il existe n tel que $n_1 \longrightarrow^* n$ et $n_2 \longrightarrow^* n$,
- ▶ comme n_1 est une forme normale $n_1 = n$,
- ▶ comme n_2 est une forme normale $n_2 = n$,
- ▶ finalement $n_1 = n_2$.

Les termes du λ -calcul ont au plus une forme normale.

Un terme est **fortement normalisant** (terminant) s'il n'existe pas de suite **infinie** de réductions depuis ce terme.

▶ $\exists (t_n)_{n \in \mathbb{N}}. (t_0 = t) \wedge (\forall k \in \mathbb{N}, t_k \longrightarrow t_{k+1})$

- ▶ $S K K$ est fortement normalisant.
- ▶ Ω n'est pas fortement normalisant.
- ▶ $K I \Omega$ n'est pas fortement normalisant.

Un modèle de calcul est **fortement normalisant** (terminant) si tous les termes sont **fortement normalisants**.

- ▶ le λ -calcul n'est **pas** fortement normalisant (divergent).

Section 4

Réursion

- ▶ λ -calcul **divergent**:
 - ▶ comment exprimer la **réursion** ?
 - ▶ nécessaire pour la **Turing-complétude**.

Un **combinateur de point fixe** *fix* est une fonction(nelle), qui quand elle est **appliquée** à une fonction *F*, donne un **point fixe** de *F*.

- ▶ $F (fix F) \equiv (fix F)$
- ▶ **plusieurs** combinateurs de points fixe en λ -calcul.
- ▶ $Y = \lambda f.(\lambda x.f (x x)) (\lambda x.f (x x))$
 - ▶ vérifier que $(Y F) \equiv F (Y F)$.
- ▶ *Y* permet de **programmer** des fonctions **récurives**.

- ▶ $isZero = \lambda n.n (\lambda y.F) K$ (égalité avec 0)
- ▶ $F = \lambda fn.(isZero n) 1 (mult n (f (pred n)))$ (fonctionnelle)
- ▶ $fact = Y F$ (factorielle, point fixe de la fonctionnelle)

$$\begin{aligned} & fact\ 3 \\ = & (Y\ F)\ 3 \\ \equiv & F\ (Y\ F)\ 3 \\ \equiv & (isZero\ 3)\ 1\ (mult\ 3\ ((Y\ F)\ (pred\ 3))) \\ \equiv & (mult\ 3\ ((Y\ F)\ 2)) \\ \equiv & (mult\ 3\ ((isZero\ 2)\ 1\ (mult\ 2\ ((Y\ F)\ (pred\ 2))))) \\ \equiv & (mult\ 3\ (mult\ 2\ ((Y\ F)\ 1))) \end{aligned}$$

Section 5

Stratégies

λ -calcul pur

$M, N ::= x \mid \lambda x.M \mid M N$

$$(\beta) \frac{}{(\lambda x.M) N \longrightarrow M[N/x]}$$

$$\frac{N \longrightarrow N'}{M N \longrightarrow M N'}$$

$$\frac{M \longrightarrow M'}{M N \longrightarrow M' N}$$

$$\frac{M \longrightarrow M'}{\lambda x.M \longrightarrow \lambda x.M'}$$

- ▶ Syntaxe et sémantique du λ -calcul **pur**.
- ▶ Modèle de programmation fonctionnelle **réaliste** ?
 - ▶ **non-déterminisme**.
 - ▶ **réduction** sous les λ .

Une **stratégie** est un ensemble de règle de réduction tel qu'un terme à **au plus un** réduit.

- ▶ une stratégie est une sous-relation déterministe de \longrightarrow .

Appel par Valeur de Gauche à Droite (LtR-CbV)

$$M, N ::= x \mid \lambda x.M \mid M N$$
$$V ::= x \mid \lambda x.M \mid x V$$

$$(\beta) \frac{}{(\lambda x.M) V \longrightarrow M[V/x]} \qquad \frac{M \longrightarrow M'}{M N \longrightarrow M' N}$$

$$\frac{N \longrightarrow N'}{V N \longrightarrow V N'}$$

- ▶ Syntaxe des **valeurs**: termes qu'on ne peut pas réduire.
 - ▶ variable, abstraction, ou application bloquée.
- ▶ β -réduction uniquement si l'argument **est une valeur**.
- ▶ On réduit **d'abord** la fonction et après **l'argument**.
- ▶ On ne réduit pas **sous le λ** .
- ▶ C'est la **stratégie** de la plupart des **langages de programmation**.

Appel par Nom (CbN)

$M, N ::= x \mid \lambda x.M \mid M N$

$$(\beta) \frac{}{(\lambda x.M) N \longrightarrow M[N/x]}$$

$$\frac{M \longrightarrow M'}{M N \longrightarrow M' N}$$

- ▶ Syntaxe **classique**.
- ▶ β -réduction **classique**.
- ▶ On ne peut **pas** réduire **l'argument**.
- ▶ On ne réduit pas **sous le λ** .
- ▶ C'est la **stratégie** de *Haskell* (stratégie paresseuse).

- ▶ programmer **par continuation**, c'est manipuler **explicitement** le **futur** du résultat
 - ▶ il est passé en **paramètre**.

```
let add x y k = k (x + y)
```

```
let n = add 1 2 (fun x -> add x 3 (fun y -> y))
```

- ▶ Programmer par continuation permet de manipuler **l'environnement**.
 - ▶ dans certain langage: `call/cc`

$$\begin{aligned}[x] &= x \\ [\lambda x.M] &= \lambda x.[M] \\ \llbracket V \rrbracket &= \lambda k.k [V] \\ \llbracket M N \rrbracket &= \lambda k.\llbracket M \rrbracket (\lambda a.\llbracket N \rrbracket (\lambda b.a b k))\end{aligned}$$

- ▶ ▶ $\llbracket \delta I \rrbracket I$
 - ▶ $= (\lambda k.\llbracket \delta \rrbracket (\lambda a.\llbracket I \rrbracket (\lambda b.a b k))) I$
 - ▶ $\longrightarrow \llbracket \delta \rrbracket (\lambda a.\llbracket I \rrbracket (\lambda b.a b I))$
 - ▶ $= (\lambda k.k[\delta]) (\lambda a.\llbracket I \rrbracket (\lambda b.a b I))$
 - ▶ $\longrightarrow (\lambda a.\llbracket I \rrbracket (\lambda b.a b I))[\delta]$
 - ▶ $\longrightarrow (\llbracket I \rrbracket (\lambda b.[\delta] b I))$
 - ▶ $\longrightarrow (\lambda k.k[I]) (\lambda b.[\delta] b I)$
 - ▶ $\longrightarrow (\lambda b.[\delta] b I) [I]$
 - ▶ $\longrightarrow ([\delta] [I] I)$
 - ▶ ...
- ▶ Encodage **par continuation**: implémente la stratégie **CbV-LtR**.

- ▶ tout terme M s'écrit

$$\lambda x_1 x_2 \dots x_k . M_1 M_2 \dots M_n$$

avec $k \geq 0$, $n > 0$ et M_1 une **variable** ou une **abstraction**

- ▶ (et si M_1 est une abstraction $n > 1$).
- ▶ si M_1 est une variable on dit que M est **en forme normale de tête**.
- ▶ sinon $M_1 = \lambda x . N_1$ et $(\lambda x . N_1 M_2)$ est le **redex de tête**.
- ▶ la **réduction standard** est la **stratégie** qui
 1. réduit le **redex de tête** s'il existe.
 2. **applique** la réduction standard aux $M_2 \dots M_n$ si M est **en forme normale de tête**.
- ▶ la réduction standard **termine** si M a un **forme normale**.

Section 6

Types Simples

- ▶ Certains termes paraissent "étranges" du point de vue de la programmation:
 - ▶ $\delta = \lambda x.x$: on applique un programme à lui-même.
 - ▶ $\lambda fe.(f e)$ ($e f$)
- ▶ On pourrait essayer d'éliminer de tels termes.
- ▶ Le **typage** peut permettre de discriminer des termes bien-formés.

$$T, S ::= \alpha \mid T \longrightarrow S$$

- ▶ α : **variables** de type (différentes des variables de termes),
- ▶ $T \longrightarrow S$: type **fonctionnel**, T paramètre, S résultat,
- ▶ exemples:
 - ▶ $\alpha \longrightarrow \alpha$,
 - ▶ $\alpha \longrightarrow (\beta \longrightarrow \alpha)$
- ▶ parenthésage **à droite**: $\alpha \longrightarrow \beta \longrightarrow \gamma = \alpha \longrightarrow (\beta \longrightarrow \gamma)$

Environnement

Un **environnement** de typage Γ est une fonction partielle des **variables** de terme dans les **types**.

▶ $\Gamma = x_1 : T_1, \dots, x_n : T_n$

- ▶ On peut modifier l'**ordre** de Γ .

Jugement

Le **jugement** de typage $\Gamma \vdash M : T$ indique que, sous l'environnement Γ , le terme M est typable avec le type T .

- ▶ $\emptyset \vdash I : \alpha \longrightarrow \alpha$
- ▶ $\emptyset \vdash I : (\alpha \longrightarrow \alpha) \longrightarrow (\alpha \longrightarrow \alpha)$
- ▶ $\emptyset \vdash K : \alpha \longrightarrow \beta \longrightarrow \alpha$
- ▶ $x : \alpha, y : \alpha \longrightarrow \beta \vdash \lambda z.z (y x) : (\beta \longrightarrow \gamma) \longrightarrow \gamma$

$$\begin{array}{c} \text{(Var)} \frac{}{\Gamma, x : T \vdash x : T} \\ \text{(Abs)} \frac{\Gamma, x : T \vdash M : S}{\Gamma \vdash \lambda x. M : T \rightarrow S} \\ \text{(App)} \frac{\Gamma \vdash M : S \rightarrow T \quad \Gamma \vdash N : S}{\Gamma \vdash M N : T} \end{array}$$

- ▶ typer une **variable**: elle doit apparaître dans l'environnement,
- ▶ typer une **abstraction**: un type $T \rightarrow S$ si le paramètre à un type T et le corps de la fonction un type S
- ▶ typer une **application**: M doit avoir un type fonctionnel et N le type du paramètre de M , et $M N$ a le type du résultat.
- ▶ de haut en bas: **déduction**.
- ▶ de bas en haut: **inférence**.

- **typage** de $K \ I \ I = (\lambda z_1 z_2. z_1) (\lambda y. y) (\lambda x. x)$

$$\begin{array}{c}
 \text{(Var)} \frac{}{z_1 : \alpha \rightarrow \alpha, z_2 : \alpha \rightarrow \alpha \vdash z_1 : \alpha \rightarrow \alpha} \\
 \text{(Abs)} \frac{}{z_1 : \alpha \rightarrow \alpha \vdash \lambda z_2. z_1 : (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)} \\
 \text{(Abs)} \frac{}{\emptyset \vdash K : (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)} \\
 \text{(App)} \frac{}{\emptyset \vdash K \ I : (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)} \\
 \text{(Var)} \frac{}{y : \alpha \vdash y : \alpha} \\
 \text{(Abs)} \frac{}{\emptyset \vdash I : \alpha \rightarrow \alpha} \\
 \text{(Abs)} \frac{}{x : \alpha \vdash x : \alpha} \\
 \text{(Abs)} \frac{}{\emptyset \vdash I : \alpha \rightarrow \alpha} \\
 \hline
 \emptyset \vdash K \ I \ I : \alpha \rightarrow \alpha
 \end{array}$$

- ▶ on part du **terme à typer**,
- ▶ on crée des **contraintes** de typage en remontant la **dérivation**,
- ▶ on **unifie** les contraintes de typage:
 - ▶ **succes**: on récupère le type le plus général,
 - ▶ **échec**: le terme n'est pas typable.

$$\frac{
 \begin{array}{c}
 \text{(Var)} \frac{}{z_1 : T_4, z_2 : T_1 \vdash z_1 : T_0} \\
 \text{(Abs)} \frac{}{z_1 : T_4 \vdash \lambda z_2. z_1 : T_1 \rightarrow T_0} \\
 \text{(Abs)} \frac{}{\emptyset \vdash K : T_4 \rightarrow T_1 \rightarrow T_0} \\
 \text{(App)} \frac{}{\emptyset \vdash K I I : T_1 \rightarrow T_0}
 \end{array}
 \quad
 \begin{array}{c}
 \text{(Var)} \frac{}{y : T_5 \vdash y : T_6} \\
 \text{(Abs)} \frac{}{\emptyset \vdash I : T_4}
 \end{array}
 \quad
 \begin{array}{c}
 \text{(Var)} \frac{}{x : T_2 \vdash x : T_3} \\
 \text{(Abs)} \frac{}{\emptyset \vdash I : T_1}
 \end{array}
 }{
 \emptyset \vdash K I I : T_0
 }$$

$$\{ T_1 = T_2 \rightarrow T_3, T_2 = T_3, T_4 = T_5 \rightarrow T_6, T_5 = T_6, T_0 = T_4 \}$$

s'unifie en

$$T_0 = T_4 = \alpha \longrightarrow \alpha, T_1 = \beta \longrightarrow \beta, T_2 = T_3 = \alpha, T_5 = T_6 = \beta$$

- ▶ On explore le terme en **construisant** la dérivation.
 - ▶ les (**Var**) **produisent** des contraintes,
 - ▶ les (**Abs**) **produisent** des contraintes et **gènèrent** des variables,
 - ▶ les (**App**) **gènèrent** des variables.
- ▶ Algorithmes d'**unification**:
 - ▶ Martelli-Montanari, W, Hindley-Milner.

- ▶ **typage** de $\delta = \lambda x.x x$
 - ▶ $\emptyset \vdash \delta : T_0$
 - ▶ **(Abs)**: $x : T_1 \vdash x x : T_2$ et $T_0 = T_1 \longrightarrow T_2$
 - ▶ **(App)**: (1) $x : T_1 \vdash x : T_3 \longrightarrow T_2$, (2) $x : T_1 \vdash x : T_3$
 - ▶ **(Var)** pour (1), on a $T_1 = T_3 \longrightarrow T_2$
 - ▶ **(Var)** pour (2), on a $T_1 = T_3$
 - ▶ l'unification **explode** devant $T_3 = T_3 \longrightarrow T_2$

Réduction assujettie

Si $\Gamma \vdash M : T$ et $M \longrightarrow M'$, alors $\Gamma \vdash M' : T$.

Terminaison

Si $\Gamma \vdash M : T$, alors M est **fortement normalisant**.

- ▶ le λ -calcul simplement typé **termine**.
 - ▶ entre autres, Ω n'est pas typable.
- ▶ plusieurs **preuves**:
 - ▶ relation logiques (interprétation des types), David, Gandi.
- ▶ la réduction est **élémentaire** en la taille du terme.

On dit que le type T est **habité** s'il existe M tel que $\Gamma \vdash M : T$.

- ▶ $\alpha \rightarrow \alpha$ est **habité** par I ,
- ▶ $\alpha \rightarrow \beta$ n'est **pas** habité,
- ▶ $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$ est **habité** (par $\lambda xy.x y$).
- ▶ $(\alpha \rightarrow \beta) \rightarrow \beta$ n'est **pas** habité.
- ▶ "on doit utiliser les types des paramètres pour produire le type résultat".

Correspondance de Curry-Howard

- ▶ les **types simples** sont les **formules** de la logique propositionnelle.
 - ▶ un **type** est **habité** si la **formule** correspondante est **prouvable** en logique intuitionniste.
 - ▶ un **terme** est une **preuve**: la **dérivation de typage** du terme est la **dérivation de preuve**.
 - ▶ la **réduction** d'un terme est l'**élimination des coupures** dans la preuve correspondante.
-
- ▶ "La **Programmation** c'est de la **Logique**" (et vice-versa),
 - ▶ construction de *theorem provers* (*Coq*, *Agda*, *Isabelle*)
 - ▶ autre correspondance: logique **classique** et **call/cc** ($\lambda_{\mu\tilde{\mu}}$)

Prochain Cours

- ▶ **Polymorphisme**: $\emptyset \vdash \lambda x.x x : (\forall \alpha. \alpha \longrightarrow \alpha) \longrightarrow (\forall \alpha. \alpha \longrightarrow \alpha)$
- ▶ **Inférence** de ML (Hindley-Milner).
- ▶ Typage des opérateurs **impératifs**.

En TD

Typer $S K K$.