TAS
Cours 04 - De λ_{ST} vers un langage fonctionnel typé.

Romain Demangeon

TAS - M2 STL

03/10/2024

De λ vers ML

- En partant de λ, on peut se diriger vers un langage fonctionnel complet.
- En ajoutant des types de bases.
- En ajoutant les types sommes et les types produits.
- En ajoutant des définitions de fonction récursives.
- En ajoutant du polymorphisme.
- En ajoutant d'autres traits :
  - traits impératifs,
  - traits concurrents,
  - traits exceptionnels.

Types de Base

- On peut ajouter des entiers natifs (par exemple) à la syntaxe :

$$M ::= x \mid M\ M \mid \lambda x.M \mid n$$

avec $n \in \mathbb{N}$.

- Avec un type et une règle de typage adéquats :

$$T ::= \alpha \mid T \to T \mid \mathbf{N} \qquad (\mathbf{Nat})\frac{n \in \mathbb{N}}{\Gamma \vdash n : \mathbf{N}}$$

Types de Base (II)

- Problème : on ne peut rien faire des entiers.
- il faut des primitives pour les manipuler.
  - par exemple `S`, `P` et `ifz`.
  - qu'il faut typer :

$$M ::= x \mid M\ M \mid \lambda x.M \mid n \mid \texttt{ifz}\ M\ M\ M \mid \texttt{S}\ M \qquad (\textbf{succ})\frac{}{\texttt{S}\ n \longrightarrow n+1}$$

$$(\texttt{pred})\frac{}{\texttt{P}\ n \longrightarrow n-1} \qquad (\texttt{ifz0})\frac{M \longrightarrow M'}{\texttt{ifz}\ M\ N_1\ N_2 \longrightarrow \texttt{ifz}\ M'\ N_1\ N_2}$$

$$(\texttt{ifz1})\frac{}{\texttt{ifz}\ 0\ M\ N \longrightarrow M} \qquad (\texttt{ifz2})\frac{n \neq 0}{\texttt{ifz}\ n\ M\ N \longrightarrow M}$$

$$(\texttt{Su})\frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash \texttt{S}\ M : \mathbf{N}} \qquad (\texttt{Pr})\frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash \texttt{P}\ M : \mathbf{N}}$$

$$(\texttt{Ifz})\frac{\Gamma \vdash M : \mathbf{N} \qquad \Gamma \vdash N_1 : T \qquad \Gamma \vdash N_2 : T}{\Gamma \vdash \texttt{ifz}\ M\ N_1\ N_2 : T}$$

- Notez l'absence de $\dfrac{N_1 \longrightarrow N_1'}{\texttt{ifz}\ M\ N_1\ N_2 \longrightarrow \texttt{ifz}\ M\ N_1'\ N_2}$
- Similaire (primitives, typage des primitives) pour d'autres types de base.

SORBONNE UNIVERSITÉ

Types de Base : Inférence

- le système est toujours dirigé par la syntaxe.
- la règle (**Nat**) génère une équation $T_i = \mathbf{N}$.
- les autres règles fixent un type **N** dans la génération récursive d'équations
  - et peuvent générer $T_i = \mathbf{N}$ (par exemple (**Su**))
- lors de l'unification :
  - $\mathbf{N} = \mathbf{N}$ est éliminée.
  - $\mathbf{N} = T_1 \to T_2$ fait échouer l'inférence.

Rappel : Types Produits

- On ajoute des déconstructeurs de couples à la syntaxe.

$$M ::= x \mid M\ M \mid \lambda x.M \mid (M, M) \mid \Pi_1\ M \mid \Pi_2\ M$$

- qui permettent de projeter un couple

$$(\textbf{ProG})\frac{M \longrightarrow M'}{(M, N) \longrightarrow (M', N)} \qquad (\textbf{ProD})\frac{N \longrightarrow N'}{(M, N) \longrightarrow (M, N')}$$

$$(\textbf{PjG})\frac{}{\Pi_1\ (M, N) \longrightarrow M} \qquad (\textbf{PjD})\frac{}{\Pi_2\ (M, N) \longrightarrow M}$$

- et que l'on doit typer

$$(\textbf{Pro})\frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : U}{\Gamma \vdash (M, N) : T \times U} \qquad (\textbf{PjG})\frac{\Gamma \vdash M : T \times U}{\Gamma \vdash \Pi_1\ M : T}$$

$$(\textbf{PjD})\frac{\Gamma \vdash M : T \times U}{\Gamma \vdash \Pi_2\ M : U}$$

Types Produits : Exemple

$$
\begin{array}{rl}
 & (\lambda c.(\Pi_1\ c)\ (\Pi_2\ c))\ (I, I\ K) \\
\longrightarrow & (\Pi_1\ (I, I\ K))\ (\Pi_2\ (I, I\ K)) \\
\longrightarrow & I\ (\Pi_2\ (I, I\ K)) \\
\longrightarrow & I\ (I\ K) \\
\longrightarrow & (I\ K) \\
\longrightarrow & K
\end{array}
$$

On pose $T_k = \alpha \to \beta \to \alpha$.

$$
(\textbf{App})\ \frac{\dfrac{\cdots}{\vdash \lambda c.(\Pi_1\ c)\ (\Pi_2\ c) : ((T_K \to T_K) \times T_K) \to T_K} \qquad \dfrac{\cdots}{\vdash (I, I\ K) : (T_K \to T_K) \times T_K}}{\vdash (\lambda c.(\Pi_1\ c)\ (\Pi_2\ c))\ (I, I\ K) : T_k}
$$

$$
(\textbf{Pro})\ \frac{\dfrac{\langle \text{TD2} \rangle}{\vdash I : T_K \to T_K} \qquad (\textbf{App})\ \dfrac{\dfrac{\langle \text{TD2} \rangle}{\vdash I : T_K \to T_K} \qquad \dfrac{\langle \text{TD2} \rangle}{\vdash K : T_K}}{\vdash I\ K : T_K}}{\vdash (I, I\ K) : (T_K \to T_K) \times T_K}
$$

$$
(\textbf{Abs})\ \frac{(\textbf{App})\ \dfrac{(\textbf{PrG})\ \dfrac{(\textbf{Var})\ \dfrac{}{\Gamma \vdash c : ((T_K \to T_K) \times T_K)}}{\Gamma \vdash (\Pi_1\ c) : T_K \to T_K} \qquad (\textbf{PrD})\ \dfrac{(\textbf{Var})\ \dfrac{}{\Gamma \vdash c : ((T_K \to T_K) \times T_K)}}{\Gamma \vdash (\Pi_1\ c) : T_K}}{c : ((T_K \to T_K) \times T_K) \vdash (\Pi_1\ c)\ (\Pi_2\ c) : T_K}}{\vdash \lambda c.(\Pi_1\ c)\ (\Pi_2\ c) : ((T_K \to T_K) \times T_K) \to T_K}
$$

SORBONNE UNIVERSITÉ

Rappel : Types Sommes

- Le destructeur lie une variable.

$$M ::= x \mid M\ M \mid \lambda x.M \mid \mathtt{g} : M \mid \mathtt{d} : M \mid \mathtt{sw}\ M\ \triangleright x : M + M$$

- On branche en fonction du côté du terme.

$$(\mathtt{SumG})\frac{M \longrightarrow M'}{\mathtt{g} : M \longrightarrow \mathtt{g} : M'} \qquad (\mathtt{SumD})\frac{N \longrightarrow N'}{\mathtt{d} : N \longrightarrow \mathtt{g} : N'}$$

$$(\mathtt{SwG})\frac{}{\mathtt{sw}\ \mathtt{g} : M\ \triangleright x : N_1 + N_2 \longrightarrow N_1[M/x]}$$

$$(\mathtt{SwG})\frac{}{\mathtt{sw}\ \mathtt{d} : M\ \triangleright x : N_1 + N_2 \longrightarrow N_2[M/x]}$$

- On type avec :

$$(\mathtt{SumG})\frac{\Gamma \vdash M : T}{\Gamma \vdash \mathtt{g} : M : T + U} \qquad (\mathtt{SumG})\frac{\Gamma \vdash M : U}{\Gamma \vdash \mathtt{d} : M : T + U}$$

$$(\mathtt{Sw})\frac{\Gamma \vdash M : T + U \qquad \Gamma, x : T \vdash N_1 : S \qquad \Gamma, x : U \vdash N_2 : S}{\Gamma \vdash \mathtt{sw}\ \mathtt{d} : M : N_1 + N_2 : S}$$

Types Sommes : Exemple

$$\begin{array}{rl} & (\lambda x.\mathtt{sw}\ x\ \triangleright y : y\ 2 + y\ 3\ 4)\ (\mathtt{g} : I) \\ \longrightarrow & \mathtt{sw}\ (\mathtt{g} : I)\ \triangleright y : y\ 2 + y\ 3\ 4 \longrightarrow I\ 2 \longrightarrow 2 \end{array}$$

On pose $T_I = (\mathbf{N} \to \mathbf{N})$ et $T_K = (\mathbf{N} \to \mathbf{N} \to \mathbf{N})$

$$(\mathtt{App})\ \frac{(\mathtt{Abs})\ \dfrac{\dfrac{\cdots}{x : (T_I + T_K) \vdash \mathtt{sw}\ x\ \triangleright y : y\ 2 + y\ 3\ 4 : \mathbf{N})}}{\vdash (\lambda x.\mathtt{sw}\ x\ \triangleright y : y\ 2 + y\ 3\ 4) : (T_I + T_K) \to \mathbf{N}} \qquad (\mathtt{SumG})\ \dfrac{\dfrac{\langle TD2 \rangle}{\vdash I : T_I}}{\vdash (\mathtt{g} : I) : T_I + T_K}}{\vdash (\lambda x.\mathtt{sw}\ x\ \triangleright y : y\ 2 + y\ 3\ 4)\ (\mathtt{g} : I) : \mathbf{N}}$$

$$(\mathtt{Sw})\ \frac{(\mathbf{Var})\ \dfrac{}{\Gamma_1 \vdash x : (T_I + T_K)} \quad (\mathbf{App})\ \dfrac{(\mathbf{Var})\ \dfrac{}{\Gamma_1 \vdash y : T_I} \qquad (\mathbf{Nat})\ \dfrac{}{\Gamma_1 \vdash 2 : \mathbf{N}}}{x : (T_I + T_K), y : T_I \vdash y\ 2 : \mathbf{N}} \quad (\mathbf{App})\ \dfrac{(\mathbf{App})\ \dfrac{(\mathbf{Var})\ \dfrac{}{\Gamma_2 \vdash y : T_K} \qquad (\mathbf{Var})\ \dfrac{}{\Gamma_2 \vdash 3 : \mathbf{N}}}{\Gamma_2 \vdash y\ 3 : \mathbf{N} \to \mathbf{N}} \quad (\mathbf{Nat})\ \dfrac{}{\Gamma_2 \vdash 4 : \mathbf{N}}}{x : (T_I + T_K), y : T_K \vdash y\ 3\ 4 : \mathbf{N}}}{x : (T_I + T_K) \vdash \mathtt{sw}\ x\ \triangleright y : y\ 2 + y\ 3\ 4 : \mathbf{N})}$$

Types Sommes et Produits : Inférence

- le système de types est toujours dirigé par la syntaxe
- génération :
  - (**SumG**) (par exemple) génère une équation $T_i = t_1 + t_2$ (avec t_1 et t_2 de nouvelles variables)
  - (**Sw**) (par exemple) force le même type dans la génération de N_1 et N_2.
- unification :
  - $T + U = S + R$ (par exemple) génère deux nouvelles équations $T = S$ et $U = R$
  - $T + U = S \times R$ (par exemple) fait échouer l'unification.

Récursion

- λ_{ST} termine $\Rightarrow$ il n'est pas Turing-complet.
- Les combinateurs de points fixe sont non-typables.
  - $Y\ F \longrightarrow^* F\ (Y\ F)$
- Passer à un langage intéressant demande d'abandonner la terminaison et d'introduire la récursion.
- Une manière de procéder est d'introduire un point fixe natif.

$$M ::= x \mid \lambda x.M \mid M\ M \mid \texttt{fix}\ M \qquad (\textbf{fix})\frac{}{\texttt{fix}\ \lambda\phi.M \longrightarrow M[(\texttt{fix}\ \lambda\phi.M)/\phi]}$$

$$(\texttt{Fix})\frac{\Gamma, \phi : T \to U \vdash M : T \to U}{\Gamma \vdash \texttt{fix}\ \lambda\phi.M : T \to U}$$

- $\texttt{fix}\ \lambda\phi.M : T \to U$, ϕ et M doivent avoir le même type.
- on peut "supposer" que c'est un type fonctionnel:
  - permet d'écrire des fonctions récursives.
  - ϕ joue le rôle de l'appel récursif.

Récursion : Exemple

$$\texttt{add} = \texttt{fix}\ (\lambda\phi.\lambda n_1\, n_2.\texttt{ifz}\ n_1\ \ n_2\ (\texttt{S}\ (\phi\ (\texttt{P}\ n_1)\ n_2))))$$

$$\begin{array}{rl}
 & \texttt{add}\ 2\ 3 \\
\longrightarrow & \lambda n_1\, n_2.\texttt{ifz}\ n_1\ \ n_2\ (\texttt{S}\ (\phi\ (\texttt{P}\ n_1)\ n_2)))[\texttt{add}/\phi] \\
= & (\lambda n_1\, n_2.\texttt{ifz}\ n_1\ \ n_2\ (\texttt{S}\ (\texttt{add}\ (\texttt{P}\ n_1)\ n_2)))\ 2\ 3 \\
\longrightarrow\longrightarrow & \texttt{ifz}\ 2\ 3\ (\texttt{S}\ (\texttt{add}\ (\texttt{P}\ 2)\ 3)) \\
\longrightarrow & \texttt{S}\ (\texttt{add}\ (\texttt{P}\ 2)\ 3) \\
\longrightarrow & \texttt{S}\ (\texttt{add}\ 1\ 3) \\
\longrightarrow & \texttt{S}\ ((\lambda n_1\, n_2.\texttt{ifz}\ n_1\ \ n_2\ (\texttt{S}\ (\texttt{add}\ (\texttt{P}\ n_1)\ n_2)))\ 1\ 3) \\
\longrightarrow\longrightarrow & \texttt{S}\ (\texttt{ifz}\ 1\ 3\ (\texttt{S}\ (\texttt{add}\ (\texttt{P}\ 1)\ 3))) \\
\longrightarrow\longrightarrow & \texttt{S}\ (\texttt{S}\ (\texttt{add}\ 0\ 3)) \\
\longrightarrow & \texttt{S}\ (\texttt{S}\ (\lambda n_1\, n_2.\texttt{ifz}\ n_1\ \ n_2\ (\texttt{S}\ (\texttt{add}\ (\texttt{P}\ n_1)\ n_2)))\ 0\ 3)) \\
\longrightarrow\longrightarrow\longrightarrow & \texttt{S}\ (\texttt{S}\ (3)) \\
\longrightarrow\longrightarrow & 5
\end{array}$$

Récursion : Exemple (II)

$$\texttt{add} = \texttt{fix}\ (\lambda\phi.\lambda n_1 n_2.\texttt{ifz}\ n_1\ n_2\ (\texttt{S}\ (\phi\ (\texttt{P}\ n_1)\ n_2)))$$

$$\dfrac{\dfrac{}{\Gamma \vdash n_1 : \mathbf{N}}(\textbf{Var}) \quad \dfrac{}{\Gamma \vdash n_2 : \mathbf{N}}(\textbf{Var}) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \phi : \mathbf{N} \to \mathbf{N} \to \mathbf{N}}(\textbf{Var}) \quad \dfrac{\dfrac{}{\Gamma \vdash n_1 : \mathbf{N}}(\textbf{Var})}{\Gamma \vdash (\texttt{P}\ n_1) : \mathbf{N}}(\texttt{Pr})}{\Gamma \vdash \phi\ (\texttt{P}\ n_1) : \mathbf{N} \to \mathbf{N}}(\textbf{App}) \quad \dfrac{}{\Gamma \vdash n_2 : \mathbf{N}}(\textbf{Var})}{\Gamma \vdash \phi\ (\texttt{P}\ n_1)\ n_2 : \mathbf{N}}(\textbf{App})}{\Gamma \vdash \texttt{S}\ (\phi\ (\texttt{P}\ n_1)\ n_2) : \mathbf{N}}(\texttt{Su})}{\phi : \mathbf{N} \to \mathbf{N} \to \mathbf{N}, n_1 : \mathbf{N}, n_2 : \mathbf{N} \vdash \texttt{ifz}\ n_1\ n_2\ (\texttt{S}\ (\phi\ (\texttt{P}\ n_1)\ n_2)) : \mathbf{N}}(\textbf{Ifz})}{\phi : \mathbf{N} \to \mathbf{N} \to \mathbf{N} \vdash \lambda n_1 n_2.\texttt{ifz}\ n_1\ n_2\ (\texttt{S}\ (\phi\ (\texttt{P}\ n_1)\ n_2)) : \mathbf{N} \to \mathbf{N} \to \mathbf{N}}(\textbf{Abs} \times 2)}{\vdash \texttt{add} : \mathbf{N} \to \mathbf{N} \to \mathbf{N}}(\texttt{Fix})$$

- On introduit des termes typables et non-terminants.
- $R = \texttt{fix}\ (\lambda\phi.\lambda x.\phi\ x)$
- $R\ M \longrightarrow (\lambda x.R\ x)\ I \longrightarrow R\ M$
-

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \phi : \alpha \to \alpha}(\textbf{Var}) \quad \dfrac{}{\Gamma \vdash x : \alpha}(\textbf{Var})}{\phi : \alpha \to \alpha, x : \alpha \vdash \phi\ x : \alpha}(\textbf{App})}{\phi : \alpha \to \alpha \vdash \lambda x.\phi\ x}(\textbf{Abs})}{\vdash R : \alpha \to \alpha}(\textbf{Fix})$$

SORBONNE UNIVERSITÉ

Récursion : Inférence

- le système de types est toujours dirigé par la syntaxe
- génération :
  - (**Fix**) génère une équation $T_i = t_1 \to t_2$ (avec t_1 et t_2 de nouvelles variables)
- unification :
  - pas d'influence.

Polymorphisme

- Polymorphisme de généricité : une même fonction/méthode est utilisable avec des types différents.
- Dans λ_{ST}, un terme typable est typable avec une infinité de type :
  - $\vdash I : \alpha \to \alpha$
  - $\vdash I : (\alpha \to \alpha) \to (\alpha \to \alpha)$
  - $\vdash I : (\alpha \to \beta \to \alpha) \to (\alpha \to \beta \to \alpha)$
  - $\vdash I : (\alpha \to (\alpha \to \alpha) \to \alpha) \to (\alpha \to (\alpha \to \alpha) \to \alpha)$
  - $\ldots$
- Dans ce processus, chaque variable de type agit comme une variable mathématique, que l'on peut remplacer par n'importe quel type.
- Cette caractéristique permet de typer $I\ I$:
  -

$$\textbf{(App)}\ \frac{\textbf{(Abs)}\ \dfrac{\textbf{(Var)}\ \dfrac{}{x : \alpha \to \alpha \vdash x : \alpha \to \alpha}}{\emptyset \vdash I : (\alpha \to \alpha) \to (\alpha \to \alpha)} \qquad \textbf{(Abs)}\ \dfrac{\textbf{(Var)}\ \dfrac{}{x : \alpha \vdash x : \alpha}}{\emptyset \vdash I : \alpha \to \alpha}}{\emptyset \vdash I\ I : \alpha \to \alpha}$$

  - Dans la branche de gauche, I est typée avec le type $(\alpha \to \alpha) \to (\alpha \to \alpha)$, et dans la branche de droite avec $\alpha \to \alpha$.

SORBONNE UNIVERSITÉ

Polymorphisme (II)

- Ce processus ne s'applique pas aux variables liées.
- $\delta = \lambda x.x\ x$ n'est pas typable :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : t_1 \vdash x : t_3 \to T_2}(\textbf{Var}) \quad \dfrac{}{x : t_1 \vdash x : t_3}(\textbf{Var})}{x : T_1 \vdash x\ x : T_2}(\textbf{App})}{\emptyset \vdash \lambda x.x\ x : t_1 \to t_2}(\textbf{Abs})$$

- on tombe sur $t_3 \to t_2 = t_3$ qui n'est pas unifiable.
- la règle (**App**) force le contexte Γ (donc le type de x) à être le même des deux côtés de l'application.
- pourtant $\delta\ I$ ne semble pas plus problématique que $I\ I$.
- On peut étendre le système de types pour typer ces termes faisant apparaître de la généricité.
  - le même code qui va substituer les deux occurences de x est utilisé "de deux manières différentes".

Polymorphisme (III)

- On ajoute la quantification universelle aux types:

$$T ::= \alpha \mid T \to T \mid \forall \alpha . T$$

- on ajoute des règles pour ce constructeur de types :

$$(\textbf{Gen}) \frac{\Gamma \vdash M : T}{\Gamma \vdash M : \forall \alpha . T} \qquad (\textbf{Inst}) \frac{\Gamma \vdash M : \forall \alpha . T}{\Gamma \vdash M : T[U/\alpha]}$$

- (**Gen**) permet de généraliser les variables de types d'un type donné à un terme.
- (**Inst**) permet d'instantier la variable liée d'un type universel par n'importe quel type.
- c'est *Système F*.

Polymorphisme (IV)

- δ est typable :

$$\textbf{(Abs)}\ \frac{\textbf{(App)}\ \dfrac{\textbf{(Inst)}\ \dfrac{\textbf{(Var)}\ \dfrac{}{x : \forall\alpha.\alpha \to \alpha \vdash x : (\forall\alpha.\alpha \to \alpha)}}{x : \forall\alpha.\alpha \to \alpha \vdash x : (\forall\alpha.\alpha \to \alpha) \to (\forall\alpha.\alpha \to \alpha)} \qquad \textbf{(Var)}\ \dfrac{}{x : \forall\alpha.\alpha \to \alpha \vdash x : \forall\alpha.\alpha \to \alpha}}{x : \forall\alpha.\alpha \to \alpha \vdash x\ x : (\forall\alpha.\alpha \to \alpha)}}{\emptyset \vdash \delta : (\forall\alpha.\alpha \to \alpha) \to (\forall\alpha.\alpha \to \alpha)}$$

- ici on utilise la règle (**Inst**) pour remplacer chaque α par le type $\forall\alpha.\alpha \to \alpha$ lui-même.
- Ω n'est pas typable.
  - le type de δ ne peut pas être généralisé de manière intéressante (il n'a pas de variable libre)
  - on ne peut donc pas typer δ avec deux types différents dans $\delta\ \delta$.

Polymorphisme (V)

- Théorème : En Système F, M est typable si et seulement si M est fortement normalisant.
- Conséquence :

Polymorphisme (V)

- Théorème : En Système F, M est typable si et seulement si M est fortement normalisant.
- Conséquence : F n'est pas inférable (l'inférence n'est pas décidable)
  - elle n'est plus dirigée par la syntaxe, (on peut utiliser (**Abs**) ou (**Inst**))
  - quand on utilise (**Inst**), on a plusieurs possibilités pour U.
- pour un langages de programmation, on veut du polymorphisme inférable.

Let-polymorphisme

- les langages fonctionnels utilisent un polymorphisme plus restreint
- ils disposent d'une construction `let`
  - $\texttt{let}\ x = N\ \texttt{in}\ M$ a la même sémantique que $(\lambda x.M)\ N$
  - c'est le seul endroit où l'on peut créer du polymorphisme

$$(\texttt{Let})\frac{\Gamma \vdash N : T \qquad \Gamma, x : \mathbf{Gen}(T) \vdash M : U}{\Gamma \vdash \texttt{let}\ x = N\ \texttt{in}\ M : U}$$

avec $\mathbf{Gen}(T)$ l'opération syntaxique qui transforme T en $\forall\alpha_1\forall\alpha_2 \ldots \forall\alpha_n.T$ si $\alpha_1, \ldots, \alpha_n$ sont les variables de types libres de T.

  - l'inférence devient décidable.
- l'algorithme de *Hindley-Milner* (variante d'unification) est utilisé pour l'inférence de types des langages *à la ML*.

Exemple de let-typage

$$\textbf{(App)}\ \frac{\textbf{(Inst)}\ \dfrac{\textbf{(Var)}\ \dfrac{}{i : \forall\alpha.\alpha \to \alpha \vdash i : \forall\alpha.\alpha \to \alpha}}{i : \forall\alpha.\alpha \to \alpha \vdash i : (\mathbf{N} \to \mathbf{N}) \to (\mathbf{N} \to \mathbf{N})} \qquad \textbf{(Inst)}\ \dfrac{\textbf{(Var)}\ \dfrac{}{i : \forall\alpha.\alpha \to \alpha \vdash i : \forall\alpha.\alpha \to \alpha}}{i : \forall\alpha.\alpha \to \alpha \vdash i : \mathbf{N} \to \mathbf{N}}}{i : \forall\alpha.\alpha \to \alpha \vdash i\ i : \mathbf{N} \to \mathbf{N}}$$

$$\textbf{(Let)}\ \frac{\textbf{(Abs)}\ \dfrac{\textbf{(Var)}\ \dfrac{}{x : \alpha \vdash x : \alpha}}{\vdash \lambda x.x : \alpha \to \alpha} \qquad \textbf{(App)}\ \dfrac{\dfrac{\ldots}{i : \forall\alpha.\alpha \to \alpha \vdash i\ i : \mathbf{N} \to \mathbf{N}} \qquad \textbf{(Nat)}\ \dfrac{}{i : \forall\alpha.\alpha \to \alpha \vdash 2 : \mathbf{N}}}{i : \forall\alpha.\alpha \to \alpha \vdash i\ i\ 2}}{\vdash \texttt{let}\ i = \lambda x.x\ \texttt{in}\ i\ i\ 2 : \mathbf{N}}$$

- **Gen**$(\alpha \to \alpha)$ c'est $\forall\alpha.\alpha \to \alpha$
- à gauche, **(Inst)** avec $[(\mathbf{N} \to \mathbf{N})/\alpha]$
- à droite, **(Inst)** avec $[\mathbf{N}/\alpha]$

Let-Polymorphisme : Inférence

- le système de type n'est plus dirigé par la syntaxe :
  - on peut utiliser (**Inst**) à tout moment.
- méthode possible :
  - pendant la génération,
  - on rencontre `let` $x = N$ `in` M,
  - fait l'inférence (génération + unification) de N.
  - on généralise le type obtenu T.
  - on continue la génération dans M.
- unification :
  - on traite $\forall\alpha.T_1 = T_2$
  - on prend une nouvelle variable de type β
  - on remplace l'équation par $T_1[\beta/\alpha] = T_2$
  - ainsi plusieurs apparitions du même type polymorphe dans les équations sont indépendantes
  - l'instantiation est faite pendant l'unification
  - attention à l'ordre ($\forall$ devrait être "prioritaire")

$$
\begin{array}{rl}
 & \{(\forall\alpha.\alpha \to \alpha, t_1), (t_1, \mathbf{N} \to \mathbf{N}), (\beta_1 \to \beta_2 \to \beta_1, \gamma \to t_1)\} \\
\longrightarrow & \{(\forall\alpha.\alpha \to \alpha, \mathbf{N} \to \mathbf{N}), (\beta_1 \to \beta_2 \to \beta_1, \gamma \to (\forall\alpha.\alpha \to \alpha))\} \\
\longrightarrow & \{(\alpha_1 \to \alpha_1, \mathbf{N} \to \mathbf{N}), (\beta_1 \to \beta_2 \to \beta_1, \gamma \to (\forall\alpha.\alpha \to \alpha))\} \\
\longrightarrow & \{(\alpha_1 \to \alpha_1, \mathbf{N} \to \mathbf{N}), (\beta_1, \gamma), (\beta_2 \to \beta_1, \forall\alpha.\alpha \to \alpha)\} \\
\longrightarrow & \{(\alpha_1 \to \alpha_1, \mathbf{N} \to \mathbf{N}), (\beta_1, \gamma), (\beta_2 \to \beta_1, \alpha_2 \to \alpha_2)\} \\
\longrightarrow^* & \{\ldots, (\alpha_1, \mathbf{N}), \ldots (\alpha_2, \gamma), \ldots\}
\end{array}
$$

SORBONNE UNIVERSITÉ

Traits Impératifs

- les langages ML contiennent des traits impératifs.
- On peut ajouter à λ :
  - allocation $\texttt{ref}\ M$
  - déréférencement $!M$
  - réaffectation $M{:=}N$
  - une unité $\circ$
- Une mémoire σ associe des régions (adresses) à des termes.
- On réduit des couples (terme;mémoire)
- On ajoute les règles :

$$(\texttt{ref1})\frac{\rho \notin M \qquad \rho \notin \sigma}{\texttt{ref}\ M;\sigma \longrightarrow \rho;\sigma[\rho \mapsto M]} \qquad (\texttt{ref2})\frac{M \longrightarrow M'}{\texttt{ref}\ M;\sigma \longrightarrow \texttt{ref}\ M';\sigma}$$

$$(\texttt{deref1})\frac{\sigma(\rho) = M}{!\rho;\sigma \longrightarrow M;\sigma} \qquad (\texttt{deref2})\frac{M \longrightarrow M'}{!M;\sigma \longrightarrow !M';\sigma}$$

$$(\texttt{as3})\frac{N \longrightarrow N'}{M := N;\sigma \longrightarrow M := N';\sigma} \qquad (\texttt{as1})\frac{\rho \in \sigma}{\rho := N;\sigma \longrightarrow \circ;\sigma[\rho \mapsto N]}$$

$$(\texttt{as2})\frac{M \longrightarrow M'}{M := N;\sigma \longrightarrow M' := N';\sigma}$$

- on met à jour les autres règles
  - (elles n'accèdent pas et ne modifient pas la mémoire)

Traits Impératifs (II)

- On ajoute un type de base $\star$ pour les commandes.
- On ajoute un constructeur de type spécifique **Ref**.
- On ajoute les règles :

$$(\texttt{Unit})\frac{}{\Gamma \vdash \circ : \star} \qquad (\texttt{Ref})\frac{\Gamma \vdash M : T}{\Gamma \vdash \texttt{ref}\ T : \textbf{Ref}\ T}$$

$$(\texttt{Deref})\frac{\Gamma \vdash M : \textbf{Ref}\ T}{\Gamma \vdash !M : T} \qquad (\texttt{As})\frac{\Gamma \vdash M : \textbf{Ref}\ T \qquad \Gamma \vdash N : T}{\Gamma \vdash M{:=}N : \star}$$

Traits Impératifs (III)

$$\textbf{(Let)}\ \frac{\textbf{(Ref)}\ \frac{\textbf{(Abs)}\ \frac{\textbf{(Var)}\ \frac{}{x : \alpha \vdash x : \alpha}}{\lambda x.x : \alpha \to \alpha}}{\vdash \texttt{ref}\ \lambda x.x : \textbf{Ref}\ (\alpha \to \alpha)} \quad \textbf{(App)}\ \frac{\textbf{(App)}\ \frac{\dots}{\Gamma, y : \bigstar \vdash !f\ \circ : \bigstar}}{\Gamma \vdash (\lambda y.!f\ \circ) : \bigstar \to \bigstar} \quad \textbf{(As)}\ \frac{\dots \qquad \textbf{(Abs)}\ \frac{\textbf{(Succ)}\ \frac{\textbf{(Var)}\ \frac{}{\Gamma, z : \mathbf{N} \vdash z : \mathbf{N}}}{\Gamma, z : \mathbf{N} \vdash \mathtt{S}\ z : \mathbf{N}}}{\Gamma \vdash \lambda z.\mathtt{S}\ z : \mathbf{N} \to \mathbf{N}}}{\Gamma \vdash f : \textbf{Ref}(\mathbf{N} \to \mathbf{N}) \qquad \Gamma \vdash (f := \lambda z.\mathtt{S}\ z) : \bigstar}}{f : \forall\alpha.\textbf{Ref}\ (\alpha \to \alpha) \vdash (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z)}}{\vdash \texttt{let}\ f = \texttt{ref}\ \lambda x.x\ \texttt{in}\ (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z)}$$

$$\textbf{(App)}\ \frac{\textbf{(Deref)}\ \frac{\textbf{(Inst)}\ \frac{\textbf{(Var)}\ \frac{}{\Gamma, y : \bigstar \vdash f : \forall\alpha.\textbf{Ref}\ (\alpha \to \alpha)}}{\Gamma, y : \bigstar \vdash f : \textbf{Ref}\ (\bigstar \to \bigstar)}}{\Gamma, y : \bigstar \vdash !f : \bigstar \to \bigstar} \quad \textbf{(Unit)}\ \frac{}{\Gamma, y : \bigstar \vdash \circ : \bigstar}}{\Gamma, y : \bigstar \vdash !f\ \circ : \bigstar}$$

$$\textbf{(Inst)}\ \frac{\textbf{(Var)}\ \frac{}{\Gamma \vdash f : \forall\alpha.\textbf{Ref}(\alpha \to \alpha)}}{\Gamma \vdash f : \textbf{Ref}(\mathbf{N} \to \mathbf{N})}$$

- **Gen**(**Ref**$(\alpha \to \alpha)$) c'est $\forall\alpha.$**Ref**$(\alpha \to \alpha)$
- à gauche, (**Inst**) avec $[\bigstar/\alpha]$
- à droite, (**Inst**) avec $[\mathbf{N}/\alpha]$

SORBONNE UNIVERSITÉ

Traits Impératifs (IV)

$$
\begin{array}{lll}
 & \texttt{let } f = \texttt{ref } \lambda x.x \texttt{ in } (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z) & ;\quad \emptyset \\
\longrightarrow & \texttt{let } f = \rho \texttt{ in } (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z) & ;\quad \rho \mapsto \lambda x.x \\
\longrightarrow & (\lambda y.!\rho\ \circ)\ (\rho := \lambda z.\mathtt{S}\ z) & ;\quad \rho \mapsto \lambda x.x \\
\longrightarrow & (\lambda y.!\rho\ \circ)\ \circ & ;\quad \rho \mapsto \lambda z.\mathtt{S}\ z \\
\longrightarrow & !\rho\ \circ & ;\quad \rho \mapsto \lambda z.\mathtt{S}\ z \\
\longrightarrow & (\lambda z.\mathtt{S}\ z)\ \circ & ;\quad \rho \mapsto \lambda z.\mathtt{S}\ z \\
\longrightarrow & \mathtt{S}\ \circ & ;\quad \rho \mapsto \lambda z.\mathtt{S}\ z \\
\not\longrightarrow & &
\end{array}
$$

- le typage échoue dans son rôle.

Traits Impératifs (V)

- l'approche naïve ne fonctionne pas.
- il ne faut pas toujours généraliser les types impératifs lors d'un `let`
- la solution des langages ML est :
  - distinguer les termes expansifs et non-expansifs
  - les termes non-expansifs sont généralisés,
  - les termes expansifs recoivent un polymorphisme faible
    - par exemple, **Ref** $\_\alpha \longrightarrow \_\alpha$
    - pendant l'inférence, la première fois qu'ils sont instantiés ils se transforment en leur instantiation : $\_\alpha \longrightarrow \_\alpha$ devient **Nat** $\longrightarrow$ **Nat** et ne peut plus être modifié.
    - en OCaml c'est `'a` vs. `'_a`

Expansivité

- Les termes suivants sont non-expansifs :
  - les variables et les termes de base (entiers, $\circ$),
  - les abstractions,
  - les applications et les let quand les sous-expressions sont non-expansives.
- **GenF(T)** généralise faiblement le type T : $\overline{\forall}\alpha_1 \ldots \overline{\forall}\alpha_n.T$
- Deux règles de typage pour let et une instantiation faible:

$$(\textbf{LetE})\ \frac{\Gamma \vdash N : T \qquad \Gamma, x : \textbf{GenF}(T) \vdash M : U \qquad N \text{ expansif}}{\Gamma \vdash \texttt{let}\ x = N\ \texttt{in}\ M : U}$$

$$(\textbf{Let})\ \frac{\Gamma \vdash N : T \qquad \Gamma, x : \textbf{Gen}(T) \vdash M : U \qquad N \text{ non-expansif}}{\Gamma \vdash \texttt{let}\ x = N\ \texttt{in}\ M : U} \qquad (\textbf{InstF})\ \frac{\Gamma, x : T[U/\alpha] \vdash M : S}{\Gamma, x : \overline{\forall}\alpha.T \vdash M : S}$$

- Il faut empêcher un contexte contenant des types faiblement polymorphe d'être copié :

$$(\textbf{App})\ \frac{\Gamma \vdash M : U \to T \qquad \Gamma \vdash N : U \qquad \overline{\forall} \notin \Gamma}{\Gamma \vdash M\ N : T}$$

$$(\textbf{LetE})\ \frac{\Gamma \vdash N : T \qquad \Gamma, x : \textbf{GenF}(T) \vdash M : U \qquad N \text{ expansif} \qquad \overline{\forall} \notin \Gamma}{\Gamma \vdash \texttt{let}\ x = N\ \texttt{in}\ M : U} \qquad (\textbf{LetNE}) \qquad (\textbf{As}) \qquad \ldots$$

SORBONNE UNIVERSITÉ

Expansivité : Exemple

$$\dfrac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}\,(\textbf{Var})}{\lambda x.x : \alpha \to \alpha}\,(\textbf{Abs})}{\vdash \texttt{ref}\ \lambda x.x : \textbf{Ref}\ (\alpha \to \alpha)}\,(\textbf{Ref}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, z : \mathbf{N} \vdash z : \mathbf{N}}\,(\textbf{Var})}{\Gamma, z : \mathbf{N} \vdash \mathtt{S}\ z : \mathbf{N}}\,(\textbf{Succ})}{\Gamma \vdash \lambda z.\mathtt{S}\ z : \mathbf{N} \to \mathbf{N}}\,(\mathtt{Abs}) \quad \dfrac{}{\Gamma \vdash f : \mathbf{Ref}(\mathbf{N} \to \mathbf{N})}\,(\textbf{Var})}{\Gamma \vdash (f := \lambda z.\mathtt{S}\ z) : \star}\,(\mathtt{As}) \quad \dfrac{\dfrac{\dfrac{}{\Gamma, y : \star \vdash !f\ \circ : \star}\,\ldots}{\Gamma \vdash (\lambda y.!f\ \circ) : \star \to \star}\,(\textbf{App})}{f : \mathbf{Ref}\ (\mathbf{N} \to \mathbf{N}) \vdash (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z)}\,(\textbf{App})}{f : \overline{\forall}\alpha.\mathbf{Ref}\ (\alpha \to \alpha) \vdash (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z)}\,(\textbf{InstF})}{\vdash \texttt{let}\ f = \texttt{ref}\ \lambda x.x\ \texttt{in}\ (\lambda y.!f\ \circ)\ (f := \lambda z.\mathtt{S}\ z)}\,(\textbf{etE})$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma, y : \star \vdash f : \mathbf{Ref}\ (\star \to \star)}\,\text{ECHEC}}{\Gamma, y : \star \vdash !f : \star \to \star}\,(\textbf{Deref}) \quad \dfrac{}{\Gamma, y : \star \vdash \circ : \star}\,(\textbf{Unit})}{\Gamma, y : \star \vdash !f\ \circ : \star}\,(\textbf{App})$$

- le terme `ref` $\lambda x.x$ est expansif.
- on généralise faiblement son type en $\overline{\forall}\alpha.\mathbf{Ref}\ (\alpha \to \alpha)$.
  - on ne peut plus utiliser (**Inst**) comme avant.
- dans le (**App**), on ne peut pas garder ce type dans le contexte.
  - on doit faire un (**InstF**) en dessous.
  - on doit choisir une même instantiation pour les deux branches.

SORBONNE UNIVERSITÉ

Traits Impératifs : Inférence

- les nouvelles constructions correspondent aux nouvelles règles.
- génération:
  - même principe, mais on doit décider de l'expansivité
  - on distingue $\forall$ et $\overline{\forall}$.
- unification:
  - quand on instantie un $\overline{\forall\alpha}$ (par renommage de α), on doit le rendre indisponible à une prochaine instantiation dans le reste du processus d'inférence.
  - on peut utiliser des références.
  - on peut sauter l'étape de renommage.

λ avec passage de messages

- On peut définir un λ-calcul qui communique par passage de message. $a, b, \ldots$ sont des canaux

$$M ::= x \mid \lambda x.M \mid M\ M \mid a(x).M \mid \overline{a}\langle M\rangle.M \qquad S ::= S \parallel S \mid [M]$$

$$(\textbf{Ter})\ \frac{M \longrightarrow M'}{[M] \longrightarrow [M']} \qquad (\textbf{Sys1})\ \frac{S_1 \longrightarrow S_1'}{S_1 \parallel S_2 \longrightarrow S_1' \parallel S_2}$$

$$(\textbf{Sys2})\ \frac{S_2 \longrightarrow S_2'}{S_1 \parallel S_2 \longrightarrow S_1 \parallel S_2'}$$

$$(\textbf{Comm})\ \frac{}{[a(x).M_1] \parallel [\overline{a}\langle N\rangle.M_2] \longrightarrow M_1[N/x] \parallel [M_2]}$$

- à rapprocher des algèbres de processus (PPC) et de `Go` (PC3R).
- on veut rejeter $[\overline{a}\langle 3\rangle.\circ] \parallel [a(x).(x\ 0)]$

λ avec passage de messages : Typage

- types pour les canaux $\sharp\ T$ (transporte du T),
- contextes Ξ pour les canaux,
- jugements pour les termes $\Gamma \mid \Xi \vdash M : T$
- jugements pour les systèmes $\Xi \vdash S$

$$(\textbf{In})\frac{\Gamma, x : U \mid \Xi, a : \sharp\ U \vdash M : T}{\Gamma \mid \Xi, a : \sharp\ U \vdash a(x).M : T}$$

$$(\textbf{Out})\frac{\Gamma \mid \Xi, a : \sharp\ U \vdash N : U \qquad \Gamma \mid \Xi, a : \sharp\ U \vdash M : T}{\Gamma \mid \Xi, a : \sharp\ U \vdash \overline{a}\langle N\rangle.M : T} \qquad (\textbf{Ter})\frac{\emptyset \mid \Xi \vdash M}{\Xi \vdash [M]}$$

$$(\textbf{Sys})\frac{\Xi \vdash S_1 \qquad \Xi \vdash S_2}{\Xi \vdash S_1 \parallel S_2}$$

SORBONNE UNIVERSITÉ

Exceptions

- On peut ajouter des mécanismes exceptionnels :

$$M ::= x \mid \lambda x.M \mid M\ M \mid \mathcal{C}\ M \mid \mathcal{E}\ M$$

- Sémantique :

$$(\mathbf{C1})\frac{M \longrightarrow M'}{\mathcal{C}\ M \longrightarrow \mathcal{C}\ M'} \qquad (\mathbf{C2})\frac{M \text{ contient } \mathcal{E}\ N}{\mathcal{C}\ M \longrightarrow N}$$

- On peut forcer que $\mathcal{E}\ N$ soit en position évaluable (si on suit une stratégie) dans (**C2**)
- Typage :

$$(\mathbf{E})\frac{\Gamma \vdash_U M : U}{\Gamma \vdash_U \mathcal{E}\ M} \qquad (\mathbf{C})\frac{\Gamma \vdash_U M : U}{\Gamma \vdash \mathcal{E}\ M : U}$$

  - jugement paramétré par le type de l'exception.
  - la valeur exceptionnelle doit être du même type que le terme dans laquelle elle se trouve.

- à raffiner ...

Curry-Howard : Logique Intuitionniste

- Formules logiques avec l'implication.

$$\phi ::= A \mid \phi \Rightarrow \phi$$

- Γ : ensemble d'hypothèses (de formules),
- Jugements $\Gamma \vdash \phi$: "ϕ est prouvable avec les hypothèses Γ"
- Séquents Intuitionnistes :

$$(\mathbf{Ax})\frac{}{\Gamma, A \vdash A} \qquad (\mathbf{MP})\frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad (\mathbf{I})\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

- Formules prouvables :

$$(\mathbf{I})\frac{(\mathbf{I})\frac{(\mathbf{I})\frac{(\mathbf{MP})\frac{(\mathbf{MP})\frac{(\mathbf{Ax})\frac{}{\Gamma \vdash A \Rightarrow (B \Rightarrow C)} \quad (\mathbf{Ax})\frac{}{\Gamma \vdash A}}{\Gamma \vdash B \Rightarrow C} \quad (\mathbf{MP})\frac{(\mathbf{Ax})\frac{}{\Gamma \vdash A \Rightarrow B} \quad (\mathbf{Ax})\frac{}{\Gamma \vdash A}}{\Gamma \vdash B}}{A \Rightarrow B \Rightarrow C, A \Rightarrow B, A \vdash C}}{A \Rightarrow B \Rightarrow C, A \Rightarrow B \vdash A \Rightarrow C}}{A \Rightarrow B \Rightarrow C \vdash (A \Rightarrow B) \Rightarrow A \Rightarrow C}}{\vdash (A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C}$$

SORBONNE UNIVERSITÉ

Curry-Howard : Correspondance

- Formules de SI $\leftrightarrow$ Types de λ_{ST}
- Preuves de SI $\leftrightarrow$ Dérivation de typage de λ_{ST}
  - $=$ Termes de λ_{ST}
  - car le système de types est dirigé par la syntaxe

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : A \to (B \to C)}\,(\textbf{Var}) \quad \dfrac{}{\Gamma \vdash z : A}\,(\textbf{Var})}{\Gamma \vdash x\ z : B \to C}\,(\textbf{App}) \quad \dfrac{\dfrac{}{\Gamma \vdash y : A \to B}\,(\textbf{Var}) \quad \dfrac{}{\Gamma \vdash z : A}\,(\textbf{Var})}{\Gamma \vdash y\ z : B}\,(\textbf{App})}{x : A \to B \to C, y : A \to B, z : A \vdash \lambda(x\ z)\ (y\ z) : C}\,(\textbf{App})}{x : A \to B \to C, y : A \to B \vdash \lambda z.(x\ z)\ (y\ z) : A \to C}\,(\textbf{Abs})}{x : A \to B \to C \vdash \lambda yz.(x\ z)\ (y\ z) : (A \to B) \to A \to C}\,(\textbf{Abs})}{\vdash S : (A \to B \to C) \to (A \to B) \to A \Rightarrow C}\,(\textbf{Abs})$$

- ??? de SI $\leftrightarrow$ Réduction de λ_{ST}

SORBONNE UNIVERSITÉ

Curry-Howard : Elimination des coupures

- Opération de transformation des preuves.
- Utilisation d'un lemme :

$$(\mathbf{App})\dfrac{(\mathbf{I})\dfrac{\dfrac{\mathcal{P}_1}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{\mathcal{P}_2}{\Gamma \vdash A}}{\Gamma \vdash B} \quad \longrightarrow \quad \dfrac{\mathcal{P}_1[\mathcal{P}_2]}{\Gamma \vdash B}$$

avec $\mathcal{P}_1[\mathcal{P}_2]$ la preuve obtenue en prenant $\mathcal{P}_1$ et en remplaçant tous les $(\mathbf{Ax})\dfrac{}{\Gamma', A \vdash A}$ par $\dfrac{\mathcal{P}_2}{\Gamma' \vdash A}$

- on simplifie une preuve en *inlinant* un lemme à tous les endroits où on en avait besoin.
- l'enchainement de (**App**) avec (**I**) à gauche s'appelle une coupure, le processus correspondant à la réduction est l'élimination des coupures.
- éliminer une coupure peut faire grossir la preuve et ajouter des coupures.
  - en copiant plusieurs fois les coupures dans $\mathcal{P}_\in$
- l'élimination des coupures termine.

Curry-Howard : Logique Classique

- Calcul des Séquents classique :

$$(\mathbf{TE})\frac{}{\vdash A \vee A} \qquad \textbf{ou} \qquad (\mathbf{Abs})\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A} \qquad \textbf{ou}$$

$$(\mathbf{PL})\frac{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}{\Gamma \vdash A}$$

- comment introduire $A \vee B$ dans λ_{ST} ?

Curry-Howard : Logique Classique

- Calcul des Séquents classique :

$$(\mathbf{TE})\frac{}{\vdash A \vee A} \qquad \textbf{ou} \qquad (\mathbf{Abs})\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A} \qquad \textbf{ou}$$

$$(\mathbf{PL})\frac{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}{\Gamma \vdash A}$$

- comment introduire $A \vee B$ dans λ_{ST} ?

$$(\mathtt{SumG})\frac{\Gamma \vdash M : T}{\Gamma \vdash \mathtt{g} : M : T \vee U} \qquad (\mathtt{SumG})\frac{\Gamma \vdash M : U}{\Gamma \vdash \mathtt{d} : M : T \vee U}$$

$$(\mathtt{Sw})\frac{\Gamma \vdash M : T \vee U \qquad \Gamma, x : T \vdash N_1 : S \qquad \Gamma, x : U \vdash N_2 : S}{\Gamma \vdash \mathtt{sw\ d} : M : N_1 + N_2 : S}$$

soit

$$(\mathbf{LI})\frac{\Gamma \vdash T}{\Gamma \vdash T \vee U} \qquad (\mathbf{RI})\frac{\Gamma \vdash U}{\Gamma \vdash T \vee U} \qquad (\mathbf{E})\frac{\Gamma \vdash T \vee U \qquad \Gamma, T \vdash S \qquad \Gamma, U \vdash S}{\Gamma \vdash S}$$

Curry-Howard : Logique Classique (II)

- il faut complexifier λ pour obtenir des calculs en relation avec SK
- le $\lambda\mu$-calcul permet de définir des termes nommés:

$$M ::= x \mid \lambda x.M \mid M\ M \mid \mu\alpha.E \qquad\qquad E ::= [\alpha]\ M$$

- la réduction structurelle permet, dans un terme liant le nom α, de distribuer un argument N à tous les sous-termes nommés par α.

$$\overline{(\mu\alpha.M)\ N \longrightarrow \mu\alpha.M[[\alpha](N\ M')/[\alpha]M']}$$

- le système de types standard de $\lambda\mu$ correspond à la déduction naturelle classique.
- $\overline{\lambda}\mu\tilde{\mu}$ correspond aux séquents classiques

$$C ::= [V|E] \qquad V ::= x \mid \lambda x.V \mid e\ v \mid \mu\alpha.c \qquad E ::= \alpha \mid \alpha\lambda.e \mid v\ e \mid \overline{\mu}x.c$$

- avec la réduction

$$\overline{[\lambda x.V|V'\ E] \longrightarrow [V'|\overline{\mu}x.[V|E]]} \qquad \overline{[E'\ V|\alpha\lambda.E] \longrightarrow [\mu\alpha.[V|E]|E']}$$

$$\overline{\mu\alpha.[V|\alpha]} \longrightarrow V \qquad \overline{[\mu\alpha.C|E] \longrightarrow C[E/\alpha]} \qquad \overline{[V|\overline{\mu}.C] \longrightarrow C[V/x]}$$

$$\overline{\overline{\mu}x.[x|E]} \longrightarrow E$$

- termes, contextes, et commandes manipulent le flot de contrôle.

SORBONNE UNIVERSITÉ

Cube de Barendregt

- trois directions pour enrichir λ_{ST}
- termes dépendants de types : Polymorphisme
  - Système F présenté avec instantiation explicite et réduction typée

$$M ::= x \mid \lambda x.M \mid M\ M \mid \Lambda T.M \mid T \qquad T ::= \alpha \mid T \to T \mid \forall \alpha.T$$

$$\frac{\Gamma \vdash \Lambda X.M : \forall \alpha.T}{\Gamma \vdash \Lambda X.M\ U \longrightarrow \Gamma \vdash M : T[U/X]}$$

  - on peut définir δ comme $\Lambda Y.\lambda x.(x\ Y \to Y)\ (x\ Y)$
  - et I comme $\Lambda X.\lambda x.x$
  - et typer $\delta\ (\forall \alpha.(\alpha \to \alpha))\ I$.
- types dépendant de termes (appelés "types dépendants")
  - formellement $T ::= \alpha \mid T \to T \mid \pi x.T$
  - permet de définir, par exemple, 4 `vect`, les vecteurs de taille inférieure à 4.
- types dépendant de types (constructeurs de types)
  - formellement $T ::= \alpha \mid T \to T \mid \Pi X.T$
  - permet de définir `Liste A`, les listes d'éléments de types A,
  - hierarchie de sortes (comme en PAF)

Curry-Howard : Correspondances

- Le Calcul des Constructions *ferme* le cube de Barendregt
- Quelques correspondances de Curry-Howard connues :
  - SI $\leftrightarrow \lambda_{ST}$
  - SK $\leftrightarrow \overline{\lambda}\mu\tilde{\mu}$
  - Arithmétique de Peano $\leftrightarrow$ Système F
  - Logique de Hilbert $\leftrightarrow$ Logique Combinatoire
  - LI d'ordre supérieur $\leftrightarrow$ Calcul des Constructions.
  - λ linéaires $\leftrightarrow$ Logiques Linéaires.

Conclusion

- TDs 03-06 - Travail en autonomie
  - réalisation d'un évaluateur-typeur,
  - synthèse d'article.
- Cours 5 : Typage en Objet.