

Le λ -calcul polymorphe.

On étend la grammaire des types :

$$A, B ::= X \mid A \rightarrow B \mid \forall X A.$$

Ici, les X ne sont plus les types de base (noté \mathbf{X} précédemment), mais ce sont des *variables de types*.

Pour mieux refléter les notations de la littérature, on aura plutôt :

$$A, B ::= \alpha \mid A \rightarrow B \mid \forall \alpha A.$$

Le « $\forall \alpha A$ » est une structure qui lie, \forall est un lieur. Il y a donc une notion de variables libres d'un type $\mathcal{V}\ell(A)$, d' α -conversion, et de substitution. Par exemple,

$$\mathcal{V}\ell(\forall \alpha \forall \beta (\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \alpha)) = \{\gamma\}$$

et

$$\forall \alpha \alpha \rightarrow \alpha =_{\alpha} \forall \beta \beta \rightarrow \beta.$$

1 Typage, version implicite.

Aux trois règles des types simples, on ajoute les deux règles ci-dessous :

$$\alpha \notin \mathcal{V}\ell(\Gamma) \quad \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall \alpha A} \mathcal{T}_g \quad \frac{\Gamma \vdash M : \forall \alpha A}{\Gamma \vdash M : A[B/\alpha]} \mathcal{T}_i.$$

Les règles correspondent à la *généralisation* et à l'*instantiation*.

Et, *on ne change pas les λ -termes*.

2 Typage, version explicite.

On change les λ -termes :

$$M, N ::= x \mid M N \mid \lambda x : A. M \mid \Lambda \alpha. M \mid M A.$$

La construction $\Lambda \alpha. M$ est une *abstraction de types*, et la construction $M A$ est l'*application d'un terme à un type*. On note **explicitement** le type « d'entrée » de l'abstraction.

On change les règles de β -réduction :

$$\frac{}{(\Lambda \alpha. M) A \rightarrow_{\beta} M[A/\alpha]} \quad \frac{M \rightarrow_{\beta} M'}{\Lambda \alpha. M \rightarrow_{\beta} \Lambda \alpha. M'}$$

$$\frac{M \rightarrow_{\beta} M'}{M A \rightarrow_{\beta} M' A}.$$

On change également les règles de typage :

$$\begin{array}{c} \Gamma(x) = A \quad \frac{}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A. M : A \rightarrow B} \quad x \notin \text{dom}(\Gamma) \\ \frac{\Gamma \vdash M : A \quad \Gamma \vdash M : \forall \alpha A}{\Gamma \vdash \Lambda \alpha. M : \forall \alpha A} \quad \alpha \notin \mathcal{V}(\Gamma) \quad \Gamma \vdash M B : A[B/\alpha] \end{array}$$

Lemme 1. On a $\Gamma \vdash M : A$ dans le système explicite ssi il existe M' dans le système explicite avec $\Gamma \vdash M' : A$ (explicite) et M est « l'effacement » de M' .

- ▷ La représentation implicite est plus utilisée dans les langages comme OCaml, où l'on doit inférer les types.
- ▷ La représentation explicite correspond plus aux langages comme Rocq, avec un lien plus naturel avec la logique.

Exemple 1. Soit le λ -terme non typé $\underline{2} := \lambda f. \lambda z. f (f z)$. Comment le typer en version explicite ?

1.

$$\frac{\frac{\frac{\vdots}{f : \alpha \rightarrow \alpha, z : \alpha \vdash f (f z) : \alpha}}{f : \alpha \rightarrow \alpha \vdash \lambda z : \alpha. f (f z) : \alpha \rightarrow \alpha}}{\emptyset \vdash \lambda f : \alpha \rightarrow \alpha. \lambda z : \alpha. f (f z) : (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}}{\emptyset \vdash \Lambda \alpha. \lambda f : \alpha \rightarrow \alpha. \lambda z : \alpha. f (f z) : \forall \alpha (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha.}$$

2.

$$\frac{\frac{\frac{\vdots}{f : \forall \alpha \alpha \rightarrow \alpha, z : \beta \vdash f \beta (f \beta z) : \beta}}{f : \forall \alpha \alpha \rightarrow \alpha \vdash \lambda z : \beta. f \beta (f \beta z) : \beta \rightarrow \beta}}{\emptyset \vdash \lambda f : \forall \alpha \alpha \rightarrow \alpha. \lambda z : \beta. f \beta (f \beta z) : (\forall \alpha \alpha \rightarrow \alpha) \rightarrow \beta \rightarrow \beta.}$$

Exemple 2. On suppose que l'on a les couples. Comment compléter le séquent

$$y : \beta, z : \gamma \vdash \lambda f : \boxed{?}. (f y, f z) : \boxed{?}.$$

- ▷ Pour les types simples, on ne peut pas si $\beta \neq \gamma$.
- ▷ Mais, avec polymorphisme, on a :

$$y : \beta, z : \gamma \vdash \lambda f : \forall \alpha \alpha \rightarrow \delta. (f y, f z) : (\forall \alpha \alpha \rightarrow \delta) \rightarrow \delta \times \delta.$$

Exemple 3.

$$\begin{array}{c}
\frac{}{x : \alpha \vdash x : \alpha} \\
\frac{}{\emptyset \vdash \lambda x. x : \alpha \rightarrow \alpha} \\
\frac{}{\emptyset \vdash \lambda x. x : \forall \alpha \alpha \rightarrow \alpha} \\
\frac{}{\emptyset \vdash \lambda x. x : (\forall \beta \beta) \rightarrow (\forall \delta \delta)}.
\end{array}$$

En effet, $(\forall \beta \beta) \rightarrow (\forall \beta \beta) =_{\alpha} (\forall \beta \beta) \rightarrow (\forall \delta \delta)$.

Exemple 4 (Ouch!).

$$\begin{array}{c}
\text{NON! On a } \alpha \in \mathcal{V}\ell(x : \alpha)! \\
\frac{}{x : \alpha \vdash x : \alpha} \\
\frac{}{x : \alpha \vdash x : \forall \alpha \alpha} \\
\frac{}{x : \alpha \vdash x : \beta} \\
\frac{}{\emptyset \vdash \lambda x. x : \alpha \rightarrow \beta} \\
\frac{}{\emptyset \vdash \lambda x. x : \forall \alpha \forall \beta \alpha \rightarrow \beta}.
\end{array}$$

3 Point de vue logique sur le polymorphisme, système F.

On se place du point de vue Curry-Howard. On ajoute deux règles de déduction supplémentaire :

$$\alpha \notin \mathcal{V}\ell(\Gamma) \quad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha A} \forall_I \quad \frac{\Gamma \vdash \forall \alpha A}{\Gamma \vdash A[B/\alpha]} \forall_E.$$

On a, encore, une possibilité d'éliminer les coupures : si $\alpha \notin \mathcal{V}\ell(\Gamma)$,

$$\frac{\frac{\frac{\delta}{\Gamma \vdash A}}{\Gamma \vdash \forall \alpha A} \forall_I}{\Gamma \vdash A[B/\alpha]} \forall_E \quad \rightsquigarrow \quad \frac{\delta[B/\alpha]}{\Gamma \vdash A[B/\alpha]}$$

Ceci correspond, par correspondance de Curry-Howard, à une β -réduction :

$$(\Lambda\alpha M) \textcolor{brown}{B} \rightarrow_{\beta} M[\textcolor{brown}{B}/\alpha].$$

On a aussi un théorème de normalisation forte.

Théorème 1. Si $\Gamma \vdash M : A$ en λ -calcul polymorphe, alors M est fortement normalisant.

Pour démontrer ce théorème, on utilise encore les candidats de réductibilité (il ne faut définir que $\mathcal{R}_{\forall\alpha A}$), mais la preuve est bien plus complexe. En effet, les types polymorphes ont un grand pouvoir expressif (*c.f.* la section suivante).

Le système que l'on a s'appelle *système F*.¹ C'est de la logique du second ordre : on peut quantifier sur les variables propositionnelles. On quantifie donc sur des ensembles de valeurs au lieu de se limiter uniquement à quantifier sur les valeurs.

4 Représentation des connecteurs logiques.

On peut représenter les connecteurs logiques différemment, sans les ajouter explicitement, mais uniquement avec le fragment contenant \rightarrow et \forall .

- ▷ On peut représenter $\perp := \forall\alpha\alpha$. En effet, on a la correspondance suivante :

$$\frac{\Gamma \vdash \forall\alpha\alpha}{\Gamma \vdash A} \forall_I \quad \rightsquigarrow \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_E.$$

- ▷ On peut représenter $\top := \forall\alpha\alpha \rightarrow \alpha$.
- ▷ On peut représenter $A \wedge B := \forall\alpha (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha$:

1. À ne pas confondre avec [Station F](#).

$$\begin{array}{c}
\frac{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash c : A \rightarrow B \rightarrow \alpha}{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash c M : B \rightarrow \alpha} \quad \frac{\Gamma \vdash M : A}{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash M : A} \quad \frac{\Gamma \vdash N : B}{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash M : A} \\
\hline
\frac{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash c M N \alpha}{\Gamma \vdash \lambda c. c M N : (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha} \\
\hline
\Gamma \vdash \lambda c. c M N : \forall \alpha (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha
\end{array}$$

d'où l'introduction du \wedge . Pour l'élimination, si

$$\Gamma \vdash M : \forall \alpha (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha,$$

alors $\Gamma \vdash M (\lambda x. \lambda y. x) : A$ et $\Gamma \vdash M (\lambda x. \lambda y. y) : B$.

5 Le λ -calcul polymorphe, côté programmation.

« **Généricité** ». On peut avoir plusieurs types en même temps. Par exemple, une fonction de tri a un type

$$\forall \alpha (\alpha \rightarrow \alpha \rightarrow \text{bool}) \rightarrow \text{list } \alpha \rightarrow \text{list } \alpha.$$

« **Paramétricité** ». Lorsque l'on a une fonction $f : (\forall \alpha : \alpha) \rightarrow A$, la fonction n'inspecte pas son argument. Elle ne fait que le dupliquer, le passer à d'autres fonctions, le rejeter, mais elle ne peut pas voir les données sous-jacentes. (On exclue ici les fonctions types `Obj.magic`).

Quelques conséquences sur les formes normales :

- ▷ Il n'y a qu'une seule forme normale ayant un type $\forall \alpha \alpha \rightarrow \alpha$.
- ▷ Il n'y a que deux formes normales ayant un type $\forall \alpha \alpha \rightarrow \alpha \rightarrow \alpha$: $\lambda u. \lambda v. u$ et $\lambda u. \lambda v. v$.

On a aussi quelques « théorèmes gratuits », comme par exemple : si $f : \alpha \rightarrow \beta$ est croissante vis à vis de $\text{ordre}_\alpha : \alpha \rightarrow \alpha \rightarrow \text{bool}$ et $\text{ordre}_\beta : \beta \rightarrow \beta \rightarrow \text{bool}$, alors

$$\text{map } f (\text{sort } \text{ordre}_\alpha \text{ } l) = \text{sort } \text{ordre}_\beta (\text{map } f \text{ } l).$$

6 Polymorphisme et inférence de types.

Pour l'inférence de types, on se place dans la représentation implicite du typage polymorphique. On s'intéresse à la question suivante : pour M un λ -terme donné, existe-t-il Γ et A tels que $\Gamma \vdash M : A$?

Cependant, il y a un problème :

Théorème 2 (1992). L'inférence de types pour le typage polymorphique implicite est *indécidable*.

Ceci conclut l'étude du Système F. On va maintenant s'intéresser à une variante de système F développée en 1978, le λ -calcul avec schéma de types. On note ρ le schéma de types définis par :

$$A ::= \alpha \mid A \rightarrow B \qquad \rho ::= A \mid \forall \alpha \rho.$$

Remarque 1. Il n'y a pas de types de la forme $(\forall \alpha \alpha \rightarrow \alpha) \rightarrow (\forall \beta A)$. La quantification est *préfixe*.

Les termes sont donnés par la grammaire :

$$M ::= x \mid \lambda x. M \mid M N \mid \text{let } x = M \text{ in } N.$$

Pour le système de types, on considère Γ un dictionnaire sur (variables du λ -calcul x , schémas de types ρ). La relation de typage est notée $\Gamma \vdash M : A$ où A est un type *simple*. On la définit par induction à l'aide des règles :

$$\begin{array}{c} \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \\[10pt] \frac{\Gamma \vdash M : A \quad \Gamma, x : \forall \vec{\alpha} A \vdash N : B}{\Gamma \vdash \text{let } x = M \text{ in } N : B} \qquad \frac{\Gamma(x) = \forall \vec{\alpha} A}{\Gamma \vdash x : A[\vec{B}/\vec{\alpha}]} \end{array}$$

$\vec{\alpha} \cap \mathcal{V}\mathcal{E}(\Gamma) = \emptyset$

Ici, on note $\forall \vec{\alpha} A$ pour $\forall \alpha_1 \cdots \forall \alpha_n A$ avec possiblement aucun \forall . De même, la substitution $A[\vec{B}/\vec{\alpha}]$ se fait simultanément.

La règle d'**instantiation** de système F se retrouve partiellement dans la règle de l'axiome. La règle d'**généralisation** de système F se retrouve uniquement dans la règle du `let... = ... in ...`, et uniquement après le `in`.

Exemple 5. Le terme

$$\text{let } f = \lambda x. x \text{ in } f$$

a pour type $\beta \rightarrow \beta$, sans généralisation !

$$\frac{\frac{x : \alpha \vdash x : \alpha}{\emptyset \vdash \lambda x. x : \alpha \rightarrow \alpha} \quad \frac{f : \forall \alpha \alpha \rightarrow \alpha \vdash f : \beta \rightarrow \beta}{\emptyset \vdash \text{let } f = \lambda x. x \text{ in } f : \beta \rightarrow \beta}}{.}$$

En OCaml, lorsqu'on essaie de typer

```
(fun z -> z) (fun x -> x)
```

on se rend compte que l'on n'a pas un type `'a -> 'a`, mais

```
'_weak1 -> '_weak1.
```

Théorème 3. Il existe un algorithme correct et complet pour l'inférence de types pour les schémas de types.

Pour ce faire, on fait comme en **Théorie de la Programmation** [Chapitre 7], où l'on part de M , on génère des contraintes, et à l'aide de l'algorithme d'unification, on trouve le type « le plus général ».