Le λ-calcul polymorphe.

On étend la grammaire des types :

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

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

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

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

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 $\mathscr{V}\ell(A)$, d'α-conversion, et de substitution. Par exemple,

$$\mathscr{V}\ell(\forall\alpha\,\forall\beta\;(\alpha \to \beta \to \gamma \to \alpha)) = \{\gamma\}$$

et

$$\forall\alpha\;\alpha \to \alpha =_\alpha \forall\beta\;\beta \to \beta.$$

1 Typage, version implicite.

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

$$\alpha \notin \mathscr{V}\ell(\Gamma)\ \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall\alpha\, A}\ \mathscr{T}_{\mathsf{g}} \qquad \frac{\Gamma \vdash M : \forall\alpha\, A}{\Gamma \vdash M : A[{}^{B}/_{\alpha}]}\ \mathscr{T}_{\mathsf{i}}.$$

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 :

$$\frac{}{(\Lambda\alpha.\ M)\ A \to_\beta M[{}^{A}/{\alpha}]} \qquad \frac{M \to_\beta M'}{\Lambda\alpha.\ M \to_\beta \Lambda\alpha.\ M'}$$

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

On change également les règles de typage :

$$\Gamma(x) = A\ \frac{}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$x \notin \mathrm{dom}(\Gamma)\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\ M : A \to B}$$

$$\alpha \notin \mathcal{V}\ell(\Gamma)\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \Lambda\alpha.\ M : \forall\alpha\ A} \frac{\Gamma \vdash M : \forall \alpha\ A}{\Gamma \vdash M\ B : A[{}^{B}/{\alpha}]}$$

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

2. $$\dfrac{\dfrac{\dfrac{\vdots}{f : \forall\alpha\ \alpha \to \alpha, z : \beta \vdash f\ \beta\ (f\ \beta\ z) : \beta}}{f : \forall\alpha\ \alpha \to \alpha \vdash \lambda z : \beta.\ f\ \beta\ (f\ \beta\ z) : \beta \to \beta}}{\emptyset \vdash \lambda f : \forall\alpha\ \alpha \to \alpha.\ \lambda z : \beta.\ f\ \beta (f\ \beta\ z) : (\forall\alpha\ \alpha \to \alpha) \to \beta \to \beta.}$$

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 :

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

Exemple 3.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}}{\emptyset \vdash \lambda x.\, x : \alpha \to \alpha}}{\emptyset \vdash \lambda x.\, x : \forall \alpha\, \alpha \to \alpha}}{\emptyset \vdash \lambda x.\, x : (\forall \beta\, \beta) \to (\forall \delta\, \delta).}$$

En effet, $(\forall \beta\, \beta) \to (\forall \beta\, \beta) =_\alpha (\forall \beta\, \beta) \to (\forall \delta\, \delta)$.

Exemple 4 (Ouch !)**.**

$$\dfrac{\dfrac{\dfrac{\text{NON ! On a } \alpha \in \mathscr{V}\ell(x : \alpha) \text{ !}\quad \dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}}{x : \alpha \vdash x : \forall \alpha\, \alpha}}{x : \alpha \vdash x : \beta}}{\emptyset \vdash \lambda x.\, x : \alpha \to \beta}}{\emptyset \vdash \lambda x.\, x : \forall \alpha\, \forall \beta\, \alpha \to \beta.}$$

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 :

$$\alpha \notin \mathscr{V}\ell(\Gamma)\ \dfrac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha\, A}\ \forall_{\mathsf{I}} \qquad \dfrac{\Gamma \vdash \forall \alpha\, A}{\Gamma \vdash A[{}^{B}/_{\alpha}]}\ \forall_{\mathsf{E}}.$$

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

$$\dfrac{\dfrac{\dfrac{\delta}{\Gamma \vdash A}}{\Gamma \vdash \forall \alpha\, A}\ \forall_{\mathsf{I}}}{\Gamma \vdash A[{}^{B}/_{\alpha}]}\ \forall_{\mathsf{E}} \quad \rightsquigarrow \quad \dfrac{\delta[{}^{B}/_{\alpha}]}{\Gamma \vdash A[{}^{B}/_{\alpha}]}$$

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

$$(\Lambda\alpha\ M)\ B \to_\beta M[{}^{B}/_{\alpha}].$$

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 $\to$ et $\forall$.

- On peut représenter $\bot := \forall\alpha\,\alpha$. En effet, on a la correspondance suivante :
$$\dfrac{\Gamma \vdash \forall\alpha\ \alpha}{\Gamma \vdash A}\ \forall_\mathsf{I} \quad\rightsquigarrow\quad \dfrac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_\mathsf{E}.$$
- On peut représenter $\top := \forall\alpha\ \alpha \to \alpha$.
- On peut représenter $A \wedge B := \forall\alpha\ (A \to B \to \alpha) \to \alpha$:

1. À ne pas confondre avec *Station F*.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, c : A \to B \to \alpha \vdash c : A \to B \to \alpha} \quad \dfrac{\Gamma \vdash M : A}{\Gamma, c : A \to B \to \alpha \vdash M : A}}{\Gamma, c : A \to B \to \alpha \vdash c\, M : B \to \alpha} \quad \dfrac{\Gamma \vdash N : B}{\Gamma, c : A \to B \to \alpha \vdash M : A}}{\dfrac{\Gamma, c : A \to B \to \alpha \vdash c\, M\; N\; \alpha}{\Gamma \vdash \lambda c.\, c\, M\; N : (A \to B \to \alpha) \to \alpha}}}{\Gamma \vdash \lambda c.\, c\, M\; N : \forall \alpha\, (A \to B \to \alpha) \to \alpha}$$

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

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

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

$$\forall \alpha\, (\alpha \to \alpha \to \texttt{bool}) \to \texttt{list}\ \alpha \to \texttt{list}\ \alpha.$$

« Paramétricité ». Lorsque l'on a une fonction $f : (\forall \alpha : \alpha) \to A$, 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 $\forall \alpha\, \alpha \to \alpha$.
- Il n'y a que deux formes normales ayant un type $\forall \alpha \alpha \to \alpha \to \alpha$: $\lambda u.\, \lambda v.\, u$ et $\lambda u.\, \lambda v.\, v$.

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

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

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 variant de système F développée en 1978, le *λ-calcul avec schéma de types*. On note ρ le schéma de types définis par :

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

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

Les termes sont donnés par la grammaire :

$$M ::= x \mid \lambda x.\, M \mid M\; N \mid \texttt{let}\; x \texttt{ = } M \;\texttt{ in }\; N.$$

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 :

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\; N : B} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to B}$$

$$\vec{\alpha} \cap \mathscr{V}\!\ell(\Gamma) = \emptyset\;\frac{\Gamma \vdash M : A \qquad \Gamma, x : \forall \vec{\alpha}\, A \vdash N : B}{\Gamma \vdash \texttt{let}\; x \texttt{ = } M \;\texttt{ in }\; N : B} \qquad \Gamma(x) = \forall \vec{\alpha}\, A\;\frac{}{\Gamma \vdash x : A[\vec{B}/\vec{\alpha}]}$$

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

$$\texttt{let}\ f\ \texttt{=}\ \lambda x.\, x\ \ \texttt{in}\ \ f$$

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

$$\dfrac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}}{\emptyset \vdash \lambda x.\, x : \alpha \to \alpha} \qquad \dfrac{}{f : \forall \alpha\ \alpha \to \alpha \vdash f : \beta \to \beta}}{\emptyset \vdash \texttt{let}\ f\ \texttt{=}\ \lambda x.\, x\ \ \texttt{in}\ \ f : \beta \to \beta}.$$

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 ».