

- ▶ Introduction
- ▶ Le système  $F (+ \eta)$   
Les fragments prédictatifs  $F_p$
- ▶ Inférence partielle à la ML dans  $F_p^{\Downarrow}$   
Propagation des annotations  $F_p^{\Uparrow}$  and  $F_p^{\Downarrow}$
- ▶ Une approche plus ambitieuse:  $ML^F$
- ▶ Conclusions  
Références

**Entre ML et le Système F**

*OU*

**Inférence partielle de types  
de second ordre**

Didier Rémy

INRIA-Rocquencourt

Grenoble, Janvier 2005

## Relatif, mais incontestable !

Il est difficile d'en connaître les raisons exacts, mais on connaît les atouts de ML :

- ▶ **Un langage fonctionnel avec une sémantique claire et un système de tyage sûr.**
- ▶ **Un noyau sans effet de bord avec une interprétation logique (isomorphisme de Curry-Howard) ?**
- ▶ **Les types concrets et le filtrage.**
- ▶ **Son polymorphisme.**
- ▶ **L'inférence de type ? (raison sociologique ?)**
- ▶ Sa couche objets (pour OCaml) ? ... un atout surtout sociologique, voire même psychologique ! (Mais des retombées indirectes.)

## ML

- ▶  $\lambda$ -calculus simplement typé avec une construction de liaison

## Synthèse des types très simple...

- ▶ *Intuitivement*, comme le  $\lambda$ -calculus simplement typé (s'appuie sur l'unification d'ordre 1) plus la généralisation aux nœuds de liaison.
- ▶ *En fait*, c'est plus délicat, les preuves sont souvent omises...

## Le langage est pourtant déjà expressif

- ▶ La quantification externe (ML) apporte déjà énormément.

## ML est une sorte d'«optimum local»

- ▶ Beaucoup de variations rentrent parfaitement dans le moule ML.
- ▶ Les autres sont tout de suite compliquées, ou à peine plus expressives.

## Observation

- ▶ 20 ans après, ML is encore à la pointe, mais ses limitations se font plus visibles...

Plusieurs extensions du langage demandentnécessitent ou invitent à du polymorphisme de première classe : objets, monades, GADT... modules ?

(En fait, la plupart ont besoin d'une forme de type existentiels)

Ces extensions sont souvent contraintes («tordues») pour rentrer dans la «boîte» ML.

- ▶ Le système F (types de second-ordre) est puissant et reste assez simple, mais il lui manque l'inférence de types.

## Solution

Conserver l'inférence... autant que possible, en fait, «autant d'inférence que possible», tout en autorisant les types de second ordre !

**Types simples** ou types de premier ordre :

Ce sont les types sans quantificateurs :  $\text{int}$ ,  $\text{int} \rightarrow \text{int}$ ,  $\alpha \rightarrow \alpha$

**Types de second ordre** ou types du système F.

Les quantificateurs sont imbriqués de façon arbitraire, mais ne portent que sur des variables de types :  $\forall \alpha. (\forall \beta. \tau \rightarrow \alpha) \rightarrow \alpha$

**Types d'ordre supérieur** ou types du système  $F^\omega$ .

On peut abstraire également sur des opérateurs de type (d'ordre arbitraire) :  $\forall F. \forall \alpha, \beta. (\alpha \rightarrow \beta) \rightarrow F\alpha \rightarrow F\beta$  qui, en remplaçant  $F$  par  $\Lambda \gamma. \gamma \times \gamma$ , s'instancie en  $(\forall \alpha, \beta. (\alpha \rightarrow \beta) \rightarrow \alpha \times \alpha \rightarrow \beta \times \beta)$

**Polymorphisme de première classe**

Les valeurs de type polymorphe peuvent être manipulées comme les valeurs de types monomorphe.

**Polymorphisme prédicatif/imprédicatif**

Les types polymorphes peuvent être instantiés par des types **moins/aussi** polymorphes. Exemple ? Dans le système F

## Un besoin intrinsèque

- ▶ ML est un langage fonctionnel, cela veut dire que les fonctions ne sont pas des définitions globales mais des valeurs comme les autres qui peuvent être passées en arguments à d'autres fonctions, mémorisées dans des structures de données, *etc.*
- ▶ C'est vrai des fonctions monomorphes, mais c'est faux avec les fonctions polymorphes ! Les mêmes arguments qui justifient le caractère fonctionnel du langage justifient également son extension à du polymorphisme de première classe.
- ▶ On ne veut pas seulement écrire (un exemple du type)

$$\text{let } id = \text{fun } (x) \ x \ \text{in } \dots \ id \ id \ \dots$$

mais

$$(\text{fun } (id) \ \dots \ id \ id \ \dots) \ \dots \ (\text{fun } (x) \ x)$$

(Remplacer *id* par *map*, *fold*, *sort*, ...)

## Un besoin intrinsèque

## Besoin de méthodes polymorphes

Dans le style fonctionnel, *map*, *iter*, *fold* sont des fonctions polymorphes.

Dans le style objet, elles deviennent des méthodes portées par les objets, donc de première classe.

## Un besoin intrinsèque

## Besoin de méthodes polymorphes

## Besoin de types existentiels de première classe...

- ▶ Pour protéger/cacher la représentation empêche de forger des fausses données ne respectant pas les invariants les données.
- ▶ Pour manipuler des données hétérogènes. Par exemple, les calculs gelés (glaçons) sont des paires  $\exists \beta. (\beta \rightarrow \tau) \times \beta$  où le type de l'argument  $\beta$  est caché.

Ils peuvent être placés dans des conteneurs, puis manipulés par des **expressions** polymorphes  $F[...]$  dans `open t0 as y in  $F[t_1]$`

## Un besoin intrinsèque

## Besoin de méthodes polymorphes

## Besoin de types existentiels de première classe...

- ▶ Pour protéger/cacher la représentation empêche de forger des fausses données ne respectant pas les invariants les données.
- ▶ Pour manipuler des données hétérogènes. Par exemple, les calculs gelés (glaçons) sont des paires  $\exists \beta. (\beta \rightarrow \tau) \times \beta$  où le type de l'argument  $\beta$  est caché.

Ils peuvent être placés dans des conteneurs, puis manipulés par des

**fonctions** polymorphes  $f$  dans `fun (f) open t0 as y in f t1`

## Un besoin intrinsèque

## Besoin de méthodes polymorphes

## Besoin de types existentiels de première classe...

- Pour protéger/cacher la représentation empêche de forger des fausses données ne respectant pas les invariants des données.
- Pour manipuler des données hétérogènes. Par exemple, les calculs gelés (glaçons) ont un type de l'argument  $\beta$ . La partie polymorphe de la fonction lui permet d'être appliquée à Ils peuvent être manipulés par des **fonctions** polymorphes  $f$  dans un type ouvert  $t_0$  as y in  $f t_1$

## ...combiné avec du polymorphisme de première classe.

- Si  $w$  a le type  $\exists \beta. \tau[\beta]$ , alors  $f$  doit avoir le type  $\forall \beta. \tau[\beta] \rightarrow \tau'$ , donc l'expression a le type  $(\forall \beta. \tau[\beta] \rightarrow \tau') \rightarrow \tau'$ .

**Un besoin intrinsèque**

**Besoin de méthodes polymorphes**

**Besoin de types existentiels de première classe...**

**...combiné avec du polymorphisme de première classe.**

## **Modules**

- ▶ Les structures contiennent des types abstraits et des fonctions polymorphes .
- ▶ Les foncteurs prennent des structures en arguments.

## *Encodages*

- ▶ *Les codages regorgent de polymorphisme de première classe...*

## Termes non typés, dits à la Curry : comme le $\lambda$ -calcul

$$t ::= x \mid \text{fun } (z) t \mid t_1 t_2$$
$$x ::= z \mid c$$

identificateur      variable      constante

## Types avec quantificateurs imbriqués

$$\sigma ::= \alpha \mid \sigma \rightarrow \sigma \mid \forall \alpha. \sigma$$

Par exemple

$$\forall \alpha \gamma. (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\beta \times \gamma) \rightarrow (\beta \times \gamma) \quad \text{ou} \quad \forall \alpha. \left( \forall \beta. \tau \rightarrow \alpha \right) \rightarrow \alpha$$
$$\approx \exists \beta. \tau$$

e.g.  $\tau$  égal à  $(\beta \rightarrow \text{int}) \times \beta$

## Pas de types principaux

La fonction  $\text{fun } (z) z z$  a les types

$$\left\{ \begin{array}{ll} (\forall \alpha. \alpha) \rightarrow (\forall \alpha. \alpha) & (1) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) & (2) \\ \dots & (3) \end{array} \right.$$

Aucun n'est principal (ne permet de décrire les autres).

C'est toujours un problème pour l'inférence.

## Solution : typage explicite, à la Church

$$t ::= x \mid \text{fun } (z \quad ) t \mid t_1 t_2$$

## Pas de types principaux

La fonction  $\text{fun } (z) z z$  a les types

$$\left\{ \begin{array}{ll} (\forall \alpha. \alpha) \rightarrow (\forall \alpha. \alpha) & (1) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) & (2) \\ \dots & (3) \end{array} \right.$$

Aucun n'est principal (ne permet de décrire les autres).

C'est toujours un problème pour l'inférence.

## Solution : typage explicite, à la Church

$$t ::= x \mid \text{fun } (z : \tau) t \mid t_1 t_2 \mid \text{Fun } (\alpha) t \mid t \tau$$

## Pas de types principaux

La fonction  $\text{fun } (z) z z$  a les types

$$\left\{ \begin{array}{l} (\forall \alpha. \alpha) \rightarrow (\forall \alpha. \alpha) \quad (1) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) \quad (2) \\ \dots \quad (3) \end{array} \right.$$

Aucun n'est principal (ne permet de décrire les autres).  
C'est toujours un problème pour l'inférence.

## Solution : typage explicite, à la Church

$$t ::= x \mid \text{fun } (z : \tau) t \mid t_1 t_2 \mid \text{Fun } (\alpha) t \mid t \tau$$

On peut définir différentes versions

$$\left\{ \begin{array}{l} \text{fun } (z : \forall \alpha. \alpha) z \\ \text{fun } (z : \forall \alpha. \alpha \rightarrow \alpha) z \dots \end{array} \right.$$

**Inconvénient** lourdeur d'écriture (temps et lisibilité).

## Pas de types principaux

La fonction  $\text{fun } (z) z z$  a les types

$$\left\{ \begin{array}{ll} (\forall \alpha. \alpha) \rightarrow (\forall \alpha. \alpha) & (1) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) & (2) \\ \dots & (3) \end{array} \right.$$

Aucun n'est principal (ne permet de décrire les autres).

C'est toujours un problème pour l'inférence.

**Inférence totale ?** Indécidable [Wells, 1994]

## Pas de types principaux

La fonction  $\text{fun } (z) z z$  a les types

$$\left\{ \begin{array}{ll} (\forall \alpha. \alpha) \rightarrow (\forall \alpha. \alpha) & (1) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) & (2) \\ \dots & (3) \end{array} \right.$$

Aucun n'est principal (ne permet de décrire les autres).

C'est toujours un problème pour l'inférence.

**Inférence totale ?** Indécidable [Wells, 1994]

## Semi-algorithmes ?

L'absence de types principaux pose de toute façon un problème de modularité, et la présentation des résultats à l'utilisateur.

## Pas de types principaux

La fonction  $\text{fun } (z) z z$  a les types

$$\left\{ \begin{array}{ll} (\forall \alpha. \alpha) \rightarrow (\forall \alpha. \alpha) & (1) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) & (2) \\ \dots & (3) \end{array} \right.$$

Aucun n'est principal (ne permet de décrire les autres).

C'est toujours un problème pour l'inférence.

## Peut-on retrouver des types principaux ?

- ▶ Modifier la relation de typage tel que certains types parmi (1), (2), (3) ne soient plus valides ?
- ▶ Modifier la relation de typage tel que l'un des types (1), (2), (3) soit meilleur que les autres ?

La difficulté est plus de spécifier ce qu'on ne veut pas typer que de typer ce que l'on veut.

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2 [Pfenning, 1988].

Inférence expressive, mais indécidable (semi-algorithme).

Plus gênant, il faut indiquer où placer les abstractions et les applications de types, même si les types eux-mêmes ne doivent pas être écrits.

$$\text{Fun } (\alpha) \text{ (fun (z : } \forall \beta. \beta \rightarrow \beta) \text{ z } (\alpha \rightarrow \alpha) \text{ z) (Fun } (\gamma) \text{ fun (y : } \gamma) \text{ y)}$$

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2 [Pfenning, 1988].

Inférence expressive, mais indécidable (semi-algorithme).

Plus gênant, il faut indiquer où placer les abstractions et les applications de types, même si les types eux-mêmes ne doivent pas être écrits.

`Fun (?) (fun (z : ?) z (?) z) (Fun (?) fun (y : ?) y)`

Les types sont remplacés par des `?` mais les `?` restent obligatoires

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2 [Pfenning, 1988].  
Inférence expressive, mais indécidable (semi-algorithme).  
Plus gênant, il faut indiquer où placer les abstractions et les applications de types, même si les types eux-mêmes ne doivent pas être écrits.

$$\text{Fun } (?) \text{ (fun (z : ?) z } (?) \text{ z) (Fun } (?) \text{ fun (y : ?) y)}$$

- ▶ En faisant seulement de l'inférence locale [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupides» :

$$\left( \text{fun ( f : } \textit{int} \rightarrow \textit{int} \text{ ) fun (z) f z} \right) \text{ (fun ( x ) x + 1)}$$

Les contraintes de typage impliquent *localement* que  $x$  a le type  $\textit{int}$ , donc  $x$  n'a pas besoin d'une annotation explicite.

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2.
- ▶ En faisant seulement de l'inférence locale [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupidés» :

$$\left( \text{fun } (f : \textit{int} \rightarrow \textit{int}) \text{ fun } (z) f z \right) \left( \text{fun } (x) x + 1 \right)$$

Les contraintes de typage impliquent *localement* que  $x$  a le type  $\textit{int}$ , donc  $x$  n'a pas besoin d'une annotation explicite.

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2.
- ▶ En faisant seulement de l'inférence locale [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupidés» :

$$\left( \text{fun } (f : \boxed{\text{int}} \rightarrow \text{int}) \text{ fun } (z) f z \right) \boxed{\text{fun } (x) x + 1}$$

Les contraintes de typage impliquent *localement* que  $x$  a le type  $\text{int}$ , donc  $x$  n'a pas besoin d'une annotation explicite.

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2.
- ▶ En faisant seulement de l'inférence locale [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupidés» :

$$\left( \text{fun } ( f : \text{int} \rightarrow \text{int} ) \text{ fun } ( z ) f z \right) \left( \text{fun } ( x ) x + 1 \right)$$

Les contraintes de typage impliquent *localement* que  $x$  a le type  $\text{int}$ , donc  $x$  n'a pas besoin d'une annotation explicite.

- ▷ La notion de localité à un caractère arbitraire (où s'arrêter?).
- ▷ En fait, il n'est pas toujours évident de savoir où il faut annoter.
- ▷ Ce n'est pas une extension conservative de ML. (L'unification transporte l'information sans restriction de localité.)

## Partir du Système F et s'approcher de ML

- ▶ En utilisant l'unification d'ordre 2.
- ▶ En faisant seulement de l'inférence locale [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupidés» :

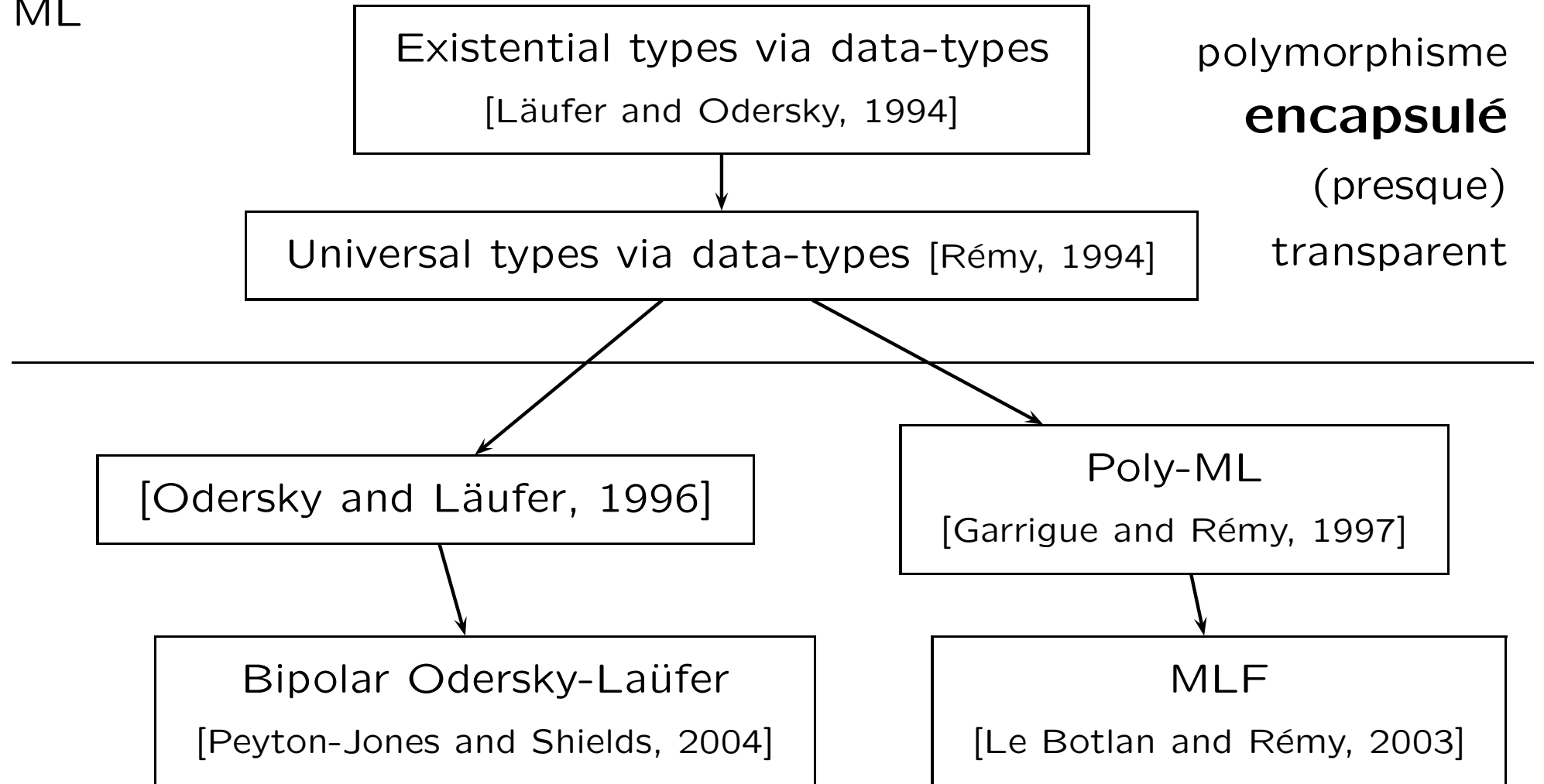
$$\left( \text{fun } ( f : \textit{int} \rightarrow \textit{int} ) \text{ fun } ( z ) f z \right) \left( \text{fun } ( x ) x + 1 \right)$$

Les contraintes de typage impliquent *localement* que  $x$  a le type  $\textit{int}$ , donc  $x$  n'a pas besoin d'une annotation explicite.

- ▷ La notion de localité à un caractère arbitraire (où s'arrêter?).
- ▷ En fait, il n'est pas toujours évident de savoir où il faut annoter.
- ▷ Ce n'est pas une extension conservative de ML. (L'unification transporte l'information sans restriction de localité.)

## Partir de ML pour atteindre tout le Système F ...

ML



Prédicatif...

vs

Imprédicatif...

Polymorphism de second ordre

C'est le polymorphisme du pauvre : on utilise un constructeur pour faire automatiquement une *injection* d'un type polymorphe en un type ML et la *projection* dans le sens inverse.

```
type id  $\alpha = Id$  of  $\forall \alpha. (\alpha \rightarrow \alpha)$ 
```

Il suffit alors de mentionner le constructeur à l'introduction et à l'élimination (filtrage) :

```
let id = Id (fun  $x \rightarrow x$ )
```

```
let auto (Id  $f$ ) =  $f$   $f$ 
```

```
auto id
```

C'est le polymorphisme du pauvre : on utilise un constructeur pour faire automatiquement une *injection* d'un type polymorphe en un type ML et la *projection* dans le sens inverse.

$$\text{type } id \ \alpha = Id \text{ of } \forall \alpha. (\alpha \rightarrow \alpha)$$

## Limitations

- ▶ Facile pour les cas simples, mais complexe à mettre en œuvre avec des quantificateurs imbriqués.
- ▶ Un type a plusieurs décompositions possibles qui ne sont pas équivalentes.
- ▶ Lourd pour un usage intensif :
  - ▷ déclaration de type nécessaire, même pour un usage unique
  - ▷ types peu lisibles (il faut donner un nom à chaque niveau de quantificateurs).

► Introduction

► Le système  $F (+ \eta)$

Les fragments prédictifs  $F_p$

► Inférence partielle à la ML dans  $F_p^{\Downarrow}$

Propagation des annotations  $F_p^{\Uparrow}$  and  $F_p^{\Updownarrow}$

► Une approche plus ambitieuse:  $ML^F$

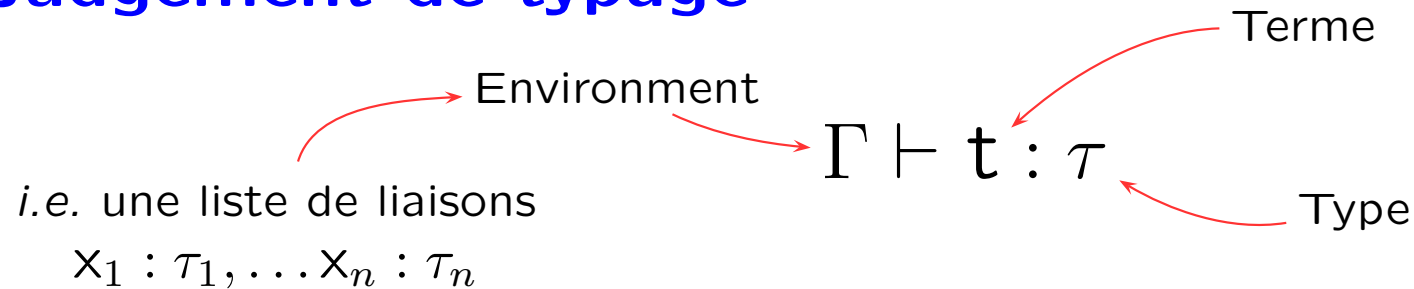
► Conclusions

Références

## Système $F^X$

- ▶ C'est le système  $F$ , *i.e.* un système avec types de second ordre,
- ▶ paramétrisé par une relation d'instanciation  $\leq_X$  entre les types.

## Judgement de typage



## Règle d'inférence

$$\frac{\mathcal{R} \quad P_1 \quad \dots \quad P_n}{C}$$

C'est une implication  $P_1 \wedge \dots \wedge P_n \implies C$ .

On parle de règle de typage lorsque la conclusion est un jugement de typage.

## Système de typage

C'est un ensemble  $\mathcal{R}$  de règles de typages qui définit le typage comme la plus petite relation satisfaisant les règles.

## Règle d'inférence

$$\frac{\begin{array}{ccc} \mathbf{R} & & \\ P_1 & \dots & P_n \end{array}}{C}$$

C'est une implication  $P_1 \wedge \dots \wedge P_n \implies C$ .

On parle de règle de typage lorsque la conclusion est un jugement de typage.

## Système de typage

C'est un ensemble  $\mathcal{R}$  de règles de typages qui définit le typage comme la plus petite relation satisfaisant les règles.

$$\frac{\frac{\overline{P_{11}} \quad \overline{P_{12}}}{C_{11}} \quad \overline{P_2}}{C_2}}$$

## Une dérivation

C'est un arbre de preuve, les feuilles sont des axiomes de typage, la conclusion est un jugement de typage valide.

$$\begin{array}{c} \text{Var} \\ \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \end{array} \qquad \begin{array}{c} \text{App} \\ \frac{\Gamma \vdash a_1 : \sigma_2 \rightarrow \sigma_1 \quad \Gamma \vdash a_2 : \sigma_2}{\Gamma \vdash a_1 a_2 : \sigma_1} \end{array} \qquad \begin{array}{c} \text{Fun} \\ \frac{\Gamma, z : \sigma \vdash a : \sigma'}{\Gamma \vdash \text{fun } (z) a : \sigma \rightarrow \sigma'} \end{array}$$

$$\begin{array}{c} \text{Gen} \\ \frac{\Gamma \vdash a : \sigma \quad \alpha \notin \text{ftv}(\Gamma)}{\Gamma \vdash a : \forall \alpha. \sigma} \end{array}$$

$$\begin{array}{c} \text{Inst} \\ \frac{\Gamma \vdash a : \sigma \quad \sigma \leq_X \sigma'}{\Gamma \vdash a : \sigma'} \end{array}$$

- ▶ Les règles sont très simples !
- ▶ Elles sont paramétrisées par une relation d'instanciation  $\leq_X$  (aussi appelée type containment)

Soit  $\sigma_{id}$  le schéma  $\forall \alpha. \alpha \rightarrow \alpha$ .

$$\begin{array}{c} \text{Var} \frac{}{} \\ \text{Sub} \frac{z : \sigma_{id} \vdash z : \sigma_{id} \quad \sigma_{id} \leq \sigma_{id} \rightarrow \sigma_{id} \text{ (1)}}{} \\ \text{App} \frac{}{} \\ \text{Fun} \frac{z : \sigma_{id} \vdash z z : \sigma_{id}}{\vdash \text{fun } (z) z z : \sigma_{id} \rightarrow \sigma_{id}} \end{array} \quad \frac{}{} \text{Var}$$

Soit  $\sigma_{id}$  le schéma  $\forall \alpha. \alpha \rightarrow \alpha$ .

$$\begin{array}{c} \text{Var} \frac{}{} \\ \text{Sub} \frac{z : \sigma_{id} \vdash z : \sigma_{id} \quad \sigma_{id} \leq \sigma_{id} \rightarrow \sigma_{id} \text{ (1)}}{} \\ \text{App} \frac{}{} \\ \text{Fun} \frac{z : \sigma_{id} \vdash z z : \sigma_{id}}{\vdash \text{fun } (z) z z : \sigma_{id} \rightarrow \sigma_{id}} \end{array} \quad \frac{}{} \text{Var}$$

Une autre ?

Soit  $\sigma_{id}$  le schéma  $\forall \alpha. \alpha \rightarrow \alpha$ .

$$\begin{array}{c}
 \text{Var} \frac{}{} \\
 \text{Sub} \frac{z : \sigma_{id} \vdash z : \sigma_{id} \quad \sigma_{id} \leq \sigma_{id} \rightarrow \sigma_{id} \quad (1)}{z : \sigma_{id} \vdash z : \sigma_{id} \rightarrow \sigma_{id}} \\
 \text{App} \frac{}{z : \sigma_{id} \vdash z z : \sigma_{id}} \\
 \text{Fun} \frac{z : \sigma_{id} \vdash z z : \sigma_{id}}{\vdash \text{fun } (z) z z : \sigma_{id} \rightarrow \sigma_{id}}
 \end{array}$$

**Une autre ?** Soit  $\tau_\alpha$  le type  $\alpha \rightarrow \alpha$ .

$$\begin{array}{c}
 \text{Var} \frac{}{} \\
 \text{Sub} \frac{z : \sigma_{id} \vdash z : \sigma_{id} \quad \sigma_{id} \leq \tau_\alpha \rightarrow \tau_\alpha}{z : \sigma_{id} \vdash z : \tau_\alpha} \\
 \text{App} \frac{}{z : \sigma_{id} \vdash z z : \tau_\alpha} \\
 \text{Gen} \frac{z : \sigma_{id} \vdash z z : \tau_\alpha \quad \alpha \notin \text{ftv}(z : \sigma_{id})}{z : \sigma_{id} \vdash z z : \sigma_{id}} \\
 \text{Fun} \frac{z : \sigma_{id} \vdash z z : \sigma_{id}}{\vdash \text{fun } (z) z z : \sigma_{id} \rightarrow \sigma_{id}}
 \end{array}$$

$\leq$  pour le Système F

La plus petite relation  $\leq$  qui satisfasse la règle :

Sub

$$\frac{\bar{\beta} \notin \text{ftv}(\forall \bar{\alpha}. \sigma)}{\forall \bar{\alpha}. \sigma \leq \forall \bar{\beta}. \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Par exemple,  $\forall \alpha. \alpha \rightarrow \alpha \leq \begin{cases} \text{int} \rightarrow \text{int} \\ \forall \alpha, \beta. (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta) \\ (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow (\forall \alpha. \alpha \rightarrow \alpha) \end{cases} \quad (1)$

En fait, on généralise le système de typage, pour autoriser plus d'instanciation implicite. Par exemple

Si  $f : \forall \alpha. \tau \rightarrow \tau'$ , alors  $f : (\forall \alpha. \tau) \rightarrow \tau'[\text{int}/\alpha]$ ,

Contrainte plus exigeante  
sur l'argument

Contrainte moins exigeante  
sur le résultat

$\leq^\eta$  pour le Système  $F^\eta$

La plus petite relation  $\leq$  qui satisfasse la règle :

Sub

$$\frac{\bar{\beta} \notin \text{ftv}(\forall \bar{\alpha}. \sigma)}{\forall \bar{\alpha}. \sigma \leq \forall \bar{\beta}. \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Trans

$$\frac{\sigma \leq \sigma' \quad \sigma' \leq \sigma''}{\sigma \leq \sigma''}$$

Arrow

$$\frac{\sigma'_1 \leq \sigma_1 \quad \sigma_2 \leq \sigma'_2}{\sigma_1 \rightarrow \sigma_2 \leq \sigma'_1 \rightarrow \sigma'_2}$$

Contra-variance

All

$$\frac{\sigma \leq \sigma'}{\forall \alpha. \sigma \leq \forall \alpha. \sigma'}$$

Distrib

$$\forall \alpha. \sigma \rightarrow \sigma' \leq (\forall \alpha. \sigma) \rightarrow \forall \alpha. \sigma'$$

Une expression d'un type flèche  $\tau_1 \rightarrow \tau_2$

— attend une expression de type  $\tau_1$ ,

peut recevoir une expression de type  $\tau'_1 \leq \tau_1$  ;

— retourne une expression de type  $\tau_2$ ,

qui est aussi de type  $\tau'_2 \geq \tau_2$ .

$\leq^\eta$  pour le Système  $F^\eta$

La plus petite relation  $\leq$  qui satisfasse la règle :

Sub

$$\frac{\bar{\beta} \notin \text{ftv}(\forall \bar{\alpha}. \sigma)}{\forall \bar{\alpha}. \sigma \leq \forall \bar{\beta}. \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Trans

$$\frac{\sigma \leq \sigma' \quad \sigma' \leq \sigma''}{\sigma \leq \sigma''}$$

Arrow

$$\frac{\sigma'_1 \leq \sigma_1 \quad \sigma_2 \leq \sigma'_2}{\sigma_1 \rightarrow \sigma_2 \leq \sigma'_1 \rightarrow \sigma'_2}$$

Contra-variance

All

$$\frac{\sigma \leq \sigma'}{\forall \alpha. \sigma \leq \forall \alpha. \sigma'}$$

Distrib

$$\forall \alpha. \sigma \rightarrow \sigma' \leq (\forall \alpha. \sigma) \rightarrow \forall \alpha. \sigma'$$

$\alpha \notin \text{ftv}(\sigma')$

$\alpha \notin \text{ftv}(\sigma)$

Distrib-Left

$$\forall \alpha. \sigma \rightarrow \sigma' \leq (\forall \alpha. \sigma) \rightarrow \sigma'$$

Distrib-Right

$$\forall \alpha. \sigma \rightarrow \sigma' \leq \sigma \rightarrow \forall \alpha. \sigma'$$

(also  $\geq$  by Sub+All)

$\leq^\eta$  pour le Système  $F^\eta$  = Système  $F$  modulo  $\eta$ -expansion.

La plus petite relation  $\leq$  qui satisfasse la règle :

Sub

$$\frac{\bar{\beta} \notin \text{ftv}(\forall \bar{\alpha}. \sigma)}{\forall \bar{\alpha}. \sigma \leq \forall \bar{\beta}. \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Trans

$$\frac{\sigma \leq \sigma' \quad \sigma' \leq \sigma''}{\sigma \leq \sigma''}$$

Arrow

$$\frac{\sigma'_1 \leq \sigma_1 \quad \sigma_2 \leq \sigma'_2}{\sigma_1 \rightarrow \sigma_2 \leq \sigma'_1 \rightarrow \sigma'_2}$$

Contra-variance

All

$$\frac{\sigma \leq \sigma'}{\forall \alpha. \sigma \leq \forall \alpha. \sigma'}$$

Distrib

$$\forall \alpha. \sigma \rightarrow \sigma' \leq (\forall \alpha. \sigma) \rightarrow \forall \alpha. \sigma'$$

$\tau \leq \tau'$  ssi il existe une fonction de retypage  $f$  (i.e. telle que  $f =_\eta id$ ) de type  $\tau \rightarrow \tau'$ .

## Correction du typage

La réduction *préserve* le typage dans les deux systèmes  $F$  et  $F^\eta$ .  
Propriété de *progression* si on ajoute des constantes.

*Note* : En présence d'effets de bord, il faut retirer *Distrib-Right*.

## Inférence de type

- ▶ Aucun des deux systèmes n'admet l'inférence
- ▶ La relation  $\leq^\eta$  n'est elle-même pas décidable.

Suggestion de John Mitchell en 1984 :  $F^\eta$  est mieux adapté à l'inférence parce qu'il admet plus de types principaux.

## Pistes pour l'inférence

- ▶ Faire des restrictions (typer moins de programmes)
- ▶ Ajouter des annotations (aider l'inférence).

# $F_p$ : Le fragment prédictatif du Système F

18(1)/50

---

C'est un meilleur candidat pour l'inférence de types à la ML.

Dans le système  $F_p$  on distingue :

$$\begin{array}{ll} \tau ::= \alpha \mid \tau \rightarrow \tau & \text{monotypes} \\ \sigma ::= \tau \mid \sigma \rightarrow \sigma \mid \forall \alpha. \sigma & \text{(poly)types} \end{array}$$

Les polytypes ne sont pas restreints, mais...

les variables de types ne peuvent être instantiées que par des monotypes.

La règle **Sub** est remplacée par :

$$\text{Sub}_p \quad \frac{\bar{\beta} \notin \text{ftv}(\forall \bar{\alpha}. \sigma)}{\forall \bar{\alpha}. \sigma \leq \forall \bar{\beta}. \sigma[\bar{\tau}/\bar{\alpha}]}$$

Au lieu de  $\bar{\sigma}$

C'est un meilleur candidat pour l'inférence de types à la ML.

Dans le système  $F_p$  on distingue :

$$\begin{array}{ll} \tau ::= \alpha \mid \tau \rightarrow \tau & \text{monotypes} \\ \sigma ::= \tau \mid \sigma \rightarrow \sigma \mid \forall \alpha. \sigma & \text{(poly)types} \end{array}$$

Les polytypes ne sont pas restreints, mais...

les variables de types ne peuvent être instantiées que par des monotypes.

La règle **Sub** est remplacée par :

$$\text{Sub}_p \quad \frac{\bar{\beta} \notin \text{ftv}(\forall \bar{\alpha}. \sigma)}{\forall \bar{\alpha}. \sigma \leq \forall \bar{\beta}. \sigma[\bar{\tau}/\bar{\alpha}]}$$

Au lieu de  $\bar{\sigma}$

## La restriction est significative

- ▶ Le codage des existentiels :

Lorsqu'on cache  $\sigma[\tau/\beta]$  en  $\exists \beta. \sigma$ , alors  $\tau$  doit être un monotype.

On ne peut pas cacher des sous-expressions contenant des parties déjà cachées.

- ▶ Codage des objects

Les objets sont (au moins) aussi compliqués que

$$\exists \alpha_R. (\alpha_R \times (\alpha_R \rightarrow \sigma_M))$$

où  $\alpha_R$  cache l'état interne. L'état devrait être monomorphique, *i.e.* ne pas contenir d'autres objets.

## La restriction est significative

- ▶ apply (or map, iter, *etc.*)

```
let apply = fun (f) fun (z) f z in ...
```

n'acceptent que des arguments monomorphes...

Besoin d'une version pour chaque *forme* de polymorphisme de f!

En particulier d'une version pour chaque *forme* de z, ce qui est déjà très embêtant!

(tous les types abstraits sont incompatibles)

## La restriction est significative

## Elle peut/doit être combinée avec le polymorphisme encapsulé

Le fragment imprédicatif peut être recouvert avec les type de donnés, comme auparavant.

Pas très élégant. Mais mieux qu'une seule des deux extensions.

**Mais c'est (très) simple...**

► Introduction

► Le système  $F (+ \eta)$

Les fragments prédictifs  $F_p$

► Inférence partielle à la ML dans  $F_p^{\Downarrow}$

Propagation des annotations  $F_p^{\Uparrow}$  and  $F_p^{\Downarrow}$

► Une approche plus ambitieuse:  $ML^F$

► Conclusions

Références

## Expressions

ajout d'une construction let

$t ::= x \mid \text{fun } (z) t \mid t_1 \ t_2 \mid \text{let } z = t_1 \ \text{in } t_2$

## Expressions

ajout d'annotations

$t ::= x \mid \text{fun } (z) t \mid t_1 (t_2 : \theta) \mid \text{let } z = (t_1 : \theta) \text{ in } t_2$

## Annotations

$\theta ::= \exists \bar{\beta}. \sigma$

Les annotations **spécifient la forme de la partie polymorphe des types** et laissent l'inférence **deviner la partie monomorphe**.

Aussi,  $\exists \bar{\beta}.$  a autant un rôle clé (en laissant de la place à l'inférence) que  $\sigma$ .

La partie **monomorphe** de la structure est toujours sous une partie (éventuellement vide) de la partie **polymorphe** de la structure.

Une annotation vide est  $\exists \beta. \beta$

Var

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

Inst

$$\frac{\Gamma \vdash t : \sigma' \quad \sigma' \leq \sigma}{\Gamma \vdash t : \sigma}$$

Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

App

$$\frac{\Gamma \vdash t_1 : \tau_2 \quad \rightarrow \tau_1 \quad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash t_1 t_2 : \tau_1}$$

Fun

$$\frac{\Gamma, z : \tau_2 \vdash t : \tau_1}{\Gamma \vdash \text{fun } (z) t : \tau_2 \rightarrow \tau_1}$$

Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \tau_1 \quad \Gamma, x : \forall \bar{\alpha}. \tau_1 \vdash t_2 : \tau_2}{\Gamma \vdash \text{let } z = t_1 \text{ in } t_2 : \tau_2}$$

**Var**

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

**Inst**

$$\frac{\Gamma \vdash t : \sigma' \quad \sigma' \leq \sigma}{\Gamma \vdash t : \sigma}$$

**Gen**

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

**App**

$$\frac{\Gamma \vdash t_1 : \tau_2[\bar{\tau}/\bar{\beta}] \rightarrow \tau_1 \quad \Gamma \vdash t_2 : \tau_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \tau_2) : \tau_1}$$

**Fun**

$$\frac{\Gamma, z : \tau_2 \vdash t : \tau_1}{\Gamma \vdash \text{fun } (z) t : \tau_2 \rightarrow \tau_1}$$

**Let**

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \tau_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \tau_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \tau_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \tau_1) \text{ in } t_2 : \tau_2}$$

$$\begin{array}{c}
 \text{Var} \\
 \hline
 \Gamma \vdash x : \sigma \\
 \hline
 \Gamma \vdash x : \sigma
 \end{array}
 \qquad
 \begin{array}{c}
 \text{Inst} \\
 \hline
 \Gamma \vdash t : \sigma' \quad \sigma' \leq_p^{\parallel} \sigma \\
 \hline
 \Gamma \vdash t : \sigma
 \end{array}
 \qquad
 \begin{array}{c}
 \text{Gen} \\
 \hline
 \Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma) \\
 \hline
 \Gamma \vdash t : \forall \bar{\alpha}. \sigma
 \end{array}$$

$$\begin{array}{c}
 \text{App} \\
 \hline
 \Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}] \\
 \hline
 \Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1
 \end{array}
 \qquad
 \begin{array}{c}
 \text{Fun} \\
 \hline
 \Gamma, z : \sigma_2 \vdash t : \sigma_1 \\
 \hline
 \Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1
 \end{array}$$

$$\begin{array}{c}
 \text{Let} \\
 \hline
 \Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2 \\
 \hline
 \Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2
 \end{array}$$

Lire  $\Gamma \vdash t : \sigma$  comme des règles de vérification :  $\Gamma$ ,  $t$ , and  $\sigma$  étant donnés.

Tous les types sont de la forme  $\sigma[\bar{\tau}/\bar{\beta}]$  où  $\sigma$  n'est jamais deviné alors que  $\bar{\tau}$  l'est.

On obtient ML quand tous les  $\sigma$  sont des  $\beta$ .

<p><b>Var</b></p> $\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$	<p><b>Inst</b></p> $\frac{\Gamma \vdash t : \sigma' \quad \sigma' \leq_p^{\parallel} \sigma}{\Gamma \vdash t : \sigma}$	<p><b>Gen</b></p> $\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$
--	---	--

<p><b>App</b></p> $\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1}$	<p><b>Fun</b></p> $\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$
---	--

**Let**

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

Pour préparer l'inférence :

Mettre les dérivations en forme canonique, dirigée par la syntaxe.

<p><b>Var</b></p> $\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$	<p><b>Inst</b></p> $\frac{\Gamma \vdash t : \sigma' \quad \sigma' \leq_p^{\parallel} \sigma}{\Gamma \vdash t : \sigma}$
--	---

**Gen**

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

**App**

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1}$$

**Fun**

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

**Let**

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

Fusionner les deux règles.

$\text{Inst}(R(D)) \rightsquigarrow R(\text{Inst}(D))$ , sauf si R est Var.

## Var-Inst

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \sigma}{\Gamma \vdash x : \sigma}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

## Var-Inst

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \sigma}{\Gamma \vdash x : \sigma}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

Changer  $\sigma$  en  $\rho$ .

A  $\rho$  est un  $\sigma$  sans les quantificateurs externes.

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \sigma_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \sigma_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

Changer  $\sigma_1$  en  $\rho_1$ , puisque

$$\text{App}(D_1, D_2) \rightsquigarrow \text{Gen}(\text{App}(\text{Inst}(D_1), D_2))$$

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \rho_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \rho_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let

$$\frac{\Gamma \vdash t_1 : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, x : \forall \bar{\alpha}. \sigma_1[\bar{\tau}/\bar{\beta}] \vdash t_2 : \sigma_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \sigma_2}$$

Changer  $\sigma_2$  into  $\rho_2$ .

$$\text{Let}(D_1, D_2) \rightsquigarrow \text{Gen}(\text{Let}(D_1, \text{Inst}(D_2)))$$

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \rho_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma_1[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2}$$

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \rho_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma_1[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2}$$

**Gen** reste (alors qu'elle disparaît en ML)

Elle peut être utilisée dans la prémisse d'une règle **Fun** ou la prémisse gauche d'une règle **Let**.

## Var-Inst-Rho

$$\frac{x : \sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

## Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

## App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \rightarrow \rho_1 \quad \Gamma \vdash t_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1}$$

## Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma_1[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma_1[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho_2}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2}$$

Regardons Let-Gen :

- Le bleu est explicite,
- Le rouge est inféré.

- ▶ On retire la règle **Distrib**

Raison (a posteriori)

- ▶ La règle **Distrib-Right** n'est pas correcte pour les effets de bord.
- ▶ La relation  $\leq_p^{\parallel}$  est décidable (efficacement).  $\leq_p^{\eta}$  l'est-elle ?
- ▶ Conséquence :  $\leq_p^{\parallel}$  est une sous-relation stricte de  $\leq_p^{\eta}$  (contient  $\leq_F$ ).  
Donc  $F_p^{\Downarrow}$  est moins expressif que  $F_p^{\eta}$ .
- ▶ Ensemble équivalent de règles :

**Inst-Refl**

$$\sigma \leq \sigma$$

**Inst-Arrow**

$$\frac{\sigma_1 \leq \sigma'_1 \quad \sigma'_2 \leq \sigma_2}{\sigma_2 \rightarrow \sigma_1 \leq \sigma'_2 \rightarrow \sigma'_1}$$

**Inst-Skol**

$$\frac{\sigma \leq \sigma' \quad \alpha \notin \text{ftv}(\sigma)}{\sigma \leq \forall \alpha. \sigma'}$$

**Inst-Spec**

$$\frac{\sigma[\tau/\alpha] \leq \sigma'}{\forall \alpha. \sigma \leq \sigma'}$$

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

(Ou n'importe quelque autre algorithme d'inférence) ▷

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## Interprétation logique des contraintes

- 1 Interprétation standard de  $\exists$ ,  $\forall$ ,  $\wedge$ .
- 2 Les contraintes let sont interprétées par macro-expansion
- 3  $(\forall \bar{\beta} [C]. \sigma) \preceq \sigma'$  (qui apparait alors) signifie  $\exists \bar{\beta}. (C \wedge \sigma \leq \sigma')$
- 4 Les contraintes  $\leq$  sont interprétées par  $\leq_p^{\parallel}$

Voir [Pierce, 2004].

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## Interprétation logique des contraintes

- 1 Interprétation standard de  $\exists$ ,  $\forall$ ,  $\wedge$ .
- 2 Les contraintes let sont interprétées par macro-expansion
- 3  $(\forall \bar{\beta} [C]. \sigma) \preceq \sigma'$  (qui apparait alors) signifie  $\exists \bar{\beta}. (C \wedge \sigma \leq \sigma')$
- 4 Les contraintes  $\leq$  sont interprétées par  $\leq_p^{\parallel}$

Voir [Pierce, 2004].

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## Interprétation logique des contraintes

- 1 Interprétation standard de  $\exists$ ,  $\forall$ ,  $\wedge$ .
- 2 Les contraintes let sont interprétées par macro-expansion
- 3  $(\forall \bar{\beta} [C]. \sigma) \preceq \sigma'$  (qui apparait alors) signifie  $\exists \bar{\beta}. (C \wedge \sigma \leq \sigma')$
- 4 Les contraintes  $\leq$  sont interprétées par  $\leq_p^{\parallel}$

Voir [Pierce, 2004].

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## Interprétation logique des contraintes

- 1 Interprétation standard de  $\exists$ ,  $\forall$ ,  $\wedge$ .
- 2 Les contraintes let sont interprétées par macro-expansion
- 3  $(\forall \bar{\beta} [C]. \sigma) \preceq \sigma'$  (qui apparait alors) signifie  $\exists \bar{\beta}. (C \wedge \sigma \leq \sigma')$
- 4 Les contraintes  $\leq$  sont interprétées par  $\leq_p^{\parallel}$

Voir [Pierce, 2004].

$$\tau \leq \tau' \longrightarrow \tau = \tau'$$

$$\sigma_1 \rightarrow \sigma_2 \leq \sigma'_1 \rightarrow \sigma'_2 \longrightarrow \sigma'_1 \leq \sigma_1 \wedge \sigma_2 \rightarrow \sigma'_2$$

$$\forall \alpha. \sigma \leq \rho \longrightarrow \exists \alpha. (\sigma \leq \rho)$$

$$\sigma \leq \forall \alpha. \sigma' \longrightarrow \forall \alpha. (\sigma \leq \sigma')$$

- ▶ Suit les règles dirigées par la syntaxe pour  $\leq_p^{\parallel}$ .
- ▶ Permet de simplifier les contraintes en contraintes d'égalités.

## Interprétation logique des contraintes

- 1 Interprétation standard de  $\exists, \forall, \wedge$ .
- 2 Les contraintes let sont interprétées par macro-expansion
- 3  $(\forall \bar{\beta}[C].\sigma) \preceq \sigma'$  (qui apparait alors) signifie  $\exists \bar{\beta}. (C \wedge \sigma \leq \sigma')$
- 4 Les contraintes  $\leq$  sont interprétées par  $\leq_p^{\parallel}$

Voir [Pierce, 2004].

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \text{ t} : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \text{ t} : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \text{ t} : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket \text{t} : \sigma_1 \rrbracket$$

$$\llbracket \text{t}_1 (\text{t}_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket \text{t}_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket \text{t}_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (\text{t}_1 : \exists \bar{\beta}. \sigma_1) \text{ in } \text{t}_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket \text{t}_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket \text{t}_2 : \rho_2 \rrbracket$$

$$\llbracket \text{t} : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket \text{t}_2 : \rho \rrbracket$$

## Interprétation logique des contraintes

- 1 Interprétation standard de  $\exists$ ,  $\forall$ ,  $\wedge$ .
- 2 Les contraintes let sont interprétées par macro-expansion
- 3  $(\forall \bar{\beta} [C]. \sigma) \preceq \sigma'$  (qui apparait alors) signifie  $\exists \bar{\beta}. (C \wedge \sigma \leq \sigma')$
- 4 Les contraintes  $\leq$  sont interprétées par  $\leq_p^{\parallel}$

Voir [Pierce, 2004].

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## L'inférence est de premier ordre

- ▶ Pas de méta variables pour  $\sigma$  ni  $\rho$ , mais seulement pour  $\tau$ .
- ▶ Les formes de la structure polymorphe ne sont que vérifiées.

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## Un problème d'inférence

$\Gamma \vdash t : \sigma$  est résolu comme  
let  $\Gamma$  in  $\llbracket t : \sigma \rrbracket$

$$\llbracket x : \rho \rrbracket \longrightarrow x \preceq \rho$$

$$\llbracket \text{fun } (z) \ t : \alpha \rrbracket \longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha)$$

$$\llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket \longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket$$

$$\llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket \longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket)$$

$$\llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket \longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket$$

$$\llbracket t : \forall \bar{\alpha}. \rho \rrbracket \longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket$$

## Un problème d'inférence

trouver  $\varphi$  tel que  $\varphi(\Gamma) \vdash t : \varphi(\sigma)$  est résolu comme

trouvé  $\varphi$  tel que  $\varphi \vdash \text{let } \Gamma \text{ in } \llbracket t : \sigma \rrbracket$

Cet algorithme est correct et complet.

$$\begin{aligned}
 \llbracket x : \rho \rrbracket &\longrightarrow x \preceq \rho \\
 \llbracket \text{fun } (z) \ t : \alpha \rrbracket &\longrightarrow \exists \beta_1 \beta_2. (\llbracket \text{fun } (z) \ t : \beta_1 \rightarrow \beta_2 \rrbracket \wedge \beta_1 \rightarrow \beta_2 \leq \alpha) \\
 \llbracket \text{fun } (z) \ t : \sigma_2 \rightarrow \sigma_1 \rrbracket &\longrightarrow \text{let } z : \sigma_2 \text{ in } \llbracket t : \sigma_1 \rrbracket \\
 \llbracket t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rrbracket &\longrightarrow \exists \bar{\beta}. (\llbracket t_1 : \sigma_2 \rightarrow \rho_1 \rrbracket \wedge \llbracket t_2 : \sigma_2 \rrbracket) \\
 \llbracket \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_2 \rrbracket &\longrightarrow \text{let } z : \forall \bar{\beta} [\llbracket t_1 : \sigma_1 \rrbracket]. \sigma_1 \text{ in } \llbracket t_2 : \rho_2 \rrbracket \\
 \llbracket t : \forall \bar{\alpha}. \rho \rrbracket &\longrightarrow \forall \bar{\alpha}. \llbracket t_2 : \rho \rrbracket
 \end{aligned}$$

## Un problème d'inférence

$$\begin{aligned}
 \Gamma \vdash t : \sigma \text{ est résolu comme} \\
 \text{let } \Gamma \text{ in } \llbracket t : \sigma \rrbracket
 \end{aligned}$$

Cet algorithme est correct et complet. Il procède par réécriture des contraintes préservant le sens jusqu'à obtenir une forme résolue.

**Remarque** : la génération des contraintes (ci-dessus) pourrait être la spécification du typage.

let  $id : \forall \alpha. \alpha \rightarrow \alpha = \text{fun } x \rightarrow x$

let  $auto : (\forall \alpha. \alpha \rightarrow \alpha) \rightarrow \forall \alpha. \alpha \rightarrow \alpha = \text{fun } (f) \ f \ f$

let  $\_ = auto (id : \forall \alpha. \alpha \rightarrow \alpha)$

Notez la différence avec

let  $id : \forall (\alpha) \alpha \rightarrow \alpha = \text{fun } x \rightarrow x$

let  $auto : \exists \beta. (\forall (\alpha) \alpha \rightarrow \alpha) \rightarrow \beta = \text{fun } (f) \ f \ f$

let  $\_ = auto (id : \forall (\alpha) \alpha \rightarrow \alpha)$

# Inférence de types = vérification de formes

26(1)/50

---

## Le slogan, répété

L'inférence de type devine les monotypes mais vérifie les polytypes !

Peut-on l'énoncé formellement ? et mieux l'exploiter ?

## Les formes $\mathcal{S}$

- ▶ On étend les polytypes avec une constante  $\#$  pour représenter les monotypes.
- ▶ Une forme est un polytype étendu clos.
- ▶ Les formes sont considérées modulo l'égalité  $\# \rightarrow \# = \#$   
(ce qui revient à ignorer la structure des monotypes)

# Inférence de types = vérification de formes

26(2)/50

---

## Le slogan, répété

L'inférence de type devine les monotypes mais vérifie les polytypes !

Peut-on l'énoncé formellement ? et mieux l'exploiter ?

**Les formes  $\mathcal{S}$**  : polytypes **clos** étendus avec  $\#$  et l'équation  $\# = \# \rightarrow \#$ .

## Operations sur les formes

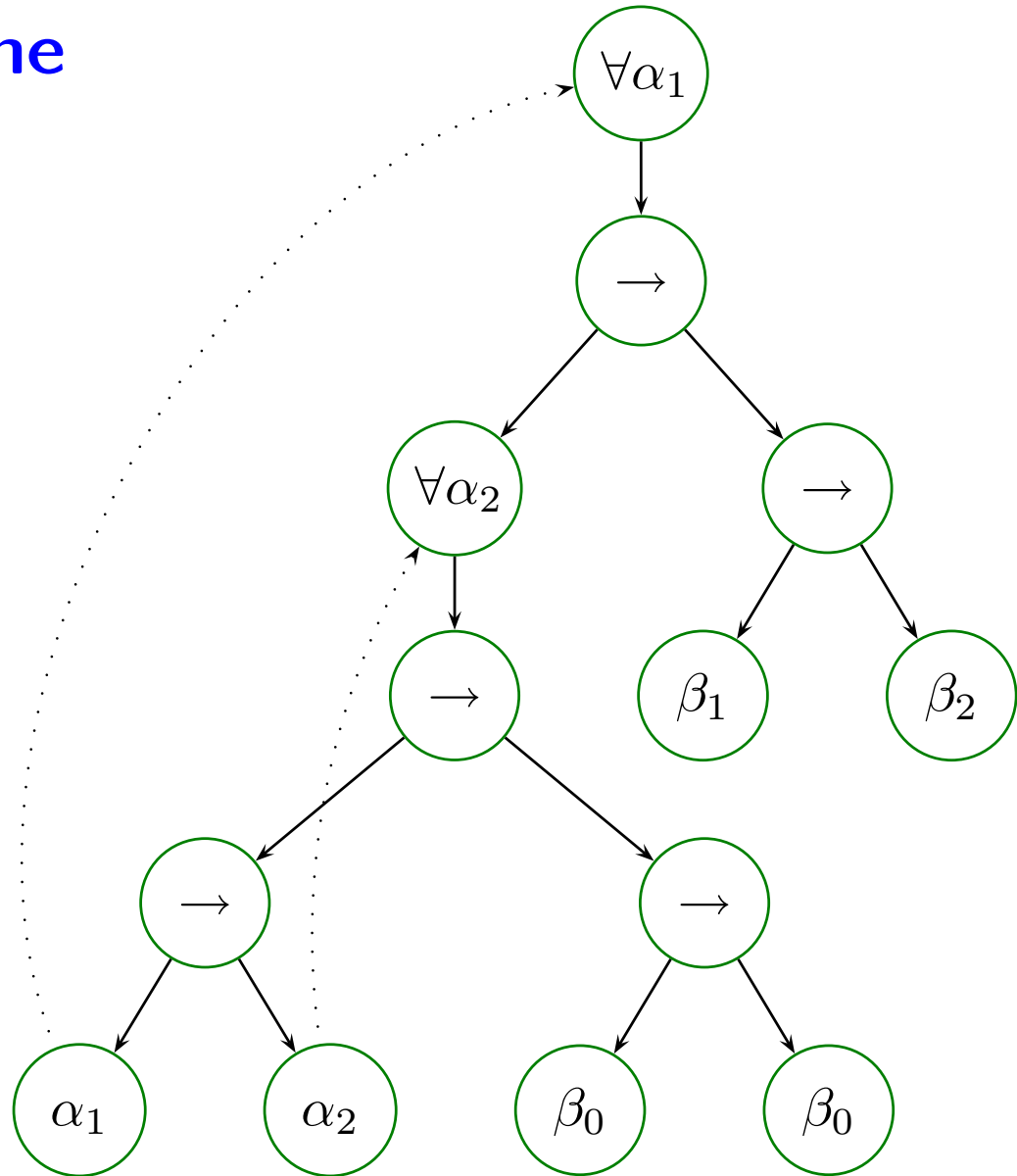
$[\sigma]$  retourne la forme de  $\sigma$ , i.e.  $\sigma[\#/\text{ftv}(\sigma)]$

$\mathcal{S}^b$  retire  $\mathcal{S}$  les quantificateurs en tête et remplace les variables libres par  $\#$ .

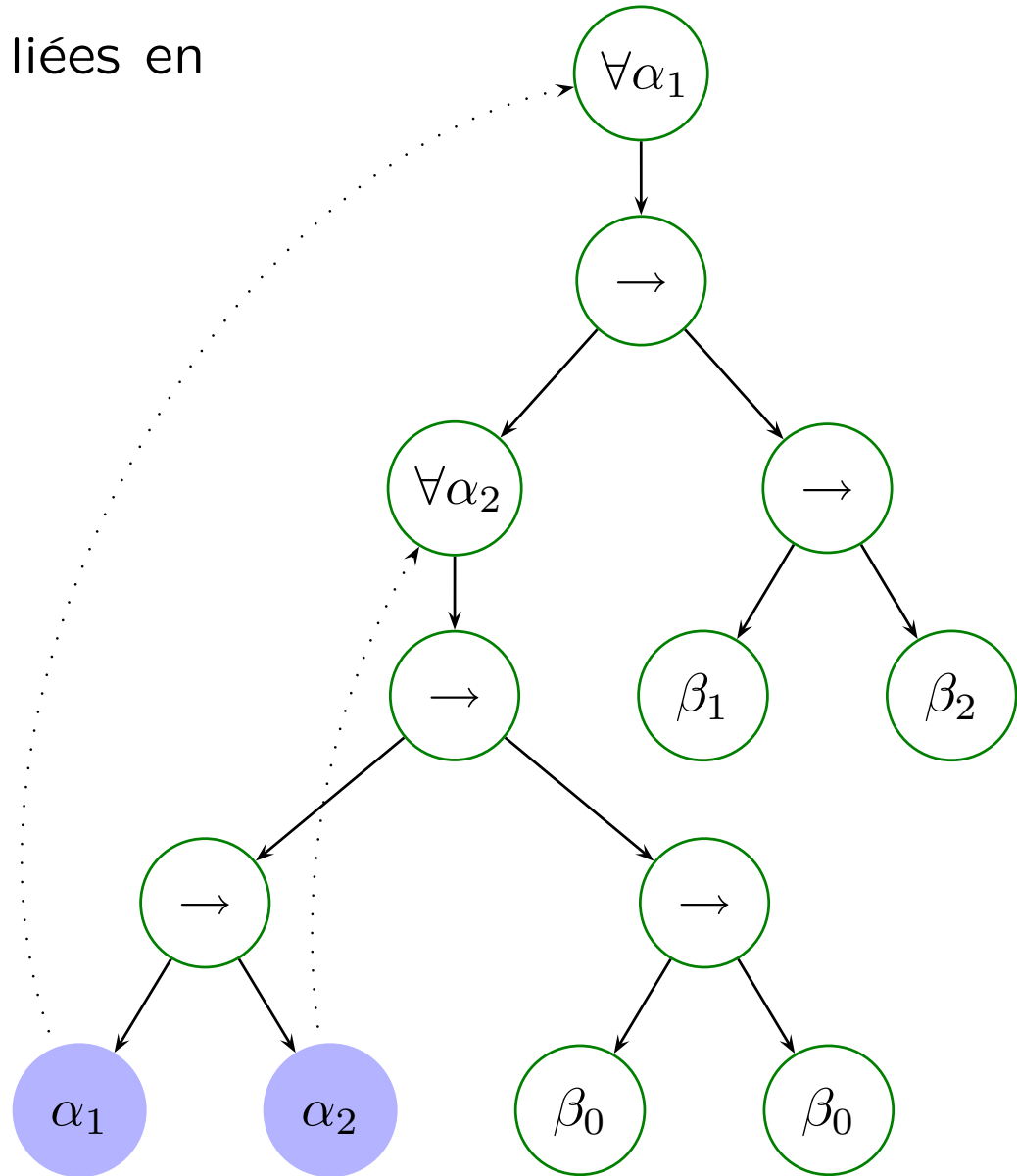
On écrit  $\mathcal{R}$  pour les formes ainsi dénudées.

$[\mathcal{S}]$  retourne l'annotation  $\exists \bar{\beta}. \mathcal{S}[\beta_i/\#_i]$ .

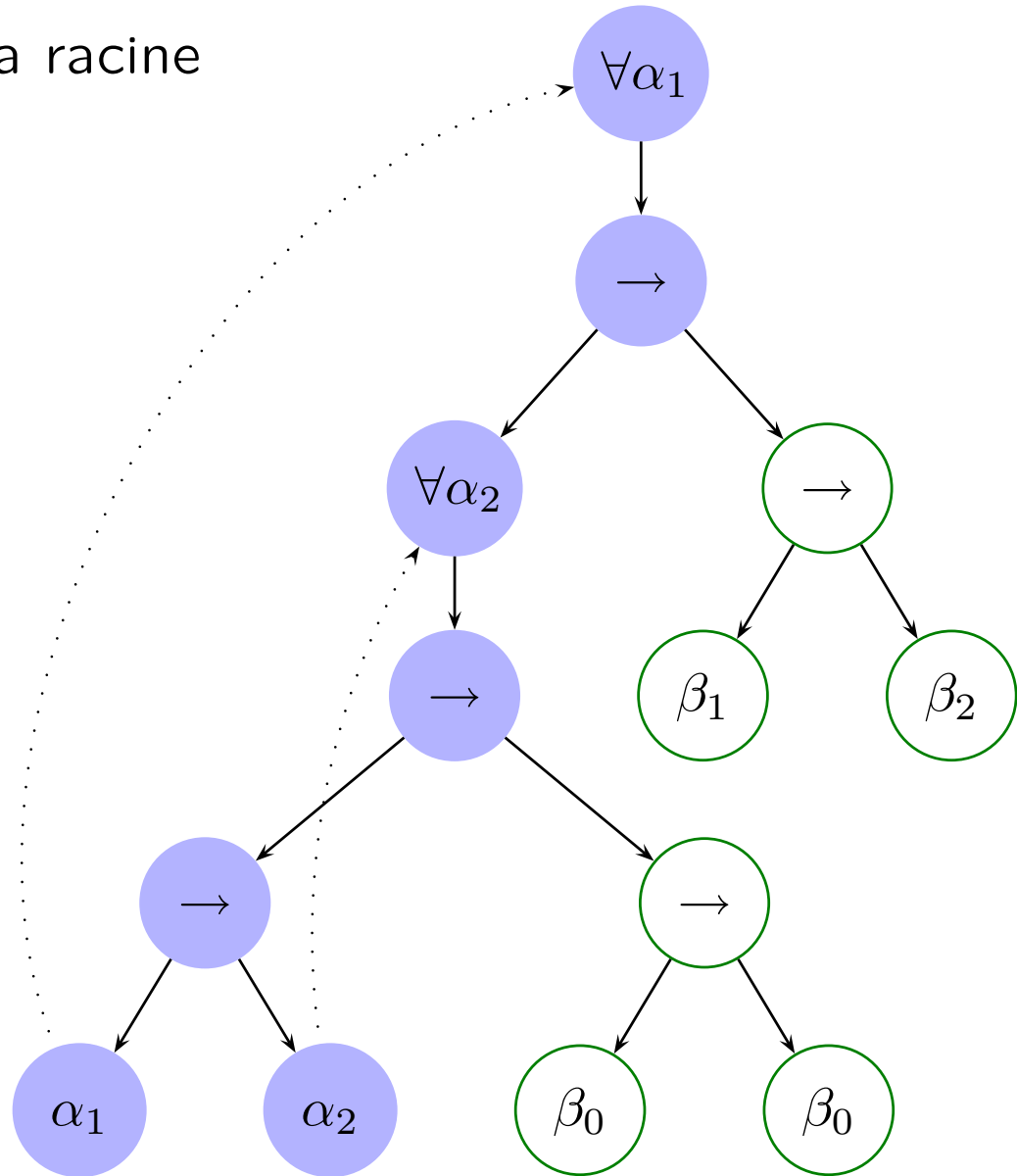
# Calcul de la forme



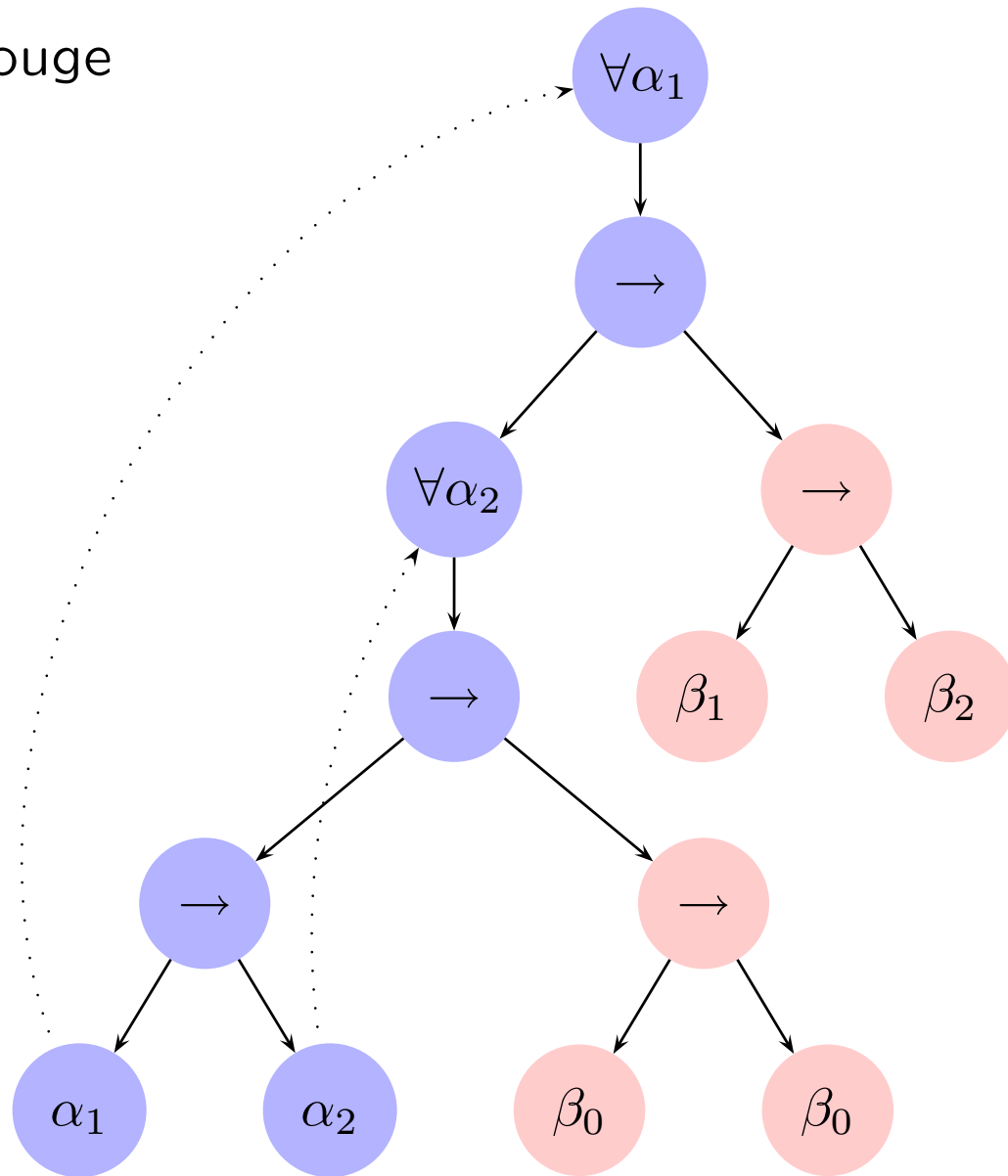
Marquer les variables liées en bleu



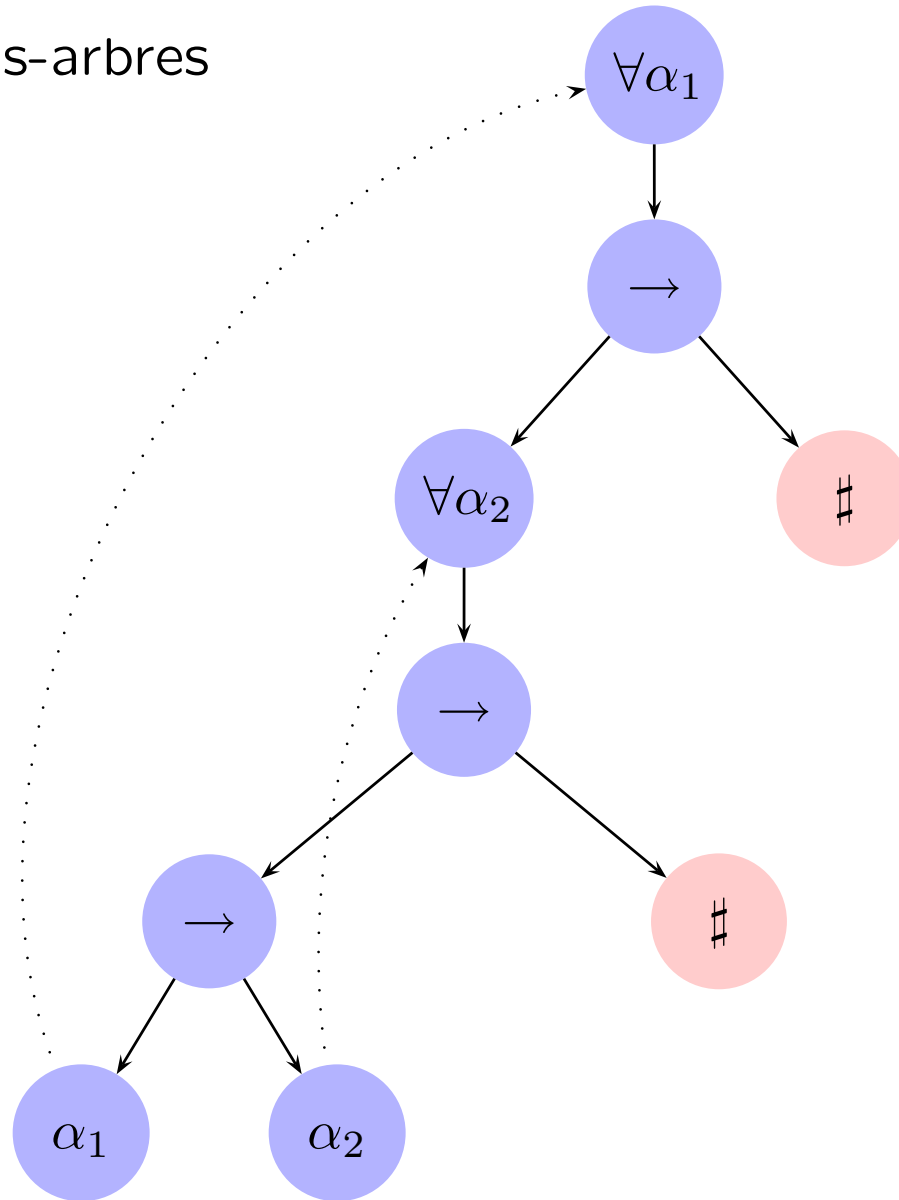
Étendre le bleu vers la racine



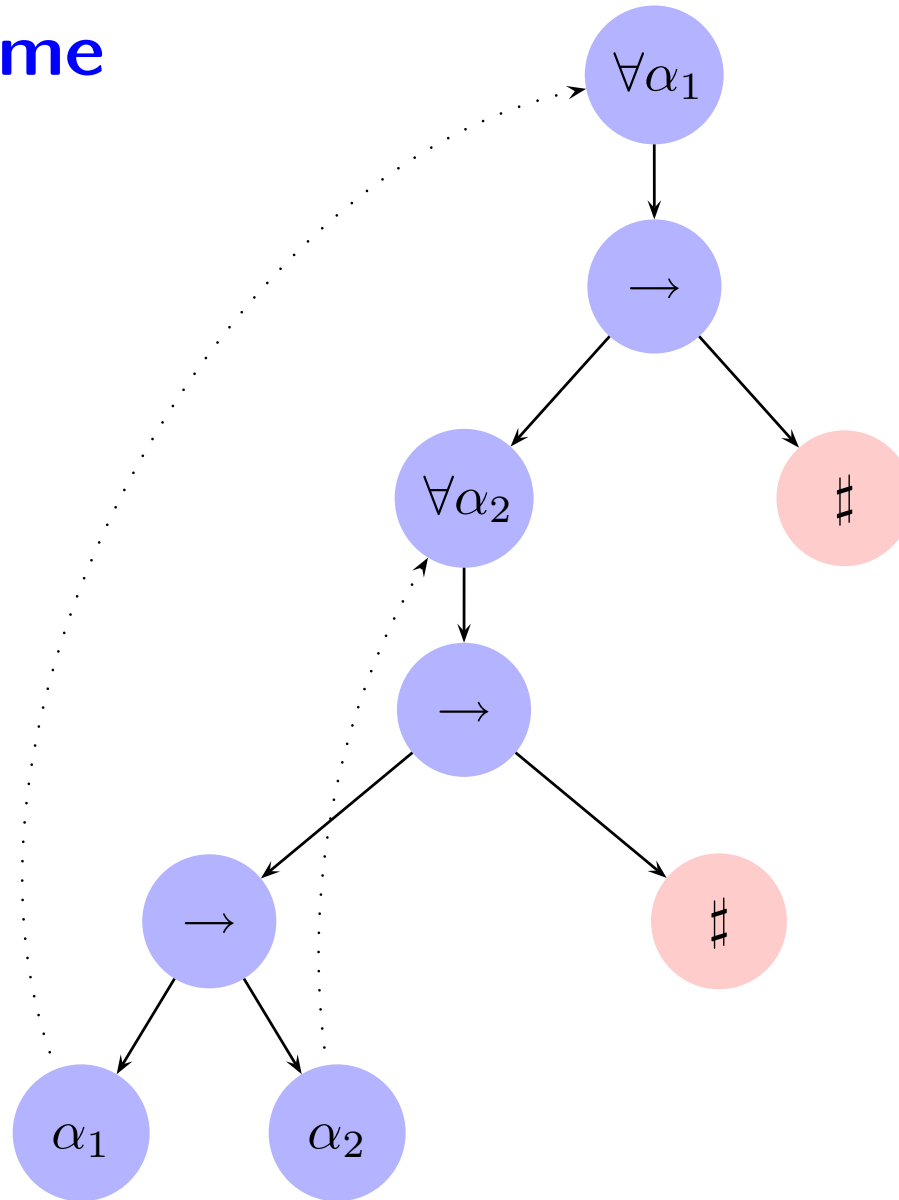
Marquer le reste en rouge



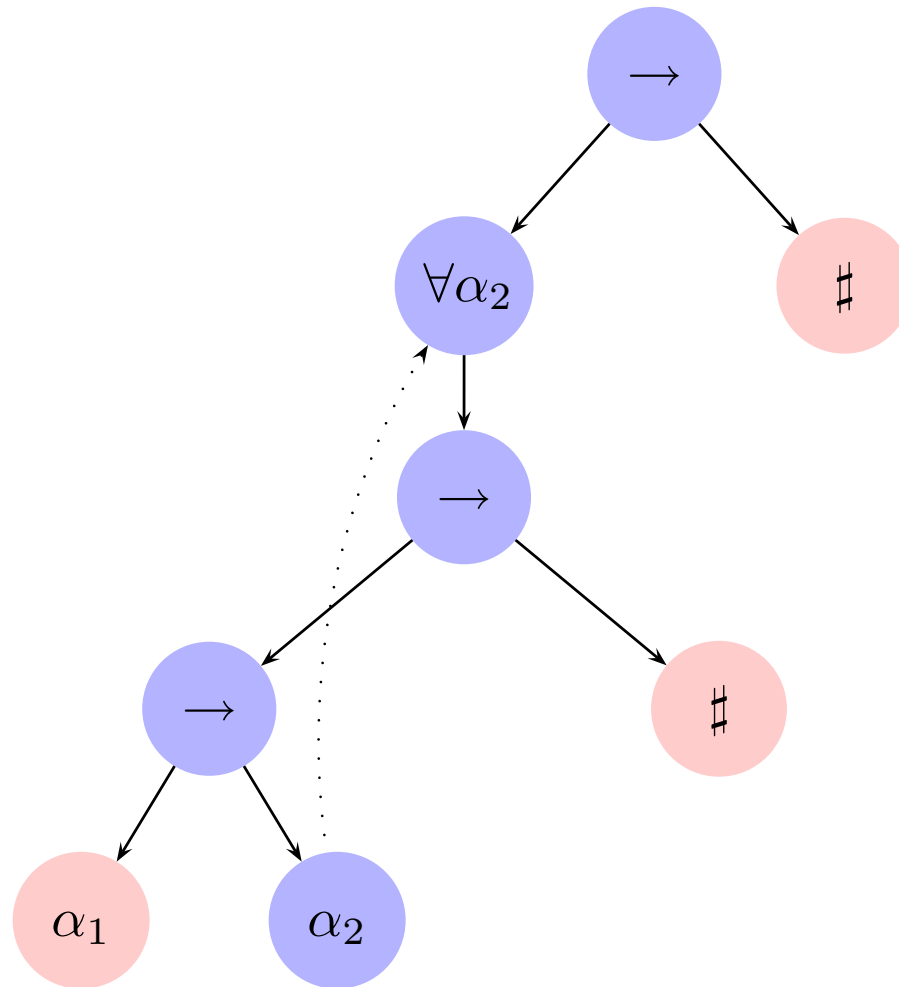
Remplacer les sous-arbres rouges par #



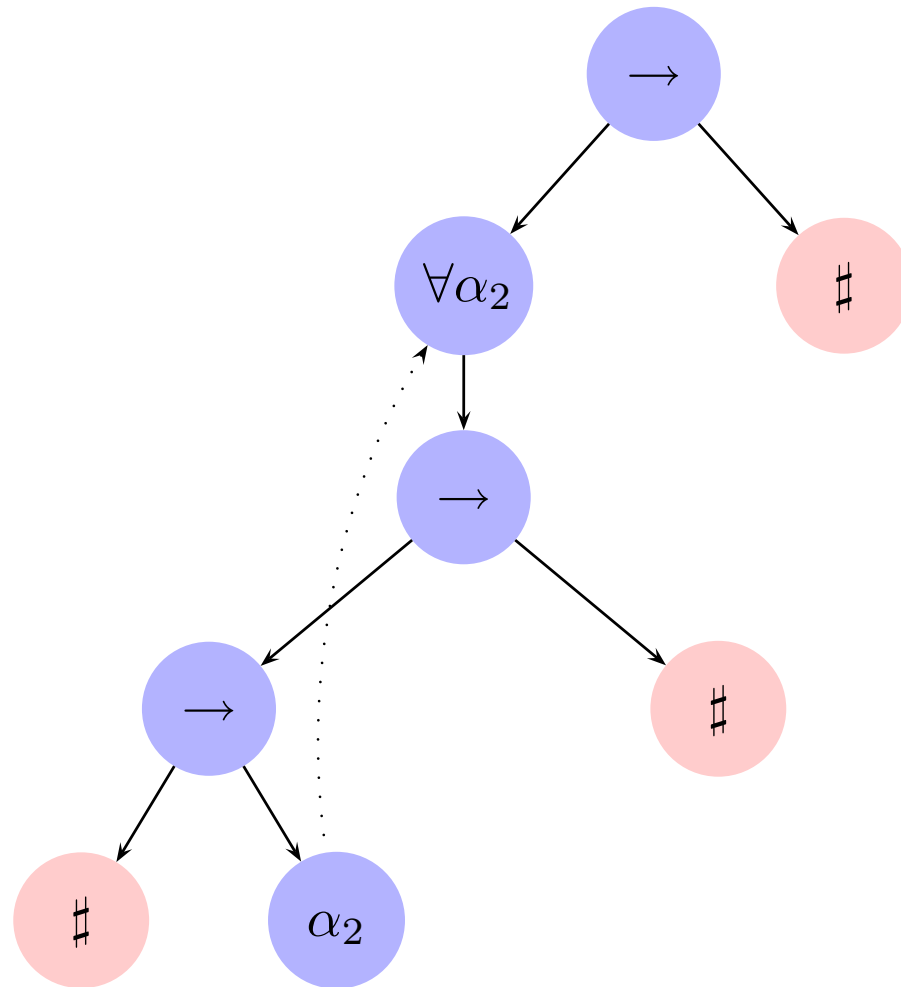
# Dénuder une forme



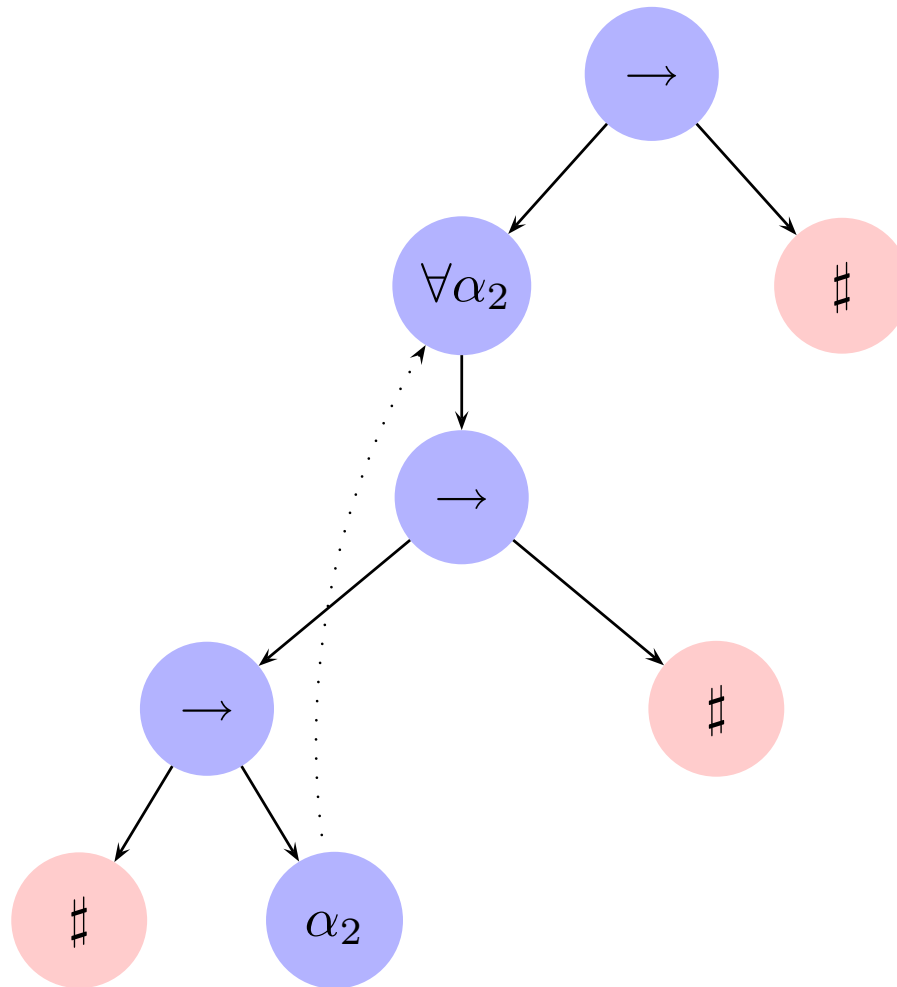
Retirer les quantificateurs en tête



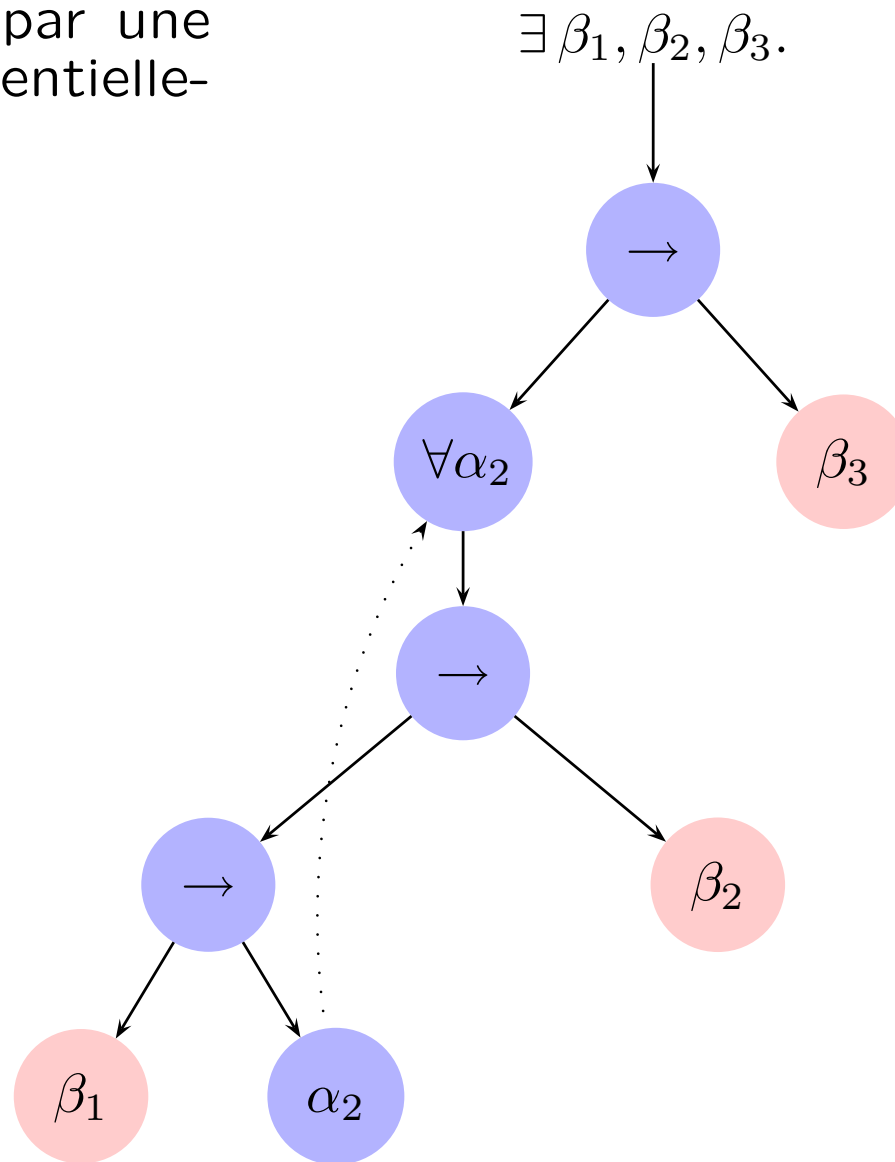
# Reformer



Construire une annotation à partir d'une forme



Remplacer chaque  $\#$  par une variable fraîche existentiellement quantifiée.



$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

Var-Rho

$$\frac{x : \sigma \in \Gamma \quad \sigma \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$\Gamma \vdash_{\downarrow} t : \mathcal{R}$

Gen

$$\frac{\Gamma \vdash t : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \sigma}$$

Var-Rho

$$\frac{x : \sigma \in \Gamma \quad \sigma \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$



Var-Rho

$$\frac{x : \sigma \in \Gamma \quad \sigma \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Var-Rho

$$\frac{x : \sigma \in \Gamma \quad \sigma \leq_p^{\parallel} \rho}{\Gamma \vdash x : \rho}$$

Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$\Gamma \vdash_{\downarrow} t : \mathcal{R}$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$\Gamma \vdash_{\downarrow} t : \mathcal{R}$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \sigma_2 \vdash t : \sigma_1}{\Gamma \vdash \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$\Gamma \vdash_{\downarrow} t : \mathcal{R}$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$\Gamma \vdash_{\downarrow} t : \mathcal{R}$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho \quad \Gamma \vdash t_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$\Gamma \vdash_{\downarrow} t : \mathcal{R}$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma[\bar{\tau}/\bar{\beta}]] \rightarrow \mathcal{R} \quad \Gamma \vdash t_2 : [\sigma[\bar{\tau}/\bar{\beta}]]^b}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \mathcal{R}}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma] \rightarrow \mathcal{S} \quad \Gamma \vdash t_2 : [\sigma]^b}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \mathcal{S}^b}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma] \rightarrow \mathcal{S} \quad \Gamma \vdash t_2 : [\sigma]^b}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \mathcal{S}^b}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : \sigma[\bar{\tau}/\bar{\beta}] \quad \Gamma, z : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash t_2 : \rho}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \rho}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma] \rightarrow \mathcal{S} \quad \Gamma \vdash t_2 : [\sigma]^b}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \mathcal{S}^b}$$

Let-Gen

$$\frac{\Gamma \vdash t_1 : [\sigma[\bar{\tau}/\bar{\beta}]]^b \quad \Gamma, z : [\langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}])]^b \vdash t_2 : \mathcal{R}}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \mathcal{R}}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma] \rightarrow \mathcal{S} \quad \Gamma \vdash t_2 : [\sigma]^b}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \mathcal{S}^b}$$

Let-Gen-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma]^b \quad \Gamma, z : [\sigma]^b \vdash t_2 : \mathcal{R}}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \mathcal{R}}$$

$$\Gamma \vdash_{\downarrow} t : \mathcal{R}$$

Var-Rho

$$\frac{x : \mathcal{R}' \in \Gamma \quad \mathcal{R}' \leq_p^{\parallel} \mathcal{R}}{\Gamma \vdash x : \mathcal{R}}$$

Fun

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash t : \mathcal{S}_1^b}{\Gamma \vdash \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma] \rightarrow \mathcal{S} \quad \Gamma \vdash t_2 : [\sigma]^b}{\Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma) : \mathcal{S}^b}$$

Let-Gen-Rho

$$\frac{\Gamma \vdash t_1 : [\sigma]^b \quad \Gamma, z : [\sigma]^b \vdash t_2 : \mathcal{R}}{\Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma) \text{ in } t_2 : \mathcal{R}}$$

## Les programmes bien typés sont bien formés

Si  $\Gamma \vdash t : \sigma$  alors  $[\Gamma] \vdash_{\downarrow} t : [\sigma]$ .

## Seule, la forme des annotations importe

Si  $\Gamma \vdash t : \sigma$  alors  $\Gamma \vdash \llbracket [t] \rrbracket : \sigma$ .

## Vérification des formes

Propage l'information de forme vers le bas (de la racine vers les feuilles)

## Inférence de forme

Propager l'information de forme vers le haut (des feuilles vers la racine)

**Remarque** : Il faut déplacer les annotations...

$$t ::= x \mid \text{fun } (z : \theta) t \mid t_1 t_2 \mid \text{let } z = t_1 \text{ in } t_2$$

$$\Gamma \vdash_{\uparrow} t : \mathcal{R}$$

Var-I

$$\frac{x : \mathcal{R} \in \Gamma}{\Gamma \vdash_{\uparrow} x : \mathcal{R}}$$

App-I

$$\frac{\Gamma \vdash_{\uparrow} t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \quad \Gamma \vdash_{\uparrow} t_2 : \mathcal{R}_2 \quad \mathcal{R}_2 \leq_p^{\parallel} \mathcal{S}_2^b}{\Gamma \vdash_{\uparrow} t_1 t_2 : \mathcal{S}_1^b}$$

Let-I

$$\frac{\Gamma \vdash_{\uparrow} t_1 : \mathcal{R}_1 \quad \Gamma, z : \mathcal{R}_1 \vdash_{\uparrow} t_2 : \mathcal{R}_2}{\Gamma \vdash_{\uparrow} \text{let } z = t_1 \text{ in } t_2 : \mathcal{R}_2}$$

Fun-I

$$\frac{\Gamma, z : [\sigma]^b \vdash_{\uparrow} t : \mathcal{R}}{\Gamma \vdash_{\uparrow} \text{fun } (z : \exists \bar{\beta}. \sigma) t : [\sigma] \rightarrow \mathcal{R}}$$

$$\Gamma \vdash_{\uparrow} t : \mathcal{R} \Rightarrow t'$$

Var-I

$$\frac{x : \mathcal{R} \in \Gamma}{\Gamma \vdash_{\uparrow} x : \mathcal{R} \Rightarrow x}$$

App-I

$$\frac{\Gamma \vdash_{\uparrow} t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \Rightarrow t'_1 \quad \Gamma \vdash_{\uparrow} t_2 : \mathcal{R}_2 \Rightarrow t'_2 \quad \mathcal{R}_2 \leq_p^{\parallel} \mathcal{S}_2^b}{\Gamma \vdash_{\uparrow} t_1 t_2 : \mathcal{S}_1^b \Rightarrow t'_1 ((t'_2 : [\mathcal{R}_2])) : [\mathcal{S}_2]}$$

Let-I

$$\frac{\Gamma \vdash_{\uparrow} t_1 : \mathcal{R}_1 \Rightarrow t'_1 \quad \Gamma, z : \mathcal{R}_1 \vdash_{\uparrow} t_2 : \mathcal{R}_2 \Rightarrow t'_2}{\Gamma \vdash_{\uparrow} \text{let } z = t_1 \text{ in } t_2 : \mathcal{R}_2 \Rightarrow \text{let } z = (t'_1 : [\mathcal{R}_1]) \text{ in } t'_2}$$

Fun-I

$$\frac{\Gamma, z : [\sigma]^b \vdash_{\uparrow} t : \mathcal{R} \Rightarrow t'}{\Gamma \vdash_{\uparrow} \text{fun } (z : \exists \bar{\beta}. \sigma) t : [\sigma] \rightarrow \mathcal{R} \Rightarrow \text{fun } (z) \text{ let } z = (z : \exists \bar{\beta}. \sigma) \text{ in } t'}$$

Definir

$$\Gamma \vdash_{\uparrow} t : \sigma$$

comme

$$\exists \mathcal{R}, ([\Gamma]^b \vdash_{\uparrow} t : \mathcal{R} \Rightarrow t' \wedge \mathcal{R} = \sigma^b \wedge \Gamma \vdash_{\downarrow} t : \sigma \Rightarrow t')$$

## Var-Inst

$$\frac{x : \forall \bar{\alpha}. \rho \in \Gamma}{\Gamma \vdash x : \rho[\bar{\tau}/\bar{\alpha}]}$$

## Fun-Gen

$$\frac{\Gamma, z : \sigma[\bar{\tau}/\bar{\beta}] \vdash t : \rho}{\Gamma \vdash \text{fun } (z : \exists \bar{\beta}. \sigma) t : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho}$$

## App

$$\frac{\Gamma \vdash t_1 : \sigma_2 \rightarrow \forall \bar{\alpha}. \rho_1 \quad \Gamma \vdash t_2 : \rho_2 \quad \langle \Gamma \rangle(\rho_2) \leq_p^{\parallel} \sigma_2}{\Gamma \vdash t_1 t_2 : \rho_1[\bar{\tau}/\bar{\alpha}]}$$

## Let-Gen

$$\frac{\Gamma \vdash t_1 : \rho_1 \quad \Gamma, z : \langle \Gamma \rangle(\rho_1) \vdash t_2 : \rho_2}{\Gamma \vdash \text{let } z = t_1 \text{ in } t_2 : \rho_2}$$

## Var-Inst

$$\frac{x : \forall \bar{\alpha}. \rho \in \Gamma}{\Gamma \vdash x : \rho[\bar{\tau}/\bar{\alpha}]}$$

## Fun-Gen

$$\frac{\Gamma, z : \sigma[\bar{\tau}/\bar{\beta}] \vdash t : \rho}{\Gamma \vdash \text{fun } (z : \exists \bar{\beta}. \sigma) t : \sigma[\bar{\tau}/\bar{\beta}] \rightarrow \rho}$$

## App

$$\frac{\Gamma \vdash t_1 : \sigma_2 \rightarrow \forall \bar{\alpha}. \rho_1 \quad \Gamma \vdash t_2 : \rho_2 \quad \langle \Gamma \rangle(\rho_2) \leq_p^{\parallel} \sigma_2}{\Gamma \vdash t_1 t_2 : \rho_1[\bar{\tau}/\bar{\alpha}]}$$

## Let-Gen

$$\frac{\Gamma \vdash t_1 : \rho_1 \quad \Gamma, z : \langle \Gamma \rangle(\rho_1) \vdash t_2 : \rho_2}{\Gamma \vdash \text{let } z = t_1 \text{ in } t_2 : \rho_2}$$

## Remarque

On retrouve le système de [Odersky and Läufer, 1996] à quelques variations près.

**Dans  $F_p^{\downarrow}$ , avec la vérification des formes seulement :**

$(\text{fun } (z) (\text{fun } (y) y : \sigma_{id} \rightarrow \sigma_{id}) : \alpha \rightarrow \sigma_{id} \rightarrow \sigma_{id})$

L'annotation en bleu (utile) doit être répétée en tête en rouge (inutile).

**Dans  $F_p^{\uparrow}$ , avec la propagation des formes seulement :**

$(\text{fun } (z) (\text{fun } (y : \sigma) y) : \alpha \rightarrow \sigma_{id} \rightarrow \sigma_{id})$

L'annotation externe en bleu (utile) doit être répétée en route (inutile).

**Peut-on mélanger les deux approches ?**

Var-C

$$\frac{}{\Gamma \vdash_! x : \mathcal{R}}$$

Var-I

$$\frac{x : \mathcal{R} \in \Gamma}{\Gamma \vdash_? x : \mathcal{R}}$$

App-C

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \quad \Gamma \vdash_! t_2 : \mathcal{S}_2^b \quad \mathcal{S}_1^b \leq_p^! \mathcal{R}_1}{\Gamma \vdash_! t_1 t_2 : \mathcal{R}_1}$$

App-I

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \quad \Gamma \vdash_! t_2 : \mathcal{S}_2^b}{\Gamma \vdash_? t_1 t_2 : \mathcal{S}_1^b}$$

Let-C

$$\frac{\Gamma \vdash_? t_1 : \mathcal{R}_1 \quad \Gamma, z : \mathcal{R}_1 \vdash_\epsilon t_2 : \mathcal{R}_2}{\Gamma \vdash_\epsilon \text{let } z = t_1 \text{ in } t_2 : \mathcal{R}_2}$$

Fun-Ce

$$\frac{\Gamma, z : [\sigma]^b \vdash_! t : \mathcal{S}_1^b \quad \mathcal{S}_2^b \leq_p^! [\sigma]^b}{\Gamma \vdash_! \text{fun } (z : \exists \beta. \sigma) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

Fun-Ie

$$\frac{\Gamma, z : [\sigma]^b \vdash_? t : \mathcal{R}}{\Gamma \vdash_? \text{fun } (z : \exists \bar{\beta}. \sigma) t : [\sigma] \rightarrow \mathcal{R}}$$

Fun-Ci

$$\frac{\Gamma, z : \mathcal{S}_2^b \vdash_! t : \mathcal{S}_1^b}{\Gamma \vdash_! \text{fun } (z) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

Fun-Ii

$$\frac{\Gamma, z : \# \vdash_? t : \mathcal{R}}{\Gamma \vdash_? \text{fun } (z) t : \# \rightarrow \mathcal{R}}$$

Var-C

Var-I

App-C

$$\frac{\begin{array}{l} \Gamma \vdash_{?} t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \\ \Gamma \vdash_{!} t_2 : \mathcal{S}_2^b \end{array} \quad \mathcal{S}_1^b \leq_p^{\parallel} \mathcal{R}_1}{\Gamma \vdash_{!} t_1 t_2 : \mathcal{R}_1}$$

App-I

$$\frac{\begin{array}{l} \Gamma \vdash_{?} t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \\ \Gamma \vdash_{!} t_2 : \mathcal{S}_2^b \end{array}}{\Gamma \vdash_{?} t_1 t_2 : \mathcal{S}_1^b}$$

Let-C

Fun-Ce

$$\frac{\begin{array}{l} \Gamma, z : [\sigma]^b \vdash_{!} t : \mathcal{S}_1^b \\ \mathcal{S}_2^b \leq_p^{\parallel} [\sigma]^b \end{array}}{\Gamma \vdash_{!} \text{fun } (z : \exists \beta. \sigma) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1}$$

Fun-Ie

Fun-Ci

Fun-Ii

$$\frac{\Gamma, z : \# \vdash_{?} t : \mathcal{R}}{\Gamma \vdash_{?} \text{fun } (z) t : \# \rightarrow \mathcal{R}}$$

Var-C

Var-I

App-C

$$\frac{\begin{array}{l} \Gamma \vdash_{?} t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \Rightarrow t'_1 \\ \Gamma \vdash_{!} t_2 : \mathcal{S}_2^b \Rightarrow t'_2 \quad \mathcal{S}_1^b \leq_p^{\parallel} \mathcal{R}_1 \end{array}}{\Gamma \vdash_{!} t_1 t_2 : \mathcal{R}_1 \Rightarrow t_1 (t_2 : [\mathcal{S}_2])}$$

App-I

$$\frac{\begin{array}{l} \Gamma \vdash_{?} t_1 : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \Rightarrow t'_1 \\ \Gamma \vdash_{!} t_2 : \mathcal{S}_2^b \Rightarrow t'_2 \end{array}}{\Gamma \vdash_{?} t_1 t_2 : \mathcal{S}_1^b \Rightarrow t_1 (t_2 : [\mathcal{S}_2])}$$

Let-C

Fun-Ce

$$\frac{\Gamma, z : [\sigma]^b \vdash_{!} t : \mathcal{S}_1^b \Rightarrow t' \quad \mathcal{S}_2^b \leq_p^{\parallel} [\sigma]^b}{\begin{array}{l} \Gamma \vdash_{!} \text{fun } (z : \exists \beta. \sigma) t : \mathcal{S}_2 \rightarrow \mathcal{S}_1 \\ \Rightarrow \text{fun } (z) \text{ let } z = (z : \exists \beta. \sigma) \text{ in } t' \end{array}}$$

Fun-Ie

Fun-Ci

Fun-Ii

$$\frac{\Gamma, z : \# \vdash_{?} t : \mathcal{R} \Rightarrow t'}{\Gamma \vdash_{?} \text{fun } (z) t : \# \rightarrow \mathcal{R} \Rightarrow \text{fun } (z) t'}$$

## Var-C

$$\frac{\sigma' \in \Gamma \quad \sigma' \leq_p^{\parallel} \rho}{\Gamma \vdash_! z : \rho}$$

## Var-I

$$\frac{z : \sigma \in \Gamma \quad \sigma \leq_p \rho}{\Gamma \vdash_? z : \rho}$$

## App-C

$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \rightarrow \sigma_1 \quad \Gamma \vdash_! t_2 : \sigma_2 \quad \langle \Gamma \rangle (\sigma_1) \leq_p^{\parallel} \rho_1}{\Gamma \vdash_! t_1 t_2 : \rho_1}$$

## App-I

$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \rightarrow \rho_1 \quad \Gamma \vdash_! t_2 : \sigma_2}{\Gamma \vdash_? t_1 t_2 : \rho_1}$$

## Let-C

$$\frac{\Gamma \vdash_? t_1 : \rho_1 \quad \Gamma, z : \langle \Gamma \rangle (\rho_1) \vdash_! t_2 : \sigma_2}{\Gamma \vdash_! \text{let } z = t_1 \text{ in } t_2 : \sigma_2}$$

## Let-I

$$\frac{\Gamma \vdash_? t_1 : \rho_1 \quad \Gamma, z : \langle \Gamma \rangle (\rho_1) \vdash_? t_2 : \rho_2}{\Gamma \vdash_? \text{let } z = t_1 \text{ in } t_2 : \rho_2}$$

## Fun-Ci

$$\frac{\Gamma, z : \sigma_2 \vdash_! t : \sigma_1}{\Gamma \vdash_! \text{fun } (z) t : \sigma_2 \rightarrow \sigma_1}$$

## Fun-Ie

$$\frac{\Gamma, z : \sigma[\tau/\bar{\beta}] \vdash_? t : \rho}{\Gamma \vdash_? \text{fun } (z : \exists \bar{\beta}. \sigma) t : \sigma[\tau/\bar{\beta}] \rightarrow \rho}$$

## Fun-Ce

$$\frac{\Gamma, z : \sigma[\bar{\tau}/\bar{\beta}] \vdash_! t : \sigma_1 \quad \sigma_2 \leq_p^{\parallel} \sigma[\bar{\tau}/\bar{\beta}] \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash_! \text{fun } (z : \exists \beta. \sigma) t : \forall \bar{\alpha}. \sigma_2 \rightarrow \sigma_1}$$

## Fun-Ii

$$\frac{\Gamma, z : \tau \vdash_? t : \rho}{\Gamma \vdash_? \text{fun } (z) t : \tau \rightarrow \rho}$$

		<p><b>App-C</b></p> $\frac{\Gamma \vdash? t_1 : \sigma_2 \rightarrow \sigma_1 \quad \Gamma \vdash! t_2 : \sigma_2 \quad \langle \Gamma \rangle (\sigma_1) \leq_p^{\parallel} \rho_1}{\Gamma \vdash! t_1 t_2 : \rho_1}$		<p><b>App-I</b></p> $\frac{\Gamma \vdash? t_1 : \sigma_2 \rightarrow \rho_1 \quad \Gamma \vdash! t_2 : \sigma_2}{\Gamma \vdash? t_1 t_2 : \rho_1}$	
<b>Var-C</b>	<b>Var-I</b>				
		<p><b>Let-C</b></p> $\frac{\Gamma \vdash? t_1 : \rho_1 \quad \Gamma, z : \langle \Gamma \rangle (\rho_1) \vdash! t_2 : \sigma_2}{\Gamma \vdash! \text{let } z = t_1 \text{ in } t_2 : \sigma_2}$	<b>Let-I</b>	<b>Fun-Ci</b>	<b>Fun-Ie</b>
<b>Fun-Ce</b>				<b>Fun-Ii</b>	
				$\frac{\Gamma, z : \tau \vdash? t : \rho}{\Gamma \vdash? \text{fun } (z) t : \tau \rightarrow \rho}$	
		$\frac{\Gamma, z : \sigma[\bar{\tau}/\bar{\beta}] \vdash! t : \sigma_1 \quad \sigma_2 \leq_p^{\parallel} \sigma[\bar{\tau}/\bar{\beta}] \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash! \text{fun } (z : \exists \beta. \sigma) t : \forall \bar{\alpha}. \sigma_2 \rightarrow \sigma_1}$			

		<b>App-C</b> $\frac{\Gamma \vdash? t_1 : \sigma_2 \rightarrow \sigma_1}{\Gamma \vdash! t_2 : \sigma_2 \quad \langle \Gamma \rangle (\sigma_1) \leq_p^{\parallel} \rho_1}$	<b>App-I</b> $\frac{\Gamma \vdash? t_1 : \sigma_2 \rightarrow \rho_1 \quad \Gamma \vdash! t_2 : \sigma_2}{\Gamma \vdash? t_1 t_2 : \rho_1}$
<b>Var-C</b>	<b>Var-I</b>		

**Let-C**

$$\frac{\Gamma \vdash? t_1 : \rho_1 \quad \Gamma, z : \langle \Gamma \rangle (\rho_1) \vdash! t_2 : \sigma_2}{\Gamma \vdash! \text{let } z = t_1 \text{ in } t_2 : \sigma_2}$$

**Let-I**

**Fun-Ci**

**Fun-Ie**

**Fun-Ce**

$$\frac{\Gamma, z : \sigma[\bar{\tau}/\bar{\beta}] \vdash! t : \sigma_1 \quad \sigma_2 \leq_p^{\parallel} \sigma[\bar{\tau}/\bar{\beta}] \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash! \text{fun } (z : \exists \beta. \sigma) t : \forall \bar{\alpha}. \sigma_2 \rightarrow \sigma_1}$$

**Fun-Ii**

$$\frac{\Gamma, z : \tau \vdash? t : \rho}{\Gamma \vdash? \text{fun } (z) t : \tau \rightarrow \rho}$$

Sucre syntaxique :

$$(t : \sigma) = (\text{fun } (z : \sigma) z) t$$

		<b>App-C</b>		<b>App-I</b>
		$\Gamma \vdash? t_1 : \sigma_2 \rightarrow \sigma_1$		$\Gamma \vdash? t_1 : \sigma_2 \rightarrow \rho_1$
<b>Var-C</b>	<b>Var-I</b>	$\Gamma \vdash! t_2 : \sigma_2$	$\langle \Gamma \rangle (\sigma_1) \leq_p^{\parallel} \rho_1$	$\Gamma \vdash! t_2 : \sigma_2$
		<hr/>		<hr/>
		$\Gamma \vdash! t_1 t_2 : \rho_1$		$\Gamma \vdash? t_1 t_2 : \rho_1$
		<b>Let-C</b>		
		$\Gamma \vdash? t_1 : \rho_1$		
		$\Gamma, z : \langle \Gamma \rangle (\rho_1) \vdash! t_2 : \sigma_2$		
		<hr/>		
		$\Gamma \vdash! \text{let } z = t_1 \text{ in } t_2 : \sigma_2$	<b>Let-I</b>	<b>Fun-Ci</b>
				<b>Fun-Ie</b>
		<b>Fun-Ce</b>		<b>Fun-Ii</b>
		$\Gamma, z : \sigma[\bar{\tau}/\bar{\beta}] \vdash! t : \sigma_1$	$\sigma_2 \leq_p^{\parallel} \sigma[\bar{\tau}/\bar{\beta}]$	$\bar{\alpha} \notin \text{ftv}(\Gamma)$
		<hr/>		
		$\Gamma \vdash! \text{fun } (z : \exists \beta. \sigma) t : \forall \bar{\alpha}. \sigma_2 \rightarrow \sigma_1$		$\Gamma, z : \tau \vdash? t : \rho$
				<hr/>
				$\Gamma \vdash? \text{fun } (z) t : \tau \rightarrow \rho$

Sucre syntaxique :

$$(t : \sigma) = (\text{fun } (z : \sigma) z) t$$

## Remarque

On retrouve le système de [Peyton-Jones and Shields, 2004] à quelques variations près.

## Séparation entre inférence et propagation des formes

- ▶ Composition de deux idées simples : simple et intuitif à comprendre.
- ▶ Chaque étape est présentée dans un formalisme adapté :
  - unification pour l'inférence dans le noyau.
  - algorithme pour la propagation.
- ▶ Variations possibles pour la propagation, mais le noyau est partagé.

## On retombe dans un cadre formel bien compris

- ▶  $F^\eta$  pour la correction.
- ▶ Contraintes de type pour l'inférence des monotypes.
- ▶ Types clos pour la propagation des formes.

## Séparation entre inférence et propagation des formes

On retombe dans un cadre formel bien compris

## Autres applications de ce cadre

- ▶ Variations faciles à envisager et étudier, car le cadre est simple.
- ▶ Propagation des formes plus agressive e.g. propagation bidirectionnelle colorée [Odersky et al., 2001] ?
- ▶ application de la technique à des rangs supérieurs  $F^\omega$  ?
- ▶ application de la technique au sous-typage (avec des contraintes de sous-typage de premier ordre) ?

## Séparation entre inférence et propagation des formes

On retombe dans un cadre formel bien compris

### Autres applications de ce cadre

- ▶ Variations faciles à envisager et étudier, car le cadre est simple.
- ▶ Propagation des formes plus agressive e.g. propagation bidirectionnelle colorée [Odersky et al., 2001] ?
- ▶ application de la technique à des rangs supérieurs  $F^\omega$  ?
- ▶ application de la technique au sous-typage (avec des contraintes de sous-typage de premier ordre) ?

**Mais expressivité limitée...**

Une approche plus ambitieuse

$ML^F$

Inférence dans le  
système F imprédicatif

(quelques idées)

## Poly-ML : une amélioration du polymorphisme encapsulé

► Grossièrement

- ▷ Les déclarations de types deviennent implicites.
- ▷ Les constructeurs sont remplacés par une annotation.
- ▷ Les destructeurs par `open`
- ▷ Le système retrouve le type à l'élimination par inférence.

► Exemple :

```
(fun (x :  $\forall \bar{\alpha}. \bar{\alpha} \rightarrow \bar{\alpha}$ ) (open x) (open x))  
  [ fun x x :  $\forall \bar{\alpha}. \bar{\alpha} \rightarrow \bar{\alpha}$  ]
```

► Le système garde trace des polytypes qui peuvent être ouverts

```
fun x (open x) (open x)
```

## Poly-ML : une amélioration du polymorphisme encapsulé

- ▶ Grossièrement
  - ▷ Les déclarations de types deviennent implicites.
  - ▷ Les constructeurs sont remplacés par une annotation.
  - ▷ Les destructeurs par `open`
  - ▷ Le système retrouve le type à l'élimination par inférence.

- ▶ Exemple :

```
(fun (x :  $\forall \bar{\alpha}. \bar{\alpha} \rightarrow \bar{\alpha}$ ) (open x) (open x))  
  [ fun x x :  $\forall \bar{\alpha}. \bar{\alpha} \rightarrow \bar{\alpha}$  ]
```

- ▶ Le système garde trace des polytypes qui peuvent être ouverts

```
fun x (open x) (open x)
```

- ▶ ML<sup>F</sup> reprend cette approche mais en plus laisse le `bleu` implicite.

## Poly-ML : une amélioration du polymorphisme encapsulé

- ▶ Grossièrement
  - ▷ Les déclarations de types deviennent implicites.
  - ▷ Les constructeurs sont remplacés par une annotation.
  - ▷ Les destructeurs par `open`
  - ▷ Le système retrouve le type à l'élimination par inférence.

- ▶ Exemple :

```
(fun (x :  $\forall \bar{\alpha}. \bar{\alpha} \rightarrow \bar{\alpha}$ ) (open x) (open x))  
  [ fun x x :  $\forall \bar{\alpha}. \bar{\alpha} \rightarrow \bar{\alpha}$  ]
```

- ▶ Le système garde trace des polytypes qui peuvent être ouverts

```
fun x (open x) (open x)
```

- ▶ ML<sup>F</sup> reprend cette approche mais en plus laisse le `bleu` implicite. Seul le `rouge` doit être indiqué explicitement.

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

let  $choice\ x\ y = \text{if } true \text{ then } x \text{ else } y : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$

let  $hard = choice\ id$

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

let  $hard = \text{choice } id \text{ else } y : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$

$id_\alpha \rightarrow id_\alpha \rightarrow id_\alpha$

$id_\alpha$

let  $hard = \text{choice } id$

$: \forall \alpha. (id_\alpha \rightarrow id_\alpha)$

Prendre une instance de  $id$

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

let  $choice = \lambda x \text{ else } y : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$   
 $id \rightarrow id \rightarrow id$

$id$

let  $hard = choice id$

$: \forall \alpha. (id_\alpha \rightarrow id_\alpha)$

$: (\forall \alpha. id_\alpha) \rightarrow (\forall \alpha. id_\alpha)$

Conserver  $id$  polymorphe

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

let  $choice\ x\ y = \text{if } true \text{ then } x \text{ else } y : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$

let  $hard = choice\ id$

$: \forall \alpha. (id_\alpha \rightarrow id_\alpha)$

$: (\forall \alpha. id_\alpha) \rightarrow (\forall \alpha. id_\alpha)$

} sont incomparables

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

let  $choice\ x\ y = \text{if } true \text{ then } x \text{ else } y : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $hard = choice\ id$

:  $\forall \alpha. (id_\alpha \rightarrow id_\alpha)$

:  $(\forall \alpha. id_\alpha) \rightarrow (\forall \alpha. id_\alpha)$

:  $\forall (\beta \geq id) \beta \rightarrow \beta$

$\beta \rightarrow \beta$

pour tout  $\beta$  qui est  
instance de  $id$

let  $id = \text{fun } (x) x : id$

~~let  $auto = \text{fun } (x) x x$~~

let  $auto = \text{fun } (x : id) x x : id \rightarrow id$

let  $auto' = id auto : id \rightarrow id$

let  $choice\ x\ y = \text{if } true \text{ then } x \text{ else } y : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$

où  $id = \forall \alpha. id_\alpha$

et  $id_\alpha = \alpha \rightarrow \alpha$

let  $hard = choice\ id$

:  $\forall \alpha. (id_\alpha \rightarrow id_\alpha)$

:  $(\forall \alpha. id_\alpha) \rightarrow (\forall \alpha. id_\alpha)$

:  $\forall (\beta \geq id) \beta \rightarrow \beta$

devrait s'écrire

$\forall (\beta \exists id, \beta' \geq id) \beta \rightarrow \beta'$

- 
- ▶  $\text{fun } (x : id) \ x \ x : \forall (\beta \exists id, \beta' \geq id) \ \beta \rightarrow \beta'$ 
    - ▷ Doit recevoir un argument de type  $id$ , puisque le polymorphisme peut avoir été utilisé.
    - ▷ Retourne une instance de  $id$ .

- 
- ▶  $\text{fun } (x : id) \ x \ x : \forall (\beta \equiv id, \beta' \geq id) \ \beta \rightarrow \beta'$ 
    - ▷  $\forall (\beta \equiv id)$  indique que le polymorphisme  $id$  de l'argument a pu être utilisé et ne permet pas de passer une instance de  $id$ .

- ▶  $\text{fun } (x : id) \ x \ x : \forall (\beta \exists id, \beta' \geq id) \ \beta \rightarrow \beta'$ 
  - ▷  $\forall (\beta \exists id)$  indique que le polymorphisme  $id$  de l'argument a pu être utilisé et ne permet pas de passer une instance de  $id$ .
  
- ▶  $\text{choice } id : \forall (\beta \geq id) \ \beta \rightarrow \beta$ 
  - ▷ peut recevoir un argument de type  $id$  ou une instance de  $id$  ;
  - ▷ le résultat est au moins aussi polymorphe que le type de l'argument.

- ▶  $\text{fun } (x : id) \ x \ x : \forall (\beta \exists id, \beta' \geq id) \ \beta \rightarrow \beta'$ 
  - ▷  $\forall (\beta \exists id)$  indique que le polymorphisme  $id$  de l'argument a pu être utilisé et ne permet pas de passer une instance de  $id$ .
  
- ▶  $\text{choice } id : \forall (\beta \geq id) \ \beta \rightarrow \beta$ 
  - ▷  $\forall (\beta \geq id)$  indique qu'une instance arbitraire de  $id$  peut être prise.
  - ▷ L'instance pourra être choisie plus tard, si besoin.

- ▶  $\text{fun } (x : id) x x : \forall (\beta \exists id, \beta' \geq id) \beta \rightarrow \beta'$ 
  - ▷  $\forall (\beta \exists id)$  indique que le polymorphisme  $id$  de l'argument a pu être utilisé et ne permet pas de passer une instance de  $id$ .
- ▶  $\text{choice } id : \forall (\beta \geq id) \beta \rightarrow \beta$ 
  - ▷  $\forall (\beta \geq id)$  indique qu'une instance arbitraire de  $id$  peut être prise.
- ▶  $\text{fun } (x) x : \forall (\beta) \beta \rightarrow \beta \leq \forall (\beta \geq id) \beta \rightarrow \beta \leq \forall (\beta \exists id) \beta \rightarrow \beta$ 
  - ▷  $\forall (\beta \exists id)$  conserve le partage des deux «occurrences» de  $id$  sous le nom  $\beta$ , ce qui empêchera d'utiliser implicitement le polymorphisme de  $id$ .

- ▶  $\text{fun } (x : id) x x : \forall (\beta \exists id, \beta' \geq id) \beta \rightarrow \beta'$ 
  - ▷  $\forall (\beta \exists id)$  indique que le polymorphisme  $id$  de l'argument a pu être utilisé et ne permet pas de passer une instance de  $id$ .
- ▶  $\text{choice } id : \forall (\beta \geq id) \beta \rightarrow \beta$ 
  - ▷  $\forall (\beta \geq id)$  indique qu'une instance arbitraire de  $id$  peut être prise.
- ▶  $\text{fun } (x) x : \forall (\beta) \beta \rightarrow \beta \leq \forall (\beta \geq id) \beta \rightarrow \beta \leq \forall (\beta \exists id) \beta \rightarrow \beta$ 
  - ▷  $\forall (\beta \exists id)$  conserve le partage
- ▶  $\forall (\alpha_1 \exists \sigma_1) \forall (\alpha_2 \geq \forall (\alpha \geq \sigma) \sigma_2) \dots \tau$ 
  - ▷ mémorise l'histoire de l'inférence,
  - ▷ *i.e.* le partage et tous les autres chemins qui auraient pu être obtenus en choisissant différentes instantiations.

Var

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

Fun

$$\frac{\Gamma, x : \tau_0 \vdash a : \tau}{\Gamma \vdash \text{fun } (x) a : \tau_0 \rightarrow \tau}$$

App

$$\frac{\begin{array}{l} \Gamma \vdash a_1 : \tau_2 \rightarrow \tau_1 \\ \Gamma \vdash a_2 : \tau_2 \end{array}}{\Gamma \vdash a_1 a_2 : \tau_1}$$

Let

$$\frac{\begin{array}{l} \Gamma \vdash a_1 : \sigma \\ \Gamma, x : \sigma \vdash a_2 : \tau \end{array}}{\Gamma \vdash \text{let } x = a_1 = a \text{ in } a_2 : \tau}$$

Gen

$$\frac{\begin{array}{l} \alpha \notin \text{ftv}(\Gamma) \\ \Gamma \vdash a : \sigma' \end{array}}{\Gamma \vdash a : \forall \alpha . \sigma'}$$

Inst

$$\frac{\Gamma \vdash a : \sigma \quad \sigma \leq \sigma'}{\Gamma \vdash a : \sigma'}$$

$$\begin{array}{c} \text{Var} \\ \frac{x : \sigma \in \Gamma}{\forall (Q) \Gamma \vdash x : \sigma} \end{array} \quad \begin{array}{c} \text{Fun} \\ \frac{\forall (Q) \Gamma, x : \tau_0 \vdash a : \tau}{\forall (Q) \Gamma \vdash \text{fun } (x) a : \tau_0 \rightarrow \tau} \end{array} \quad \begin{array}{c} \text{App} \\ \frac{\forall (Q) \Gamma \vdash a_1 : \tau_2 \rightarrow \tau_1 \quad \forall (Q) \Gamma \vdash a_2 : \tau_2}{\forall (Q) \Gamma \vdash a_1 a_2 : \tau_1} \end{array}$$
$$\begin{array}{c} \text{Let} \\ \frac{\forall (Q) \Gamma \vdash a_1 : \sigma \quad \forall (Q) \Gamma, x : \sigma \vdash a_2 : \tau}{\forall (Q) \Gamma \vdash \text{let } x = a_1 = a \text{ in } a_2 : \tau} \end{array} \quad \begin{array}{c} \text{Gen} \\ \frac{\alpha \notin \text{ftv}(\Gamma) \quad \forall (Q, \alpha \diamond \sigma) \Gamma \vdash a : \sigma'}{\forall (Q) \Gamma \vdash a : \forall \alpha \diamond \sigma. \sigma'} \end{array}$$
$$\begin{array}{c} \text{Inst} \\ \frac{\forall (Q) \Gamma \vdash a : \sigma \quad \forall (Q) \sigma \leq \sigma'}{\forall (Q) \Gamma \vdash a : \sigma'} \end{array}$$

$$\begin{array}{c}
 \text{Var} \\
 \frac{x : \sigma \in \Gamma}{\forall (Q) \Gamma \vdash x : \sigma}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{Fun} \\
 \frac{\forall (Q) \Gamma, x : \tau_0 \vdash a : \tau}{\forall (Q) \Gamma \vdash \text{fun } (x) a : \tau_0 \rightarrow \tau}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{App} \\
 \frac{\forall (Q) \Gamma \vdash a_1 : \tau_2 \rightarrow \tau_1 \quad \forall (Q) \Gamma \vdash a_2 : \tau_2}{\forall (Q) \Gamma \vdash a_1 a_2 : \tau_1}
 \end{array}$$
  

$$\begin{array}{c}
 \text{Let} \\
 \frac{\forall (Q) \Gamma \vdash a_1 : \sigma \quad \forall (Q) \Gamma, x : \sigma \vdash a_2 : \tau}{\forall (Q) \Gamma \vdash \text{let } x = a_1 = a \text{ in } a_2 : \tau}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{Gen} \\
 \frac{\alpha \notin \text{ftv}(\Gamma) \quad \forall (Q, \alpha \diamond \sigma) \Gamma \vdash a : \sigma'}{\forall (Q) \Gamma \vdash a : \forall \alpha \diamond \sigma. \sigma'}
 \end{array}$$
  

$$\begin{array}{c}
 \text{Inst} \\
 \frac{\forall (Q) \Gamma \vdash a : \sigma \quad \forall (Q) \sigma \leq \sigma'}{\forall (Q) \Gamma \vdash a : \sigma'}
 \end{array}$$

$Q$  liste les variables de types libres avec leur bornes.

$$\tau = \alpha \mid \tau_1 \rightarrow \tau_1$$

Monotypes

$$\sigma = \tau \mid \perp \mid \forall (\alpha \geq \sigma_1) \sigma_2 \mid \forall (\alpha \exists \sigma_1) \sigma_2$$

Schémas de types

Signifie «tous les types»

- ▶  $\forall (\alpha) \sigma$  est une abbréviatiion pour  $\forall (\alpha \geq \perp) \sigma$ .
- ▶ Remarque : les types polymorphes  $\sigma$  ne peuvent pas apparaître dans les monotypes ; ils doivent être liés à des variables auxiliaires par  $\beta \exists \sigma$  et remplacés par  $\beta$ , mais ce n'est pas restrictif :

e.g., type  $\forall \alpha. (\forall \alpha'. \tau \rightarrow \alpha) \rightarrow \alpha$

s'écrit :

$$\forall (\alpha) \forall (\beta \exists \forall (\alpha') \tau \rightarrow \alpha) \beta \rightarrow \alpha$$

## Examples

$$id \triangleq \forall \alpha. \alpha \rightarrow \alpha$$

$$id_\alpha \triangleq \alpha \rightarrow \alpha$$

$$\perp \leq \sigma$$

$$\forall (\alpha) \alpha \rightarrow \alpha \leq \alpha \rightarrow \alpha$$

$$\forall (\alpha) \alpha \rightarrow \alpha \leq \forall (\alpha, \alpha') (\alpha \rightarrow \alpha') \rightarrow (\alpha \rightarrow \alpha')$$

$$\forall (\beta \geq id) \beta \rightarrow \beta \leq \forall (\beta \exists id) \beta \rightarrow \beta$$

$$\forall (\beta \geq id) \beta \rightarrow \beta \leq \forall (\alpha) \forall (\beta \geq id_\alpha) \beta \rightarrow \beta$$

$$\leq \forall (\alpha) id_\alpha \rightarrow id_\alpha$$

$$\forall (\beta \geq \forall (\alpha) \tau') \tau \rightarrow \beta \leq \forall (\alpha) (\tau \rightarrow \tau') \quad \text{if } \alpha \notin \text{ftv}(\tau)$$

$$\approx \tau \rightarrow \forall (\alpha) \tau'$$

$$\leq \forall (\alpha) \forall (\beta \exists \tau') \tau \rightarrow \beta \leq$$

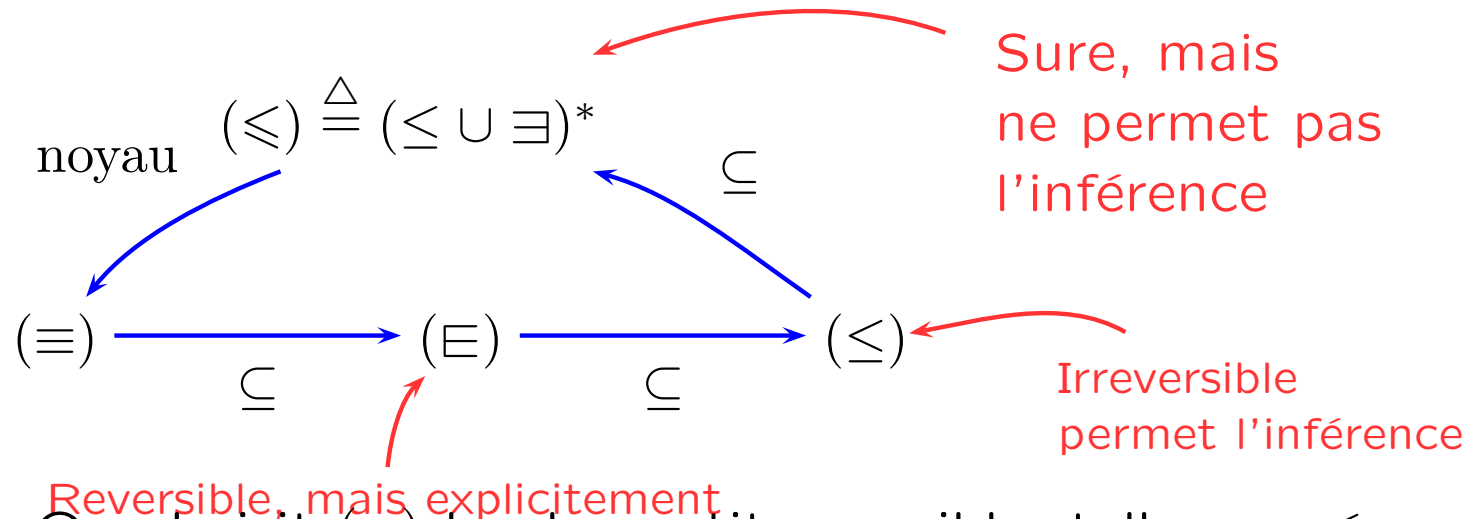
Un préfixe  $Q$  est une séquence de  $\alpha_1 \diamond_1 \sigma_1, \dots, \alpha_n \diamond_n \sigma_n$  où  $\alpha \diamond \sigma$  stands for either  $\alpha \geq \sigma$  ou  $\alpha \exists \sigma$ .

- ▶ Instance relation is defined under prefixes

$$\forall (Q) \sigma \leq \sigma'$$

## Par des exemples

- ▶  $\forall (\alpha \geq \sigma) \sigma \leq \alpha$  ( $\alpha$  généralise  $\sigma$ )
- ▶  ~~$\forall (\alpha \geq \sigma) \alpha \leq \sigma$~~  (sans surprise)
- ▶  $\forall (\alpha \exists \sigma) \sigma \leq \alpha$  ( $\alpha$  abstrait  $\sigma$ )
- ▶  ~~$\forall (\alpha \exists \sigma) \alpha \leq \sigma$~~ 
  - ▷ serait sûr, mais casserait l'inférence.
  - ▷ les noms abstrait ne peuvent pas être réifiés (implicitement).



$\leq$  est donnée. On choisit  $(\exists)$  la plus petite possible, telle que  $\leq$  permette l'inférence.

On récupère la partie manquante  $\exists$  de  $\leq$  par des fonctions de coercions explicites (primitives).

$$(- : \sigma) : \forall (\alpha \exists \sigma, \alpha' \exists \sigma) \alpha \rightarrow \alpha'$$

Sucre syntaxique :

$$\text{fun } (x : \sigma) a \triangleq \text{fun } (x) \text{ let } x = (x : \sigma) \text{ in } a$$

## Types existentiels

```
type ice = ['b] [ 'ab = ['a] ( 'a -> unit ) * 'a -> 'b ] 'ab -> 'b ;;
let pack_ice x =
  (fun ( f : [ 'a ] ( 'a -> unit ) * 'a -> 'b ) -> f x : ice );;

let packed_ice_int = pack_ice ( print_int , 1 );;
let packed_ice_string = pack_ice ( print_string , " -hi!" );;
let packed_ice = [ packed_ice_int ; packed_ice_string ];;

let unpack x f = x f;;
let melt fx = (fst fx) (snd fx);;

packed_ice_int melt;;
let melted_ice = listiter (fun packed -> packed melt) packed_ice;;
```

## Importance des types d'ordre supérieur

### Deux solutions pour l'intégration

- ▶ Très simple mais limitée
- ▶ Expressive et prometteuse mais (encore) compliquée.
- ▶ **Role clé de la relation d'instantiation.**
- ▶ *Autre approche également possible : inférence locale.*

### Types d'ordre supérieure indispensables

- ▶ Avec ou sans inférence.
- ▶ En pratique, une forme d'inférence semble indispensable.
- ▶ La relation d'instantiation de  $ML^F$  est intéressante même sans inférence (elle permet de réduire l'ordre).

# Un peu de lecture...

- [Garrigue and Rémy, 1997] Garrigue, J. and Rémy, D. (1997). Extending ML with semi-explicit higher-order polymorphism. In Ito, T. and Abadi, M., editors, *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 20–46. Springer-Verlag.
- [Läufer and Odersky, 1994] Läufer, K. and Odersky, M. (1994). Polymorphic type inference and abstract data types. *ACM Transactions on Programming Languages and Systems*, 16(5) :1411–1430.
- [Le Botlan and Rémy, 2003] Le Botlan, D. and Rémy, D. (2003). MLF : Raising ML to the power of system-F. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 27–38.
- [Mitchell, 1988] Mitchell, J. C. (1988). Polymorphic type inference and containment. *Information and Computation*, 76 :211–249.
- [Odersky and Läufer, 1996] Odersky, M. and Läufer, K. (1996). Putting type annotations to work. In *ACM Symposium on Principles of Programming*, pages 54–67, St. Petersburg, Florida. ACM Press.
- [Odersky et al., 2001] Odersky, M., Zenger, C., and Zenger, M. (2001). Colored local type inference. *ACM SIGPLAN Notices*, 36(3) :41–53.
- [Peyton-Jones and Shields, 2004] Peyton-Jones, S. and Shields, M. (2004). Practical type inference for arbitrary-rank types. Available electronically.

- [Pfenning, 1988] Pfenning, F. (1988). Partial polymorphic type inference and higher-order unification. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 153–163. ACM Press.
- [Pierce, 2004] Pierce, B. C., editor (2004). *Advanced Topics in Types and Programming Languages*. MIT Press. To appear.
- [Pierce and Turner, 1998] Pierce, B. C. and Turner, D. N. (1998). Local type inference. Full version in *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(1), January 2000, pp. 1–44.
- [Rémy, 1994] Rémy, D. (1994). Programming objects with ML-ART : An extension to ML with abstract and record types. In Hagiya, M. and Mitchell, J. C., editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 321–346. Springer-Verlag.
- [Wells, 1994] Wells, J. B. (1994). Typability and type checking in the second-order  $\lambda$ -calculus are equivalent and undecidable. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 176–185.



## Algorithme

$$(\varphi \wedge \Gamma \vdash t : \rho) \rightsquigarrow \varphi'$$

Étant donné une substitution partielle  $\varphi$ , et un problème d'inférence  $\Gamma \vdash t : \rho$ , l'algorithme calcule une meilleure solution  $\varphi'$ .

C'est-à-dire,  $\varphi'$  est une substitution la plus générale qui soit une instance de  $\varphi$  et satisfasse le problème de typage (*i.e.* telle que  $\varphi'(\Gamma) \vdash t : \varphi'(\rho)$ ). *Plus générale* signifie que toutes les autres solutions sont de la forme  $\varphi'' \circ \varphi'$ .

En fait, on doit garder trace des variables fraîches et écrire  $\exists W. (\varphi \wedge \Gamma \vdash t : \rho) \rightsquigarrow \exists W'. \varphi'$ . où  $W$  est l'ensemble des variables introduites par  $\varphi$ .

## Algorithme

$$(\varphi \wedge \Gamma \vdash t : \rho) \rightsquigarrow \varphi'$$

$$\frac{\varphi(\Gamma(z)) \leq \varphi(\sigma) \rightsquigarrow \varphi'}{\varphi \wedge \Gamma \vdash z : \sigma \rightsquigarrow \varphi' \circ \varphi}$$

$$\frac{\varphi \wedge \Gamma \vdash t_1 : \sigma_2 \rightarrow \rho_1 \rightsquigarrow \varphi_1 \quad \varphi_1 \wedge \Gamma \vdash t_2 : \sigma_2 \rightsquigarrow \varphi_2 \quad \bar{\beta} \text{ fresh}}{\varphi \wedge \Gamma \vdash t_1 (t_2 : \exists \bar{\beta}. \sigma_2) : \rho_1 \rightsquigarrow \varphi_2}$$

$$\frac{\varphi \wedge \Gamma, z : \sigma \vdash t : \rho \rightsquigarrow \varphi'}{\varphi \wedge \Gamma \vdash \text{fun } (z) t : \sigma \rightarrow \rho \rightsquigarrow \varphi'}$$

$$\frac{\varphi(\tau) = \beta' \rightarrow \beta \rightsquigarrow \varphi' \quad \beta\beta' \text{ fresh} \quad \varphi' \wedge \Gamma, z : \beta' \vdash t : \beta \rightsquigarrow \varphi''}{\varphi \wedge \Gamma \vdash \text{fun } (z) t : \tau \rightsquigarrow \varphi''}$$

$$\frac{\varphi \wedge \Gamma \vdash t_1 : \rho_1 \rightsquigarrow \varphi_1 \quad \bar{\beta} \text{ fresh} \quad \varphi_1 \wedge \Gamma, z : \langle \Gamma \rangle(\varphi_1(\sigma_1)) \vdash t_2 : \forall \bar{\alpha}. \rho_2 \rightsquigarrow \varphi_2}{\varphi_2 \wedge \Gamma \vdash \text{let } z = (t_1 : \exists \bar{\beta}. \sigma_1) \text{ in } t_2 : \rho_1}$$

## Algorithme

$$(\varphi \wedge \Gamma \vdash t : \rho) \rightsquigarrow \varphi'$$

It can be (advantageously) seen as a type constraints with a particular eager resolution strategy.