Contents

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 connait 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 phychologique ! (Mais des retombées indirectes.)

ML

- λ-calculus simplement typé avec une construction de liaison

Synthèse des types très simple...

- *Intuitivement*, comme le λ-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.

ML a pris de l'âge...

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 !

Parenthèse : un peu de vocabulaire

Types simples ou types de premier ordre :

Ce sont les types sans quantificateurs : int, int $\to$ int, $\alpha \to \alpha$

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 : $\forall\, \alpha.(\forall\, \beta.\, \tau \to \alpha) \to \alpha$

Types d'ordre supérieur ou types du système F^{ω}.

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

Polymorphisme de première classe

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

Polymorphisme prédicatif/imprédicatif

Les types polymorphmes peuvent être instantiés par des types **moins/aussi** polymorphormes. 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)

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

mais

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

(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.

Parenthèse : un peu de vocabulaire

Un besoin intrinsèque

Besoin de méthodes polymorphes

Besoin de types existentiels de première classe...

- Pour protégerCacher la représentaion
  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 $\exists\beta.(\beta \to \tau) \times \beta$ où le type de l'argument β est caché.

  Ils peuvent être placés dans des conteneurs, puis manipulés par des **expressions** polymorphes $F[\ldots]$ dans open t_0 as y in $F[t_1]$

Parenthèse : un peu de vocabulaire

Un besoin intrinsèque

Besoin de méthodes polymorphes

Besoin de types existentiels de première classe...

- Pour protégerCacher la représentaion
  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 $\exists\beta.(\beta \to \tau) \times \beta$ où le type de l'argument β est caché.

  Ils peuvent être placés dans des conteneurs, puis manipulés par des **fonctions** polymorphes f dans fun (f) open t_0 as y in f t_1

Parenthèse : un peu de vocabulaire

Un besoin intrinsèque

Besoin de méthodes polymorphes

Besoin de types existentiels de première classe...

- Pour protégerCacher la représentaion
  empêche de forger des fausses données
  ne respectant pas les invariants les données.
- Pour manipuler des données hétérogènes. Par exemple, les calculs gelés (glaçons ype de l'argument β

  Ils peuvent êt nipulés par des **fonctions** polymorphes f dans fun (f) open t_0 as y in $f\ t_1$

La partie polymorphe de la fonction
lui permet d'être appliquée à
la partie inconnue de l'argument

...combiné avec du polymorphisme de première classe.

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

Parenthèse : un peu de vocabulaire

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 λ-calcul

$$\mathsf{t} ::= \mathsf{x} \mid \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} \mid \mathsf{t}_1\ \mathsf{t}_2$$

$$\mathsf{x} ::= \mathsf{z} \mid \mathsf{c}$$

identificateur — variable — constante

Types avec quantificateurs imbriqués

$$\sigma ::= \alpha \mid \sigma \to \sigma \mid \forall\, \alpha.\, \sigma$$

Par exemple

$$\forall\, \alpha\gamma.(\forall\, \alpha.\, \alpha \to \alpha) \to (\beta \times \gamma) \to (\beta \times \gamma) \quad \text{ou}$$

$$\forall\, \alpha.\, \Big(\forall\, \beta.\, \tau \to \alpha\Big) \to \alpha$$

$$\approx \exists\, \beta.\, \tau$$

e.g. τ égal à $(\beta \to \mathsf{int}) \times \beta$

Pas de types principaux

La fonction fun (z) z z a les types $\begin{cases} (\forall\,\alpha.\,\alpha) \to (\forall\,\alpha.\,\alpha) & (1) \\ (\forall\,\alpha.\,\alpha \to \alpha) \to (\forall\,\alpha.\,\alpha \to \alpha) & (2) \\ \ldots & (3) \end{cases}$

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

C'est toujours un problème pour l'inférence.

Solution : typage explicite, à la Church

$$\mathsf{t} ::= \mathsf{x} \mid \mathsf{fun}\ (\mathsf{z} \quad)\ \mathsf{t} \mid \mathsf{t}_1\ \mathsf{t}_2$$

Pas de types principaux

La fonction fun (z) z z a les types $\begin{cases} (\forall\,\alpha.\,\alpha) \to (\forall\,\alpha.\,\alpha) & (1) \\ (\forall\,\alpha.\,\alpha \to \alpha) \to (\forall\,\alpha.\,\alpha \to \alpha) & (2) \\ \ldots & (3) \end{cases}$

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

C'est toujours un problème pour l'inférence.

Solution : typage explicite, à la Church

$$\mathsf{t} ::= \mathsf{x} \mid \mathsf{fun}\ (\mathsf{z} : \tau)\ \mathsf{t} \mid \mathsf{t}_1\ \mathsf{t}_2 \mid \mathsf{Fun}\ (\alpha)\ \mathsf{t} \mid \mathsf{t}\ \tau$$

Pas de types principaux

La fonction fun (z) z z a les types $\begin{cases} (\forall\,\alpha.\,\alpha) \to (\forall\,\alpha.\,\alpha) & (1) \\ (\forall\,\alpha.\,\alpha \to \alpha) \to (\forall\,\alpha.\,\alpha \to \alpha) & (2) \\ \ldots & (3) \end{cases}$

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

Solution : typage explicite, à la Church

$$\mathsf{t} ::= \mathsf{x} \mid \mathsf{fun}\ (\mathsf{z} : \tau)\ \mathsf{t} \mid \mathsf{t}_1\ \mathsf{t}_2 \mid \mathsf{Fun}\ (\alpha)\ \mathsf{t} \mid \mathsf{t}\ \tau$$

On peut définir différentes versions $\begin{cases} \mathsf{fun}\ (\mathsf{z} : \forall\,\alpha.\,\alpha)\ \mathsf{z} \\ \mathsf{fun}\ (\mathsf{z} : \forall\,\alpha.\,\alpha \to \alpha)\ \mathsf{z} \ldots \end{cases}$

Inconvénient lourdeur d'écriture (temps et lisibilité).

Pas de types principaux

La fonction fun (z) z z a les types $\begin{cases} (\forall\,\alpha.\,\alpha) \to (\forall\,\alpha.\,\alpha) & (1) \\ (\forall\,\alpha.\,\alpha \to \alpha) \to (\forall\,\alpha.\,\alpha \to \alpha) & (2) \\ \ldots & (3) \end{cases}$

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 fun (z) z z a les types
$$\begin{cases} (\forall\,\alpha.\,\alpha) \to (\forall\,\alpha.\,\alpha) & (1) \\ (\forall\,\alpha.\,\alpha \to \alpha) \to (\forall\,\alpha.\,\alpha \to \alpha) & (2) \\ \ldots & (3) \end{cases}$$

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 fun (z) z z a les types $\begin{cases} (\forall\,\alpha.\,\alpha) \to (\forall\,\alpha.\,\alpha) & (1) \\ (\forall\,\alpha.\,\alpha \to \alpha) \to (\forall\,\alpha.\,\alpha \to \alpha) & (2) \\ \ldots & (3) \end{cases}$

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 spéficier 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.

  Fun (α) (fun (z : $\forall\, \beta.\, \beta \to \beta$) z $(\alpha \to \alpha)$ z) (Fun (γ) fun (y : γ) 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.

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

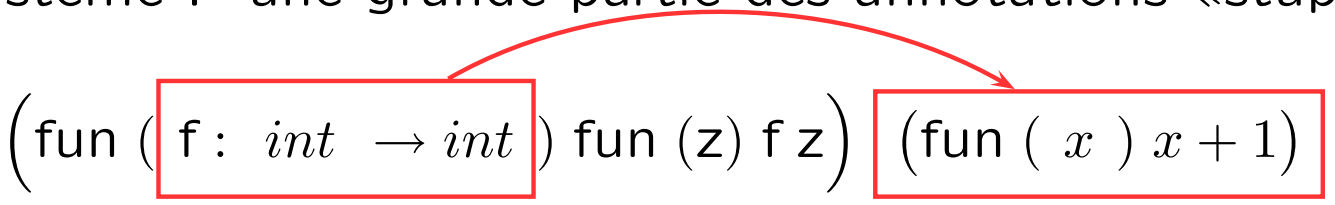
- En faisant seuleument de l'inférence locale
  [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupides» :

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

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

Partir du Système F et s'approcher de ML

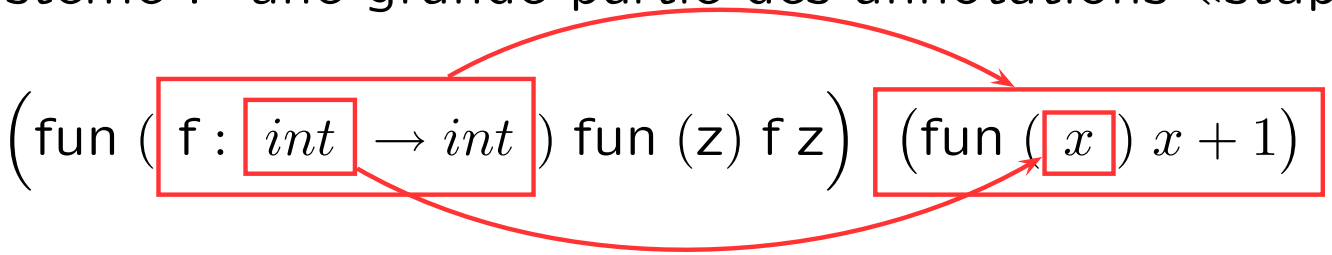
- En utilisant l'unification d'ordre 2.
- En faisant seuleument de l'inférence locale
  [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupides» :



  Les contraintes de typage impliquent *localement* que x a le type 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 seuleument 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» :



  Les contraintes de typage impliquent *localement* que x a le type 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 seuleument 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» :

$$\Big(\text{fun (f : } \; int \; \rightarrow int \; \text{) fun (z) f z}\Big) \;\; \big(\text{fun (} x \text{) } x + 1\big)$$

  Les contraintes de typage impliquent *localement* que x a le type 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

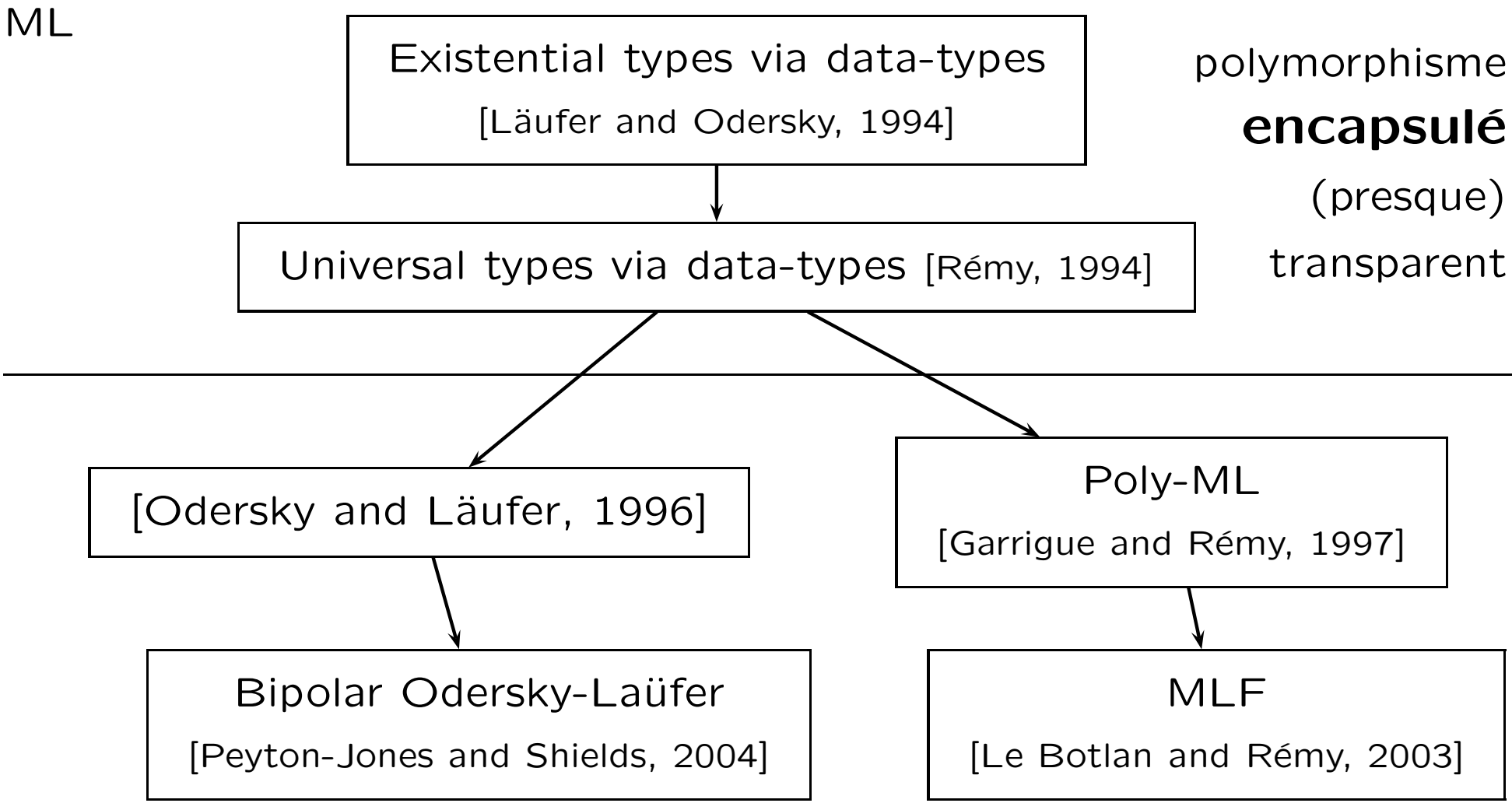
- En utilisant l'unification d'ordre 2.
- En faisant seuleument de l'inférence locale
  [Pierce and Turner, 1998] et [Odersky et al., 2001] pour éliminer dans le système F une grande partie des annotations «stupides» :

$$\Big(\text{fun (f : } \mathit{int} \rightarrow \mathit{int} \text{) fun (z) f z}\Big) \ \ \big(\text{fun (} x \text{) } x+1\big)$$

  Les contraintes de typage impliquent *localement* que x a le type 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 avec des types de second ordre



Prédicatif... *vs* Imprédicatif...

Polymorphism de second ordre

Polymorphisme encapsulé

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.

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

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

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

$$\text{let } auto\ (Id\ f) = f\ f$$

$$auto\ id$$

Polymorphisme encapsulé

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.

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

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 prédicatifs F_p
- Inférence partielle à la ML dans $\mathsf{F}_p^{\Downarrow}$

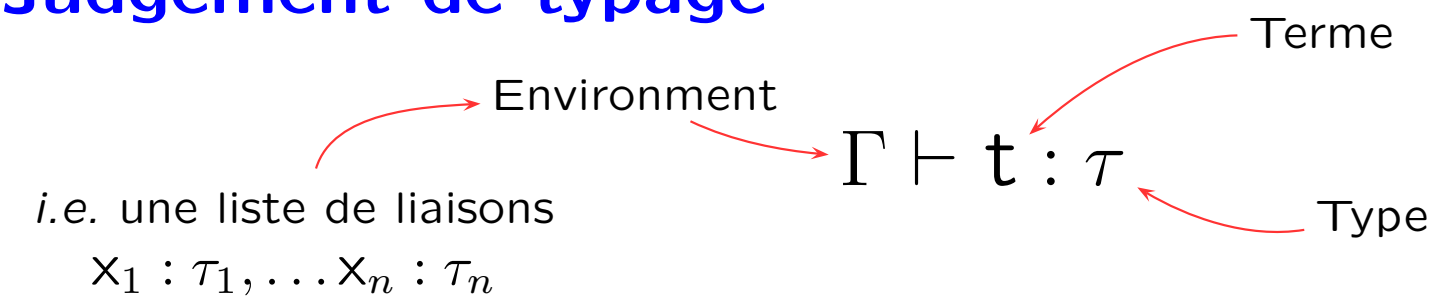
  Propagation des annotations $\mathsf{F}_p^{\Uparrow}$ and $\mathsf{F}_p^{\Updownarrow}$
- Une approche plus ambitieuse: ML^{F}
- Conclusions

  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

$$\mathsf{R}\ \frac{P_1 \qquad \ldots \qquad P_n}{C}$$

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

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

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

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

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.

Une dérivation

$$\dfrac{\dfrac{\overline{P_{11}} \qquad \overline{P_{12}}}{C_{11}} \qquad \overline{P_2}}{C_2}$$

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

Règles de typage (implicite) pour F^X

Var

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

App

$$\frac{\Gamma \vdash a_1 : \sigma_2 \to \sigma_1 \qquad \Gamma \vdash a_2 : \sigma_2}{\Gamma \vdash a_1\ a_2 : \sigma_1}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \sigma \vdash a : \sigma'}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ a : \sigma \to \sigma'}$$

Gen

$$\frac{\Gamma \vdash a : \sigma \qquad \alpha \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash a : \forall\, \alpha.\, \sigma}$$

Inst

$$\frac{\Gamma \vdash a : \sigma \qquad \sigma \leq_X \sigma'}{\Gamma \vdash a : \sigma'}$$

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

Exemple de dérivation

Soit σ_{id} le schéma $\forall\,\alpha.\,\alpha \to \alpha$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var} \qquad \sigma_{id} \leq \sigma_{id} \to \sigma_{id}\ (1)}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Sub} \qquad \dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var}}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z}\ \mathsf{z} : \sigma_{id}}\,\textsf{App}}{\vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{z}\ \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Fun}$$

Exemple de dérivation

Soit σ_{id} le schéma $\forall\,\alpha.\,\alpha \to \alpha$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var} \qquad \sigma_{id} \leq \sigma_{id} \to \sigma_{id}\ (1)}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Sub} \qquad \dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var}}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z}\ \mathsf{z} : \sigma_{id}}\,\textsf{App}}{\vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{z}\ \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Fun}$$

Une autre ?

Exemple de dérivation

Soit σ_{id} le schéma $\forall\,\alpha.\,\alpha \to \alpha$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var} \qquad \sigma_{id} \leq \sigma_{id} \to \sigma_{id}\ (1)}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Sub} \qquad \dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var}}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z}\ \mathsf{z} : \sigma_{id}}\,\textsf{App}}{\vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{z}\ \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Fun}$$

Une autre ? Soit τ_α le type $\alpha \to \alpha$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var} \qquad \sigma_{id} \leq \tau_\alpha \to \tau_\alpha}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \tau_\alpha}\,\textsf{Sub} \qquad \dfrac{\dfrac{}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \sigma_{id}}\,\textsf{Var} \qquad \sigma_{id} \leq \tau_\alpha}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z} : \tau_\alpha}\,\textsf{Sub}}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z}\ \mathsf{z} : \tau_\alpha}\,\textsf{App}}{\mathsf{z} : \sigma_{id} \vdash \mathsf{z}\ \mathsf{z} : \sigma_{id}}\,\textsf{Gen} \qquad \alpha \notin \mathsf{ftv}(\mathsf{z} : \sigma_{id})}{\vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{z}\ \mathsf{z} : \sigma_{id} \to \sigma_{id}}\,\textsf{Fun}$$

Instantiation $\leq$

$\leq$ pour le Système F

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

Sub

$$\frac{\bar{\beta} \notin \mathsf{ftv}(\forall\, \bar{\alpha}.\, \sigma)}{\forall\, \bar{\alpha}.\, \sigma \leqslant \forall\, \bar{\beta}.\, \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Par exemple, $\forall\, \alpha.\, \alpha \to \alpha \leq$

$$\begin{cases} \mathsf{int} \to \mathsf{int} \\ \forall\, \alpha, \beta.(\alpha \to \beta) \to (\alpha \to \beta) \\ (\forall\, \alpha.\, \alpha \to \alpha) \to (\forall\, \alpha.\, \alpha \to \alpha) \end{cases} \quad (1)$$

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

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

Contrainte plus exigeante sur l'argument

Contrainte moins exigeante sur le résultat

Instantiation $\leq^\eta$ [Mitchell, 1988]

$\leq^\eta$ pour le Système F$^\eta$

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

Contra-variance

Sub

$$\frac{\bar\beta \notin \mathsf{ftv}(\forall\,\bar\alpha.\,\sigma)}{\forall\,\bar\alpha.\,\sigma \leqslant \forall\,\bar\beta.\,\sigma[\bar\sigma/\bar\alpha]}$$

Trans

$$\frac{\sigma \leqslant \sigma' \qquad \sigma' \leqslant \sigma''}{\sigma \leqslant \sigma''}$$

Arrow

$$\frac{\sigma'_1 \leqslant \sigma_1 \qquad \sigma_2 \leqslant \sigma'_2}{\sigma_1 \to \sigma_2 \leqslant \sigma'_1 \to \sigma'_2}$$

All

$$\frac{\sigma \leqslant \sigma'}{\forall\,\alpha.\,\sigma \leqslant \forall\,\alpha.\,\sigma'}$$

Distrib

$$\forall\,\alpha.\,\sigma \to \sigma' \leqslant (\forall\,\alpha.\,\sigma) \to \forall\,\alpha.\,\sigma'$$

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

— attend une expresssion de type τ_1,

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

— retourne une expression de type τ_2,

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

Instantiation $\leq^\eta$ [Mitchell, 1988]

$\leq^\eta$ pour le Système F$^\eta$

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

Contra-variance

Sub

$$\frac{\bar{\beta} \notin \mathsf{ftv}(\forall\, \bar{\alpha}.\, \sigma)}{\forall\, \bar{\alpha}.\, \sigma \leqslant \forall\, \bar{\beta}.\, \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Trans

$$\frac{\sigma \leqslant \sigma' \qquad \sigma' \leqslant \sigma''}{\sigma \leqslant \sigma''}$$

Arrow

$$\frac{\sigma'_1 \leqslant \sigma_1 \qquad \sigma_2 \leqslant \sigma'_2}{\sigma_1 \to \sigma_2 \leqslant \sigma'_1 \to \sigma'_2}$$

All

$$\frac{\sigma \leqslant \sigma'}{\forall\, \alpha.\, \sigma \leqslant \forall\, \alpha.\, \sigma'}$$

Distrib

$$\forall\, \alpha.\, \sigma \to \sigma' \leqslant (\forall\, \alpha.\, \sigma) \to \forall\, \alpha.\, \sigma'$$

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

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

Distrib-Left

$$\forall\, \alpha.\, \sigma \to \sigma' \leqslant (\forall\, \alpha.\, \sigma) \to \sigma'$$

Distrib-Right

$$\forall\, \alpha.\, \sigma \to \sigma' \leqslant \sigma \to \forall\, \alpha.\, \sigma'$$

(also $\geqslant$ by Sub+All)

Instantiation $\leq^\eta$ [Mitchell, 1988]

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

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

Contra-variance

Sub
$$\frac{\bar{\beta} \notin \mathsf{ftv}(\forall\, \bar{\alpha}.\, \sigma)}{\forall\, \bar{\alpha}.\, \sigma \leqslant \forall\, \bar{\beta}.\, \sigma[\bar{\sigma}/\bar{\alpha}]}$$

Trans
$$\frac{\sigma \leqslant \sigma' \qquad \sigma' \leqslant \sigma''}{\sigma \leqslant \sigma''}$$

Arrow
$$\frac{\sigma'_1 \leqslant \sigma_1 \qquad \sigma_2 \leqslant \sigma'_2}{\sigma_1 \to \sigma_2 \leqslant \sigma'_1 \to \sigma'_2}$$

All
$$\frac{\sigma \leqslant \sigma'}{\forall\, \alpha.\, \sigma \leqslant \forall\, \alpha.\, \sigma'}$$

Distrib
$$\forall\, \alpha.\, \sigma \to \sigma' \leqslant (\forall\, \alpha.\, \sigma) \to \forall\, \alpha.\, \sigma'$$

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

Correction du typage

La réduction *préserve* le typage dans les deux systèmes F et F^{η}.
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^{η} 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 prédicatif du Système F

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

Dans le système F_p on distingue :

$$\tau ::= \alpha \mid \tau \to \tau \qquad \text{monotypes}$$

$$\sigma ::= \tau \mid \sigma \to \sigma \mid \forall\, \alpha.\, \sigma \qquad \text{(poly)types}$$

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 :

Sub_p

$$\frac{\bar{\beta} \notin \mathsf{ftv}(\forall\, \bar{\alpha}.\, \sigma)}{\forall\, \bar{\alpha}.\, \sigma \leqslant \forall\, \bar{\beta}.\, \sigma[\bar{\tau}/\bar{\alpha}]}$$

Au lieu de $\bar{\sigma}$

F^{η}_{p} : Le fragment prédicatif du Système F^{η}

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

Dans le système F_p on distingue :

$$\tau ::= \alpha \mid \tau \to \tau \qquad \text{monotypes}$$

$$\sigma ::= \tau \mid \sigma \to \sigma \mid \forall\, \alpha.\, \sigma \qquad \text{(poly)types}$$

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 :

Sub_p

$$\frac{\bar{\beta} \notin \mathsf{ftv}(\forall\, \bar{\alpha}.\, \sigma)}{\forall\, \bar{\alpha}.\, \sigma \leqslant \forall\, \bar{\beta}.\, \sigma[\bar{\tau}/\bar{\alpha}]}$$

Au lieu de $\bar{\sigma}$

La restriction est significative

- Le codage des existentiels :

  Lorsqu'on cache $\sigma[\tau/\beta]$ en $\exists\,\beta.\,\sigma$, alors τ 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

$$\exists\,\alpha_R.\,(\alpha_R \times (\alpha_R \to \sigma_M))$$

  où α_R cache l'état interne. L'état devrait être monomorphic, *i.e.* ne pas contenir d'autres objets.

La restriction est significative

- apply (or map, iter, *etc.*)

$$\text{let apply} = \text{fun (f) fun (z) f z in } \ldots$$

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 recouvré 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 prédicatifs F_p
- Inférence partielle à la ML dans $\mathsf{F}_p^{\Downarrow}$
  Propagation des annotations $\mathsf{F}_p^{\Uparrow}$ and $\mathsf{F}_p^{\Updownarrow}$
- 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 role clé (en laissant de la place à l'inférence) que σ.

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

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

Règles de typage : ML

Var

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

Inst

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

Gen

$$\frac{\Gamma \vdash t : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash t : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash t_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash t_1\ t_2 : \tau_1}$$

Fun

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

Let

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

Règles de typage : ML avec annotations

Var

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

Inst

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

Gen

$$\frac{\Gamma \vdash t : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash t : \forall\, \bar{\alpha}.\, \sigma}$$

App

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

Fun

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

Let

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

Règles de typage : $\mathrm{F}^{\Downarrow}_{p}$

Var

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

Inst

$$\frac{\Gamma \vdash \mathsf{t} : \sigma' \qquad \sigma' \leq^{\shortparallel}_{p} \sigma}{\Gamma \vdash \mathsf{t} : \sigma}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Lire $\Gamma \vdash \mathsf{t} : \sigma$ comme des règles de vérification : Γ, t, and σ étant donnés.

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

On obtient ML quand tous les σ sont des β.

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var

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

Inst

$$\frac{\Gamma \vdash \mathsf{t} : \sigma' \qquad \sigma' \leq_p^{\shortparallel} \sigma}{\Gamma \vdash \mathsf{t} : \sigma}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Pour préparer l'inférence :

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

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

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

Inst
$$\frac{\Gamma \vdash \mathsf{t} : \sigma' \qquad \sigma' \leq_p^{\shortparallel} \sigma}{\Gamma \vdash \mathsf{t} : \sigma}$$

Gen
$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App
$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun
$$\frac{\Gamma, \mathsf{z} : \sigma_2 \vdash \mathsf{t} : \sigma_1}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1}$$

Let
$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Fusionner les deux règles.

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

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \sigma}{\Gamma \vdash \mathsf{x} : \sigma}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \sigma}{\Gamma \vdash \mathsf{x} : \sigma}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Changer σ en ρ.

A ρ est un σ sans les quantificateurs externes.

Règles de typage : $\mathsf{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \sigma_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \sigma_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Changer σ_1 en ρ_1, puisque

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

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1}$$

Fun

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

Let

$$\frac{\Gamma \vdash \mathsf{t}_1 : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, x : \forall\, \bar{\alpha}.\, \sigma_1[\bar{\tau}/\bar{\beta}] \vdash \mathsf{t}_2 : \sigma_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Changer σ_2 into ρ_2.

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

Règles de typage : $\mathsf{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1}$$

Fun

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

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle \Gamma \rangle(\sigma_1[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2}$$

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1}$$

Fun

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

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma_1[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2}$$

Gen reste (alors qu'elle disparaît en ML)

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

Règles de typage : $\mathrm{F}_p^{\Downarrow}$

Var-Inst-Rho

$$\frac{\mathsf{x} : \sigma' \in \Gamma \qquad \sigma' \leq_p^{\shortparallel} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2[\bar{\tau}/\bar{\beta}] \to \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \sigma_2[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1}$$

Fun

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

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_1[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma_1[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2}$$

Regardons Let-Gen :

— Le bleu est explicite,

— Le rouhe est inféré.

Relation d'instance $\leq_p^{\shortparallel}$ pour l'inférence

- On retire la règle Distrib

  Raison (a posteriori)

  - La règle Distrib-Right n'est pas correcte pour les effets de bord.
  - La relation $\leq_p^{\shortparallel}$ est décidable (efficacement). $\leq_p^{\eta}$ *l'est-elle ?*

- Conséquence : $\leq_p^{\shortparallel}$ est une sous-relation stricte de $\leq_p^{\eta}$ (contient $\leq_F$). Donc $\mathsf{F}_p^{\Downarrow}$ est moins expressif que F_p^{η}.

- Ensemble équivalent de règles :

Inst-Refl

$$\sigma \leqslant \sigma$$

Inst-Arrow

$$\frac{\sigma_1 \leqslant \sigma_1' \qquad \sigma_2' \leqslant \sigma_2}{\sigma_2 \to \sigma_1 \leqslant \sigma_2' \to \sigma_1'}$$

Inst-Skol

$$\frac{\sigma \leqslant \sigma' \qquad \alpha \notin \mathsf{ftv}(\sigma)}{\sigma \leqslant \forall\, \alpha.\, \sigma'}$$

Inst-Spec

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

Inférence par résolution de contraintes

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

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

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

Interprétation logique des constraintes

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

Voir [Pierce, 2004].

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

Interprétation logique des constraintes

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

Voir [Pierce, 2004].

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

Interprétation logique des constraintes

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

Voir [Pierce, 2004].

$$[\![\mathsf{x} : \rho]\!] \longrightarrow \mathsf{x} \preceq \rho$$

$$[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] \longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha)$$

$$[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] \longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!]$$

$$[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] \longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!])$$

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

$$[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] \longrightarrow \forall\, \bar{\alpha}. [\![\mathsf{t}_2 : \rho]\!]$$

Interprétation logique des constraintes

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

Voir [Pierce, 2004].

$$\tau \leqslant \tau' \longrightarrow \tau = \tau'$$
$$\sigma_1 \to \sigma_2 \leqslant \sigma_1' \to \sigma_2' \longrightarrow \sigma_1' \leqslant \sigma_1 \wedge \sigma_2 \to \sigma_2'$$
$$\forall\, \alpha.\, \sigma \leqslant \rho \longrightarrow \exists\, \alpha.(\sigma \leqslant \rho)$$
$$\sigma \leqslant \forall\, \alpha.\, \sigma' \longrightarrow \forall\, \alpha.(\sigma \leqslant \sigma')$$

- Suit les règles dirigées par la syntaxe pour $\leq_p^{\text{II}}$.
- Permet de simplifierène les contraintes en contraintents d'égalités.

Interprétation logique des constraintes

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

Voir [Pierce, 2004].

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

Interprétation logique des constraintes

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

Voir [Pierce, 2004].

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta} . \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta} . ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta} . \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha} . \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

L'inférence est de premier ordre

- Pas de méta variables pour σ ni ρ, mais seulement pour τ.
- Les formes de la structure polymorphe ne sont que vérifiées.

$$[\![\mathsf{x} : \rho]\!] \longrightarrow \mathsf{x} \preceq \rho$$

$$[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] \longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha)$$

$$[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] \longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!]$$

$$[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1]\!] \longrightarrow \exists\, \bar{\beta}. ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!])$$

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

$$[\![\mathsf{t} : \forall\, \bar{\alpha}.\, \rho]\!] \longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]$$

Un problème d'inférence

$\Gamma \vdash \mathsf{t} : \sigma$ est résolu comme

$\mathsf{let}\ \Gamma\ \mathsf{in}\ [\![\mathsf{t} : \sigma]\!]$

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}. \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta} . ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}. \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}. \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

Un problème d'inférence

trouver φ tel que $\varphi(\Gamma) \vdash \mathsf{t} : \varphi(\sigma)$ est résolu comme
trouvé φ tel que $\varphi \Vdash \mathsf{let}\ \Gamma\ \mathsf{in}\ [\![\mathsf{t} : \sigma]\!]$

Cet algorithme est correct et complet.

$$
\begin{aligned}
[\![\mathsf{x} : \rho]\!] &\longrightarrow \mathsf{x} \preceq \rho \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \alpha]\!] &\longrightarrow \exists\, \beta_1 \beta_2 . ([\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \beta_1 \to \beta_2]\!] \wedge \beta_1 \to \beta_2 \leqslant \alpha) \\
[\![\mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \sigma_2 \to \sigma_1]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \sigma_2\ \mathsf{in}\ [\![\mathsf{t} : \sigma_1]\!] \\
[\![\mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}. \sigma_2) : \rho_1]\!] &\longrightarrow \exists\, \bar{\beta} . ([\![\mathsf{t}_1 : \sigma_2 \to \rho_1]\!] \wedge [\![\mathsf{t}_2 : \sigma_2]\!]) \\
[\![\mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}. \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_2]\!] &\longrightarrow \mathsf{let}\ \mathsf{z} : \forall \bar{\beta} [[\![\mathsf{t}_1 : \sigma_1]\!]] . \sigma_1\ \mathsf{in}\ [\![\mathsf{t}_2 : \rho_2]\!] \\
[\![\mathsf{t} : \forall\, \bar{\alpha}. \rho]\!] &\longrightarrow \forall\, \bar{\alpha} . [\![\mathsf{t}_2 : \rho]\!]
\end{aligned}
$$

Un problème d'inférence

$\Gamma \vdash \mathsf{t} : \sigma$ est résolu comme $\mathsf{let}\ \Gamma\ \mathsf{in}\ [\![\mathsf{t} : \sigma]\!]$

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 constraintes (ci-dessus) pourrait être la spécification du typage.

Exemples

$$\text{let } \mathit{id} : \forall\, \alpha.\ \alpha \to \alpha = \text{fun } x \to x$$

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

$$\text{let } \_ = \mathit{auto}\ (\mathit{id} : \forall\, \alpha.\ \alpha \to \alpha)$$

Notez la différence avec

$$\text{let } \mathit{id} : \forall\, (\alpha)\ \alpha \to \alpha = \text{fun } x \to x$$

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

$$\text{let } \_ = \mathit{auto}\ (\mathit{id} : \forall\, (\alpha)\ \alpha \to \alpha)$$

Inférence de types = vérification de formes

Le slogan, répété

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

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

Les formes $\mathcal{S}$

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

Le slogan, répété

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

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

Les formes $\mathcal{S}$: polytypes clos étendus avec $\sharp$ et l'équation $\sharp = \sharp \to \sharp$.

Operations sur les formes

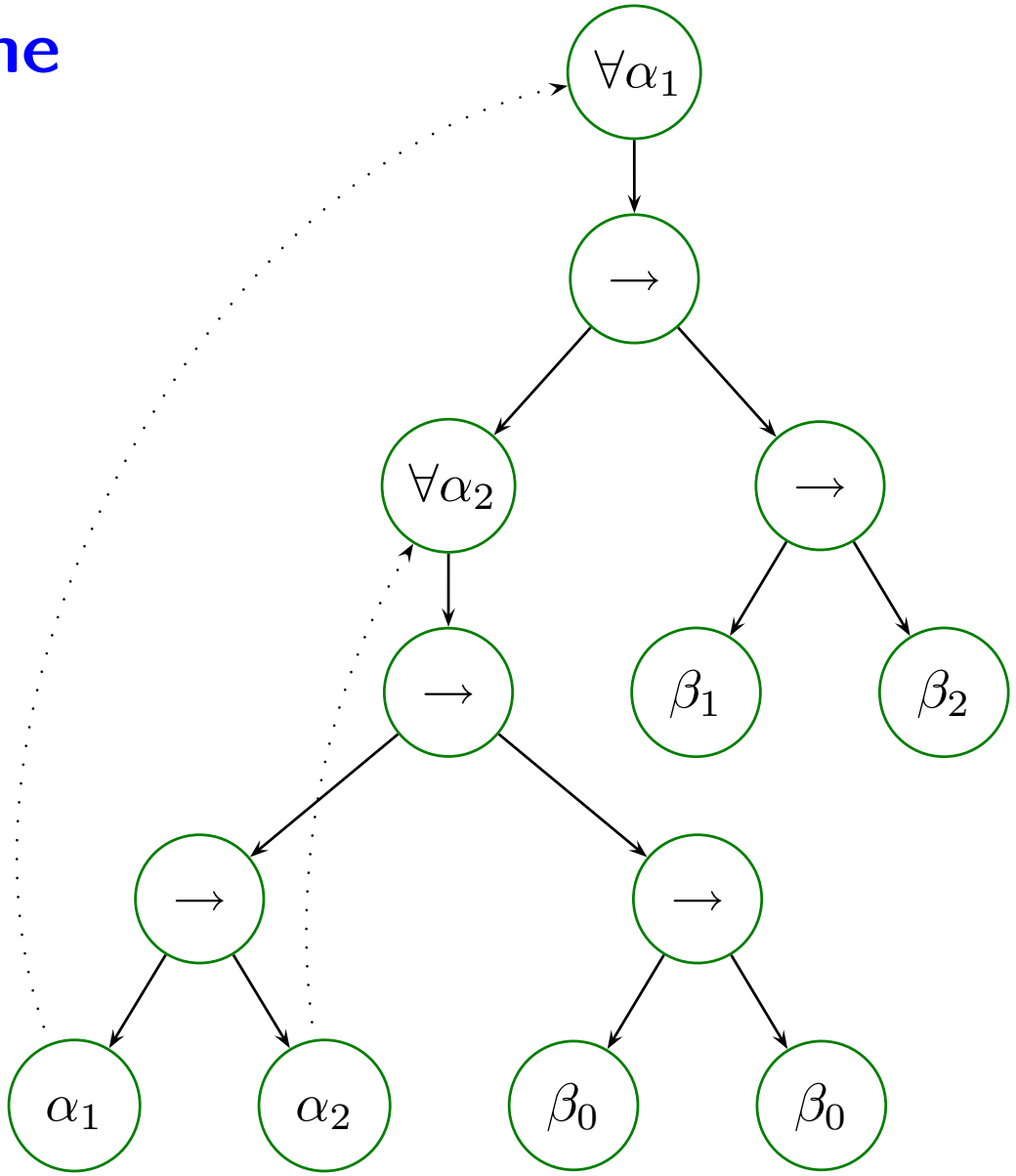
$\lceil\sigma\rceil$ retourne la forme de σ, *i.e.* $\sigma[\sharp/\mathrm{ftv}(\sigma)]$

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

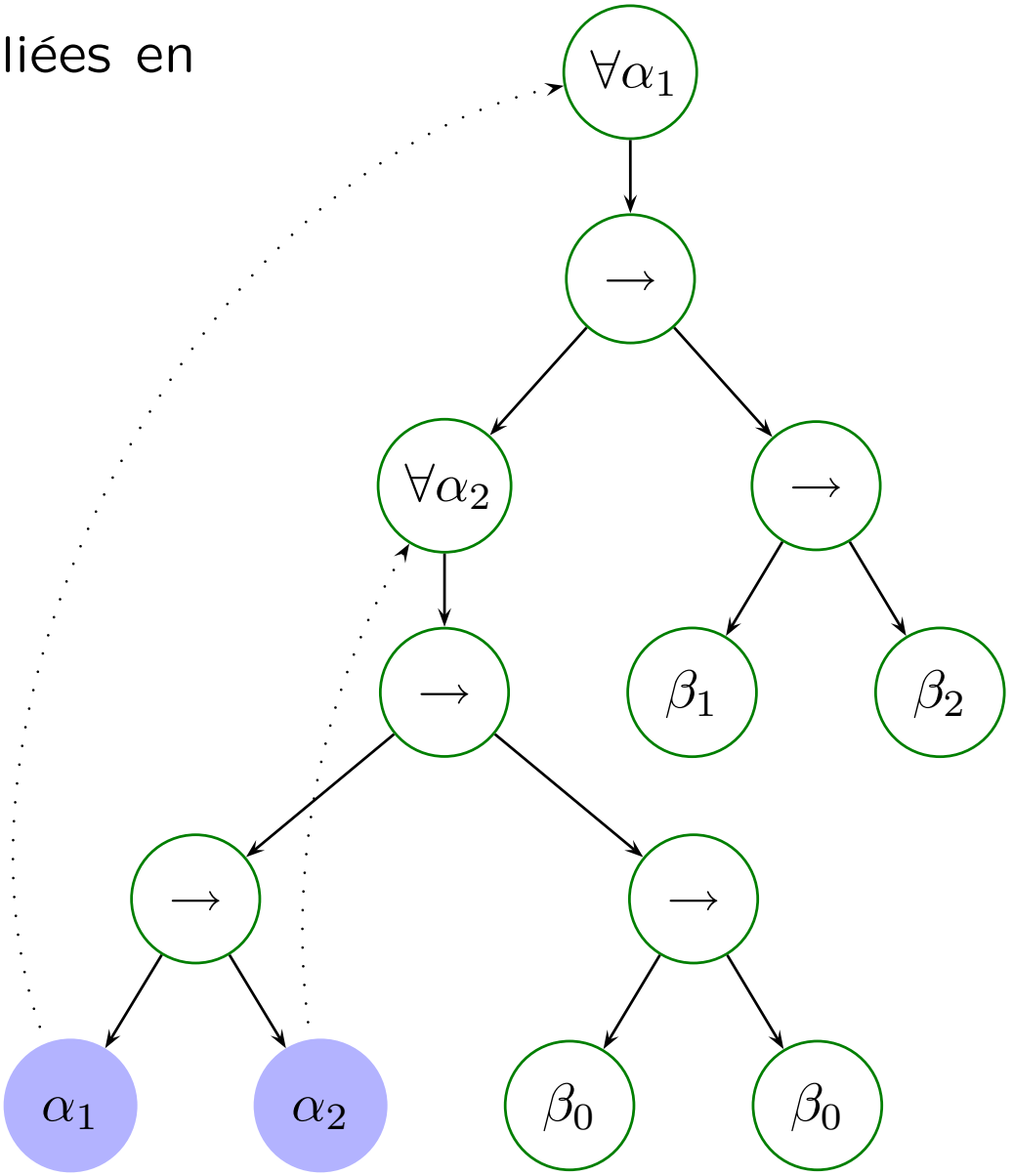
On écrit $\mathcal{R}$ pour les formes ainsi dénudées.

$\lfloor\mathcal{S}\rfloor$ retourne l'annotation $\exists\,\bar{\beta}.\,\mathcal{S}[\beta_i/\sharp_i]$.

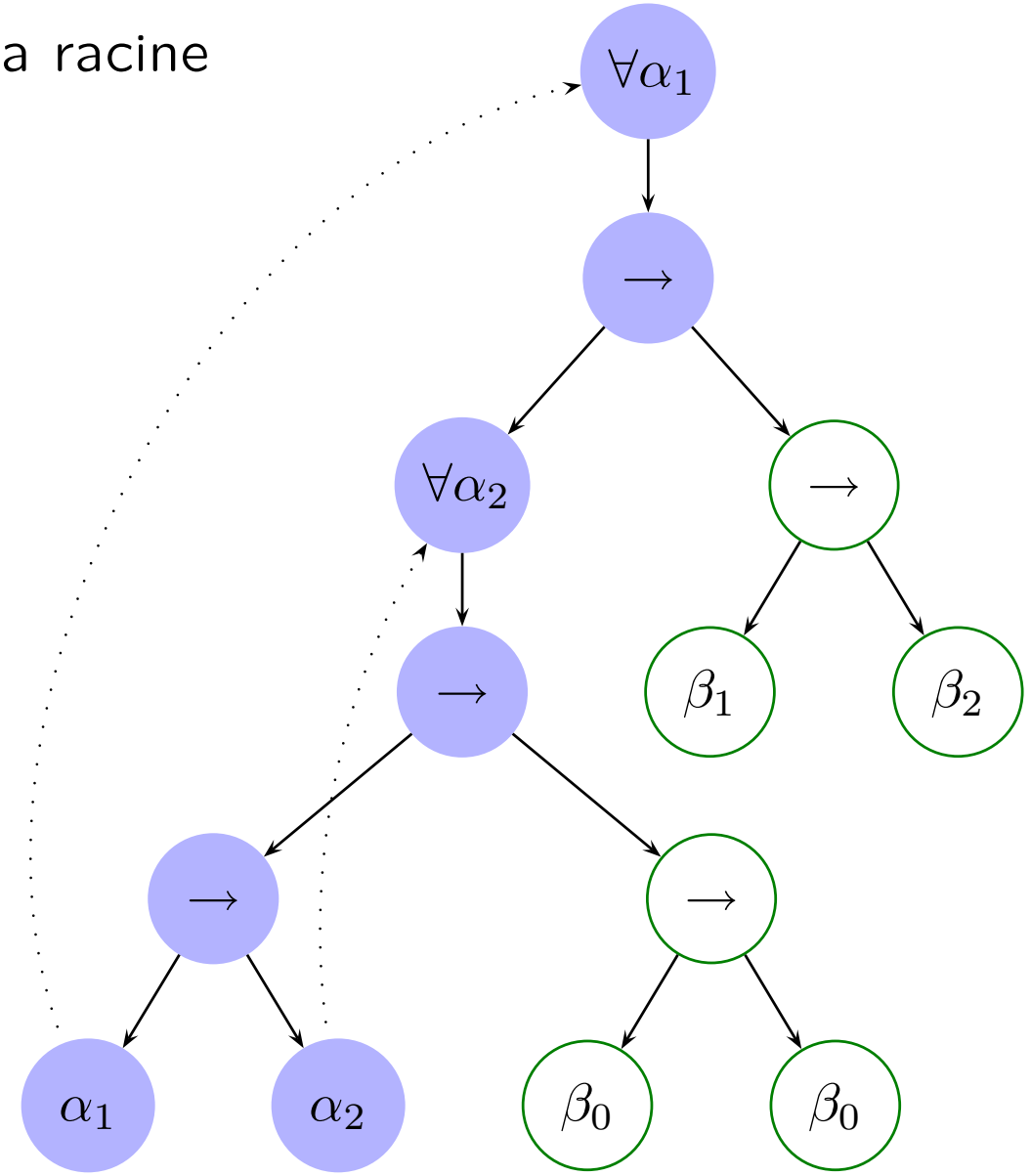
Calcul de la forme



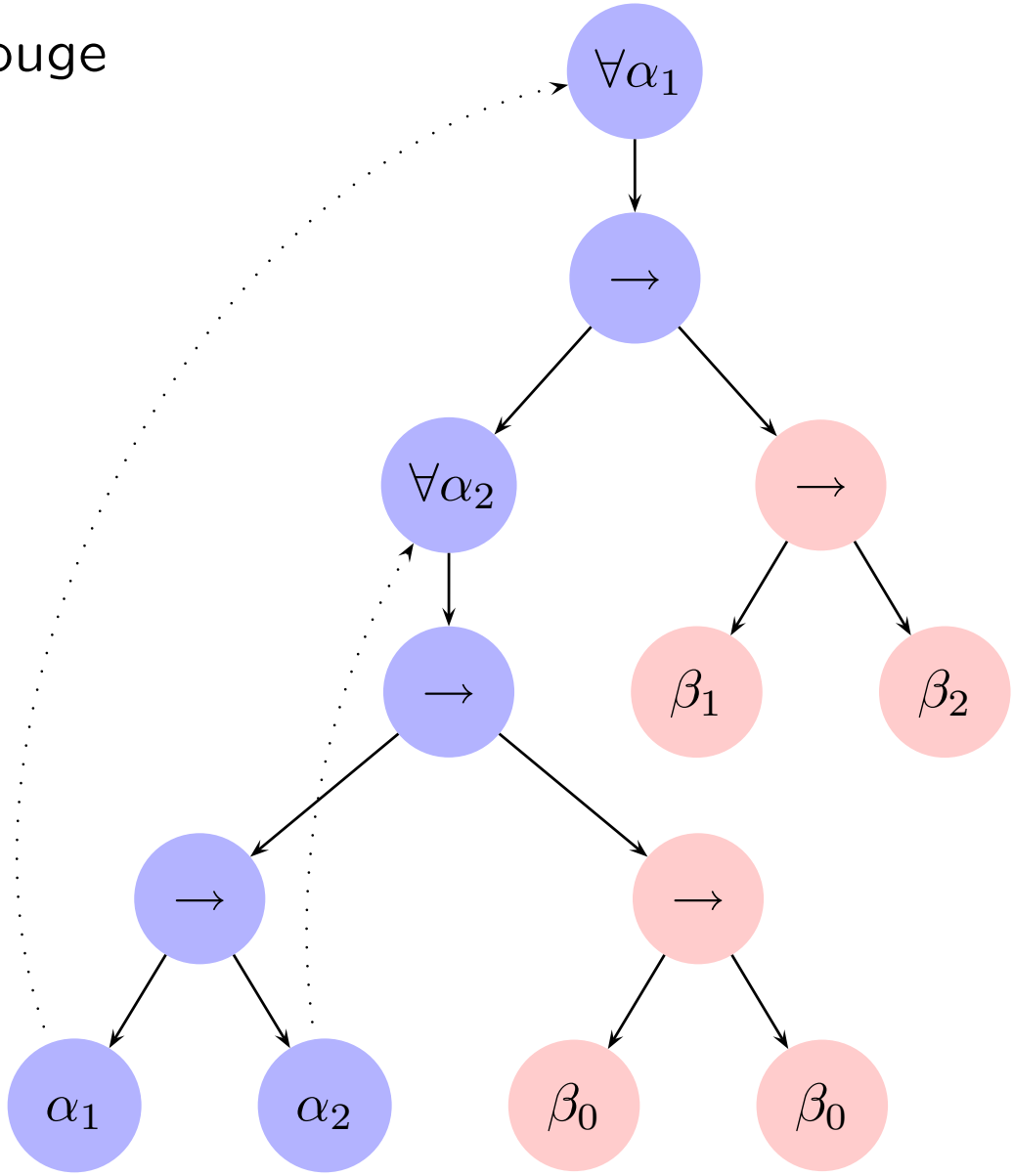
Marquer les variables liées en bleu



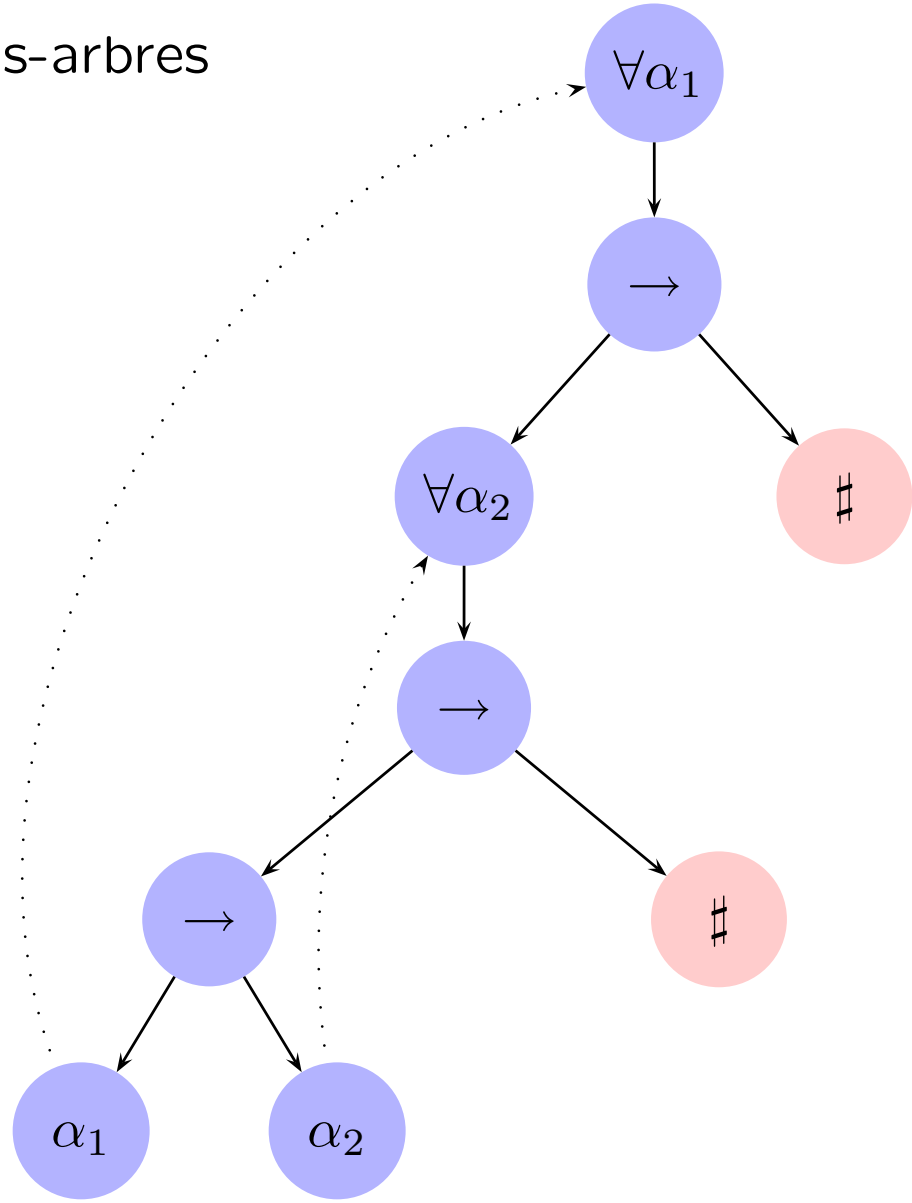
Étendre le bleu vers la racine



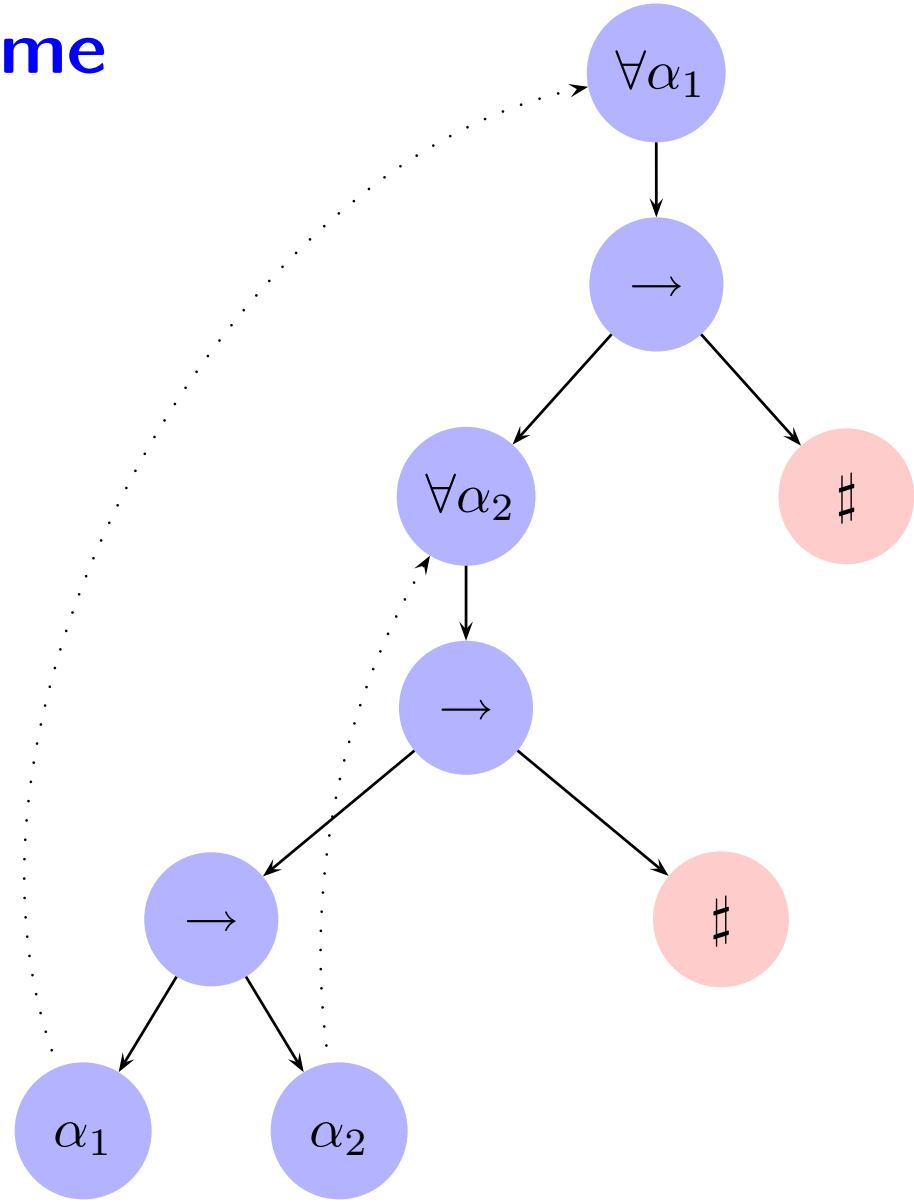
Marquer le reste en rouge



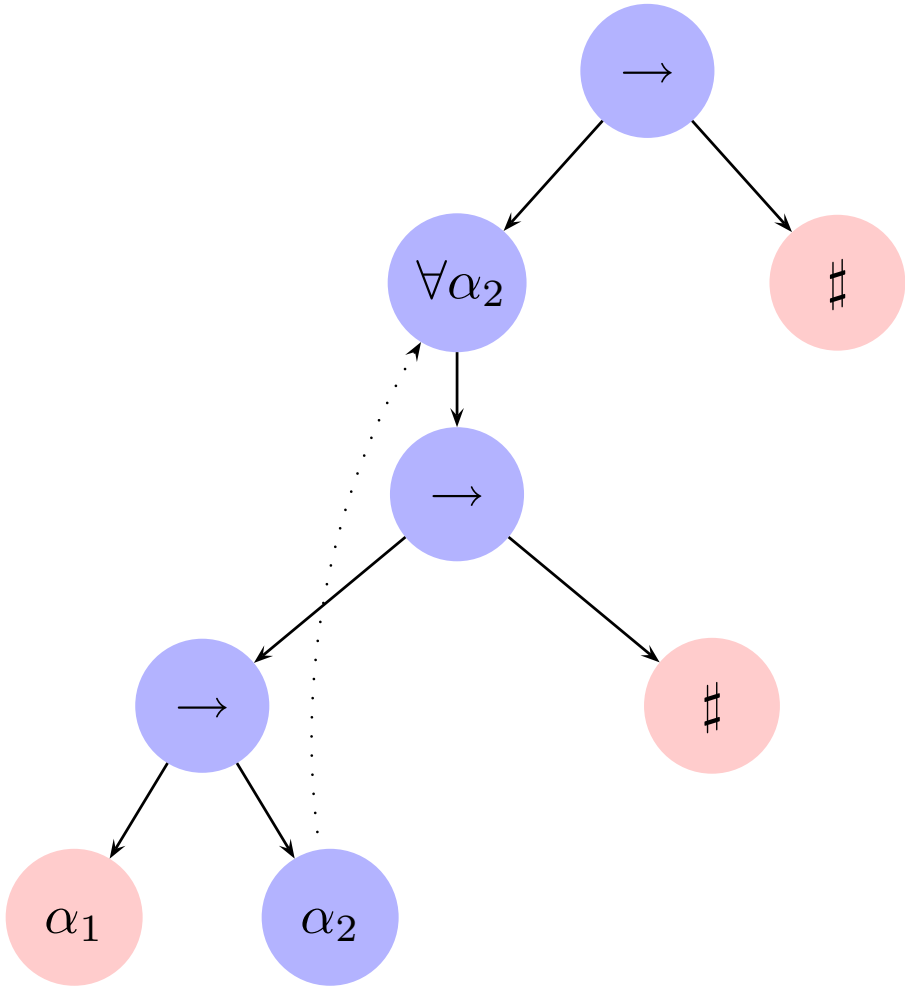
Remplacer les sous-arbres rouges par ♯



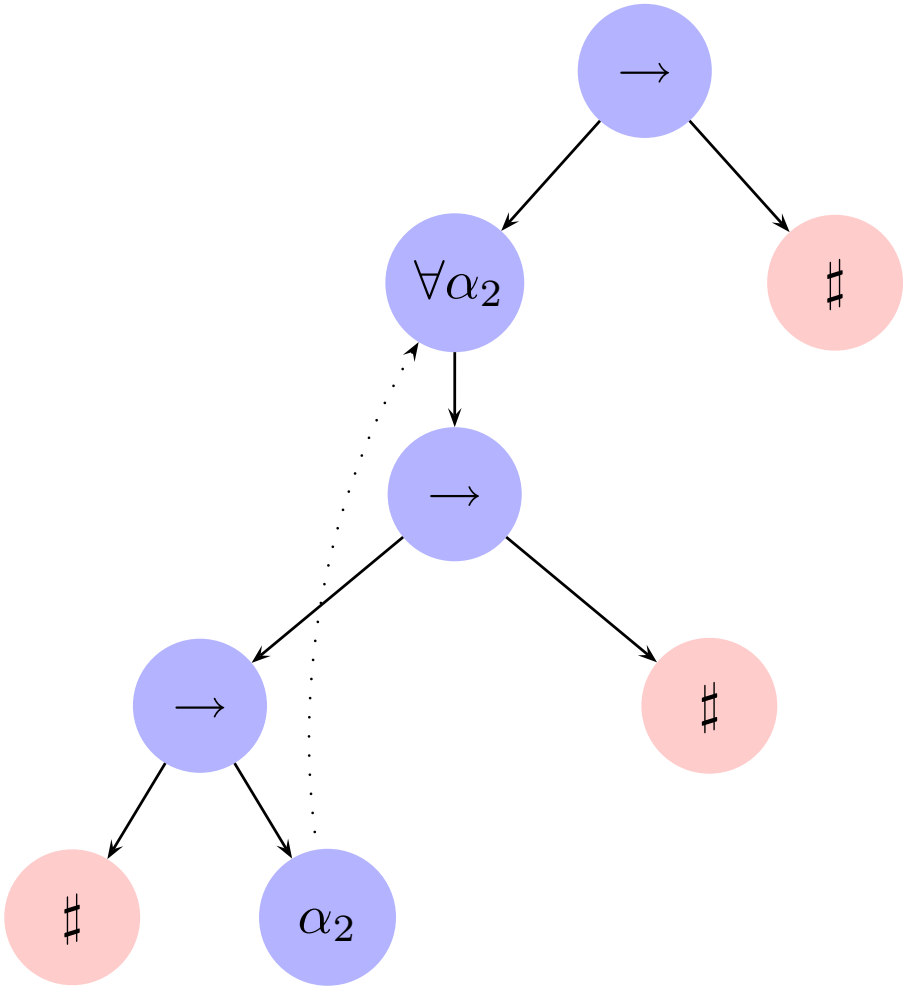
Dénuder une forme



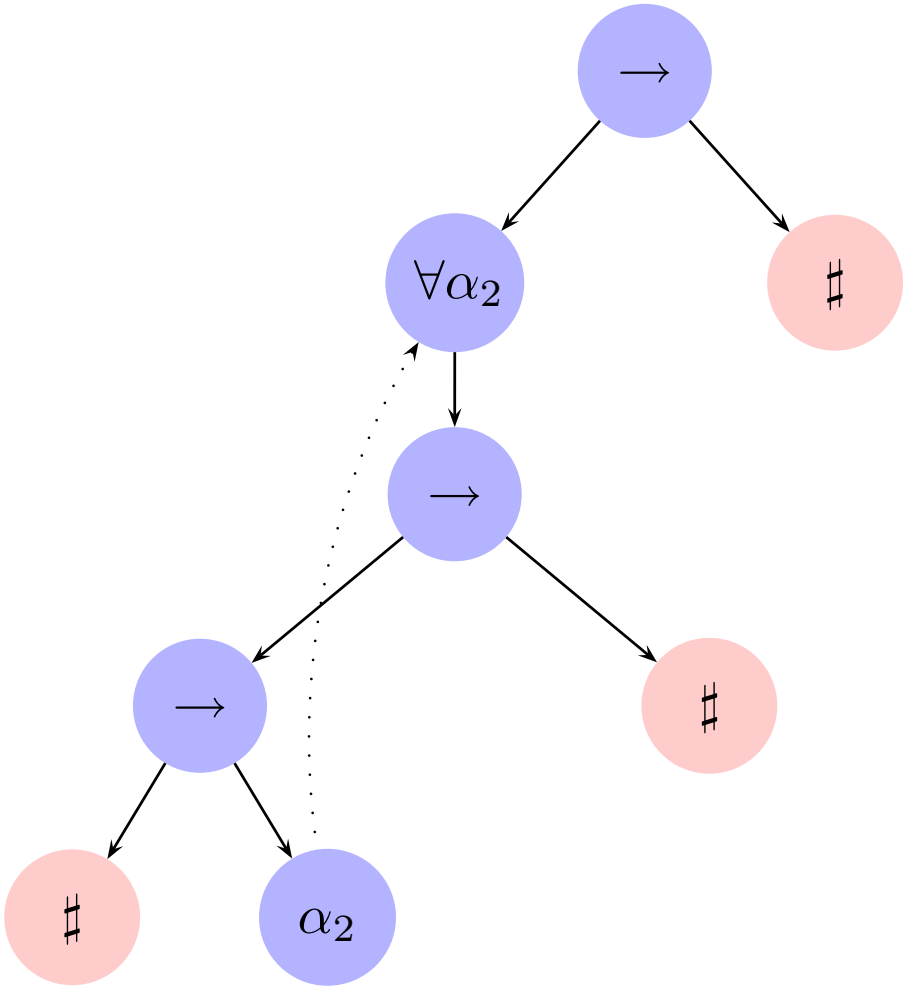
Retirer les quantificateurs en tête



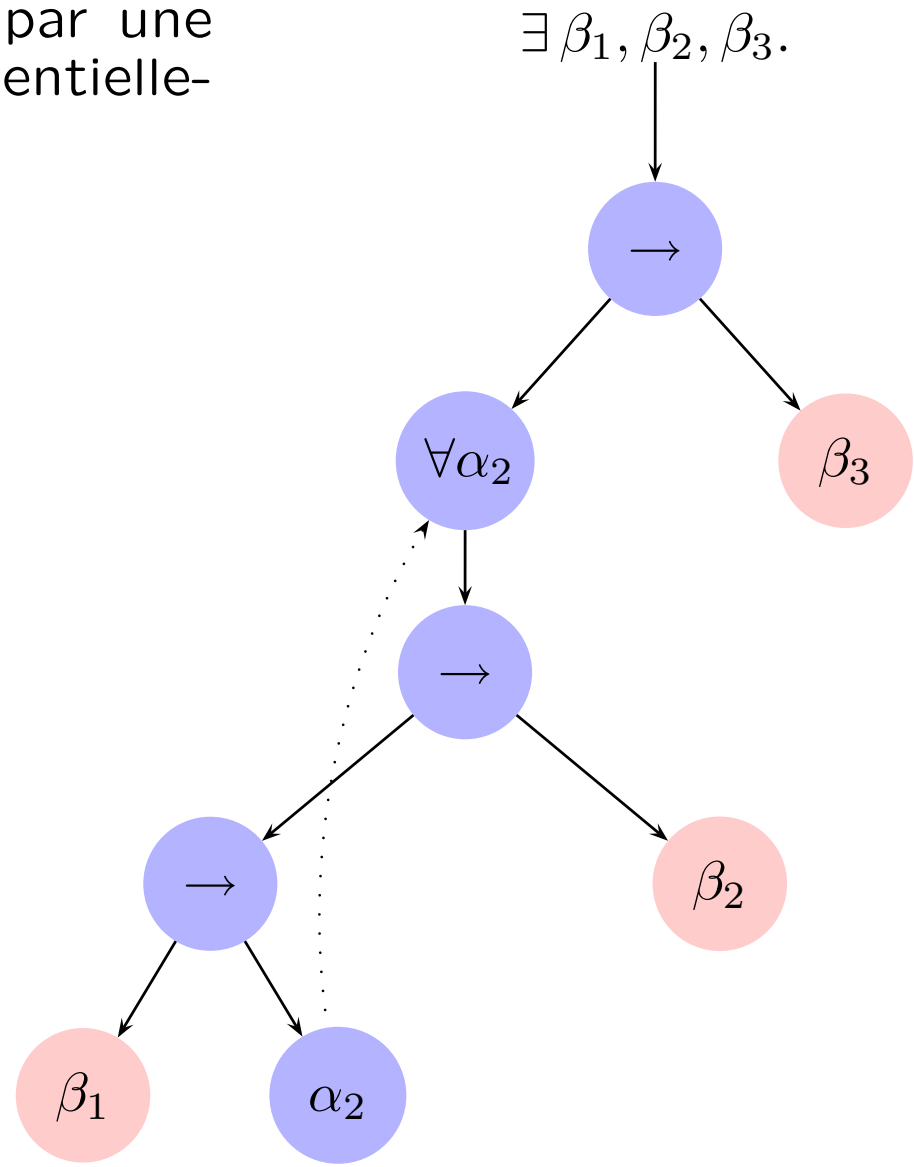
Reformer



Construire une annotation à partir d'une forme



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



Vérification des formes

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

Var-Rho

$$\frac{\mathsf{x} : \sigma \in \Gamma \qquad \sigma \leq^{\shortparallel}_{p} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Fun

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

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

Vérification des formes

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Gen

$$\frac{\Gamma \vdash \mathsf{t} : \sigma \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma}$$

Var-Rho

$$\frac{\mathsf{x} : \sigma \in \Gamma \qquad \sigma \leq^{\shortparallel}_{p} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Fun

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

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \sigma \in \Gamma \qquad \sigma \leq^{\shortparallel}_{p} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Fun

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

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \sigma \in \Gamma \qquad \sigma \leq^{\shortparallel}_{p} \rho}{\Gamma \vdash \mathsf{x} : \rho}$$

Fun

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

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

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

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

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

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

Vérification des formes

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq_p^{\shortparallel} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^\flat \vdash \mathsf{t} : \mathcal{S}_1^\flat}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

Vérification des formes

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^{\flat} \vdash \mathsf{t} : \mathcal{S}_1^{\flat}}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \to \rho \qquad \Gamma \vdash \mathsf{t}_2 : \sigma[\bar{\tau}/\bar{\beta}]}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \rho}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^{\flat} \vdash \mathsf{t} : \mathcal{S}_1^{\flat}}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma[\bar{\tau}/\bar{\beta}] \rceil \to \mathcal{R} \qquad \Gamma \vdash \mathsf{t}_2 : \lceil \sigma[\bar{\tau}/\bar{\beta}] \rceil^{\flat}}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \mathcal{R}}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

Vérification des formes

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^\flat \vdash \mathsf{t} : \mathcal{S}_1^\flat}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil \to \mathcal{S} \qquad \Gamma \vdash \mathsf{t}_2 : \lceil \sigma \rceil^\flat}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \mathcal{S}^\flat}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle \Gamma \rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^{\flat} \vdash \mathsf{t} : \mathcal{S}_1^{\flat}}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil \to \mathcal{S} \qquad \Gamma \vdash \mathsf{t}_2 : \lceil \sigma \rceil^{\flat}}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \mathcal{S}^{\flat}}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma[\bar{\tau}/\bar{\beta}] \qquad \Gamma, \mathsf{z} : \langle \Gamma \rangle(\sigma[\bar{\tau}/\bar{\beta}]) \vdash \mathsf{t}_2 : \rho}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \rho}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^\flat \vdash \mathsf{t} : \mathcal{S}_1^\flat}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil \to \mathcal{S} \qquad \Gamma \vdash \mathsf{t}_2 : \lceil \sigma \rceil^\flat}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \mathcal{S}^\flat}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma[\bar{\tau}/\bar{\beta}] \rceil^\flat \qquad \Gamma, \mathsf{z} : \lceil \langle \Gamma \rangle (\sigma[\bar{\tau}/\bar{\beta}]) \rceil^\flat \vdash \mathsf{t}_2 : \mathcal{R}}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \mathcal{R}}$$

Vérification des formes

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^{\flat} \vdash \mathsf{t} : \mathcal{S}_1^{\flat}}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil \to \mathcal{S} \qquad \Gamma \vdash \mathsf{t}_2 : \lceil \sigma \rceil^{\flat}}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \mathcal{S}^{\flat}}$$

Let-Gen-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil^{\flat} \qquad \Gamma, \mathsf{z} : \lceil \sigma \rceil^{\flat} \vdash \mathsf{t}_2 : \mathcal{R}}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \mathcal{R}}$$

$$\Gamma \vdash_{\Downarrow} \mathsf{t} : \mathcal{R}$$

Var-Rho

$$\frac{\mathsf{x} : \mathcal{R}' \in \Gamma \qquad \mathcal{R}' \leq^{\shortparallel}_{p} \mathcal{R}}{\Gamma \vdash \mathsf{x} : \mathcal{R}}$$

Fun

$$\frac{\Gamma, \mathsf{z} : \mathcal{S}_2^{\flat} \vdash \mathsf{t} : \mathcal{S}_1^{\flat}}{\Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \mathcal{S}_2 \to \mathcal{S}_1}$$

App-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil \to \mathcal{S} \qquad \Gamma \vdash \mathsf{t}_2 : \lceil \sigma \rceil^{\flat}}{\Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma) : \mathcal{S}^{\flat}}$$

Let-Gen-Rho

$$\frac{\Gamma \vdash \mathsf{t}_1 : \lceil \sigma \rceil^{\flat} \qquad \Gamma, \mathsf{z} : \lceil \sigma \rceil^{\flat} \vdash \mathsf{t}_2 : \mathcal{R}}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}_2 : \mathcal{R}}$$

Propriétés de la vérification des formes

Les programmes bien typés sont bien formés

Si $\Gamma \vdash \mathsf{t} : \sigma$ alors $\lceil \Gamma \rceil \vdash_{\Downarrow} \mathsf{t} : \lceil \sigma \rceil$.

Seule, la forme des annotations importe

Si $\Gamma \vdash \mathsf{t} : \sigma$ alors $\Gamma \vdash \lfloor \lceil \mathsf{t} \rceil \rfloor : \sigma$.

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 annoations...

$$\mathsf{t} ::= \mathsf{x} \mid \mathsf{fun}\ (\mathsf{z} : \theta)\ \mathsf{t} \mid \mathsf{t}_1\ \mathsf{t}_2 \mid \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2$$

$$\Gamma \vdash_{\Uparrow} \mathsf{t} : \mathcal{R}$$

Var-I

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

App-I

$$\frac{\Gamma \vdash_{\Uparrow} \mathsf{t}_1 : \mathcal{S}_2 \to \mathcal{S}_1 \qquad \Gamma \vdash_{\Uparrow} \mathsf{t}_2 : \mathcal{R}_2 \qquad \mathcal{R}_2 \leq_p^{\shortparallel} \mathcal{S}_2^\flat}{\Gamma \vdash_{\Uparrow} \mathsf{t}_1\ \mathsf{t}_2 : \mathcal{S}_1^\flat}$$

Let-I

$$\frac{\Gamma \vdash_{\Uparrow} \mathsf{t}_1 : \mathcal{R}_1 \qquad \Gamma, \mathsf{z} : \mathcal{R}_1 \vdash_{\Uparrow} \mathsf{t}_2 : \mathcal{R}_2}{\Gamma \vdash_{\Uparrow} \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathcal{R}_2}$$

Fun-I

$$\frac{\Gamma, \mathsf{z} : \lceil \sigma \rceil^\flat \vdash_{\Uparrow} \mathsf{t} : \mathcal{R}}{\Gamma \vdash_{\Uparrow} \mathsf{fun}\ (\mathsf{z} : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{t} : \lceil \sigma \rceil \to \mathcal{R}}$$

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

Var-I

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

App-I

$$\frac{\Gamma \vdash_{\Uparrow} \mathsf{t}_1 : \mathcal{S}_2 \to \mathcal{S}_1 \Rightarrow \mathsf{t}'_1 \qquad \Gamma \vdash_{\Uparrow} \mathsf{t}_2 : \mathcal{R}_2 \Rightarrow \mathsf{t}'_2 \qquad \mathcal{R}_2 \leq^{\shortparallel}_p \mathcal{S}_2^{\flat}}{\Gamma \vdash_{\Uparrow} \mathsf{t}_1\ \mathsf{t}_2 : \mathcal{S}_1^{\flat} \Rightarrow \mathsf{t}'_1\ ((\mathsf{t}'_2 : \lfloor \mathcal{R}_2 \rfloor) : \lfloor \mathcal{S}_2 \rfloor)}$$

Let-I

$$\frac{\Gamma \vdash_{\Uparrow} \mathsf{t}_1 : \mathcal{R}_1 \Rightarrow \mathsf{t}'_1 \qquad \Gamma, \mathsf{z} : \mathcal{R}_1 \vdash_{\Uparrow} \mathsf{t}_2 : \mathcal{R}_2 \Rightarrow \mathsf{t}'_2}{\Gamma \vdash_{\Uparrow} \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \mathcal{R}_2 \Rightarrow \mathsf{let}\ \mathsf{z} = (\mathsf{t}'_1 : \lfloor \mathcal{R}_1 \rfloor)\ \mathsf{in}\ \mathsf{t}'_2}$$

Fun-I

$$\frac{\Gamma, \mathsf{z} : \lceil \sigma \rceil^{\flat} \vdash_{\Uparrow} \mathsf{t} : \mathcal{R} \Rightarrow \mathsf{t}'}{\Gamma \vdash_{\Uparrow} \mathsf{fun}\ (\mathsf{z} : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{t} : \lceil \sigma \rceil \to \mathcal{R} \Rightarrow \mathsf{fun}\ (\mathsf{z})\ \mathsf{let}\ \mathsf{z} = (\mathsf{z} : \exists\, \bar{\beta}.\, \sigma)\ \mathsf{in}\ \mathsf{t}'}$$

Typage dans $\mathrm{F}_p^{\Uparrow}$ par élaboration dans $\mathrm{F}_p^{\Downarrow}$

Definir

$$\Gamma \vdash_{\Uparrow} \mathsf{t} : \sigma$$

comme

$$\exists \mathcal{R},\ (\lceil \Gamma \rceil^{\flat} \vdash_{\Uparrow} \mathsf{t} : \mathcal{R} \Rightarrow \mathsf{t}' \ \ \wedge \ \ \mathcal{R} = \sigma^{\flat} \ \ \wedge \ \ \Gamma \vdash_{\Downarrow} \mathsf{t} : \sigma \Rightarrow \mathsf{t}')$$

Spécification directe de $\mathsf{F}^{\Uparrow}_{p}$

Var-Inst

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

Fun-Gen

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

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2 \to \forall\, \bar{\alpha}.\, \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \rho_2 \qquad \langle \Gamma \rangle(\rho_2) \leq^{\shortparallel}_{p} \sigma_2}{\Gamma \vdash \mathsf{t}_1\ \mathsf{t}_2 : \rho_1[\bar{\tau}/\bar{\alpha}]}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \rho_1 \qquad \Gamma, \mathsf{z} : \langle \Gamma \rangle(\rho_1) \vdash \mathsf{t}_2 : \rho_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \rho_2}$$

Var-Inst

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

Fun-Gen

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

App

$$\frac{\Gamma \vdash \mathsf{t}_1 : \sigma_2 \to \forall\, \bar{\alpha}.\, \rho_1 \qquad \Gamma \vdash \mathsf{t}_2 : \rho_2 \qquad \langle\Gamma\rangle(\rho_2) \leq_p^{\shortparallel} \sigma_2}{\Gamma \vdash \mathsf{t}_1\ \mathsf{t}_2 : \rho_1[\bar{\tau}/\bar{\alpha}]}$$

Let-Gen

$$\frac{\Gamma \vdash \mathsf{t}_1 : \rho_1 \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\rho_1) \vdash \mathsf{t}_2 : \rho_2}{\Gamma \vdash \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \rho_2}$$

Remarque

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

Dans $\mathrm{F}_p^{\Downarrow}$, avec la vérification des formes seulement :

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

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

Dans $\mathrm{F}_p^{\Uparrow}$, avec la propagation des formes seulement :

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

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

Peut-on mélanger les deux approches ?

Propagation bidirectionnelle dans $F^{\Updownarrow}_{p}$

Var-C

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

Var-I

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

App-C

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \to \mathcal{S}_1 \qquad \Gamma \vdash_! t_2 : \mathcal{S}_2^\flat \qquad \mathcal{S}_1^\flat \leq_p^{\shortparallel} \mathcal{R}_1}{\Gamma \vdash_! t_1\ t_2 : \mathcal{R}_1}$$

App-I

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \to \mathcal{S}_1 \qquad \Gamma \vdash_! t_2 : \mathcal{S}_2^\flat}{\Gamma \vdash_? t_1\ t_2 : \mathcal{S}_1^\flat}$$

Let-C

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

Fun-Ce

$$\frac{\Gamma, z : \lceil \sigma \rceil^\flat \vdash_! t : \mathcal{S}_1^\flat \qquad \mathcal{S}_2^\flat \leq_p^{\shortparallel} \lceil \sigma \rceil^\flat}{\Gamma \vdash_! \mathsf{fun}\ (z : \exists\, \beta.\, \sigma)\ t : \mathcal{S}_2 \to \mathcal{S}_1}$$

Fun-Ie

$$\frac{\Gamma, z : \lceil \sigma \rceil^\flat \vdash_? t : \mathcal{R}}{\Gamma \vdash_? \mathsf{fun}\ (z : \exists\, \bar{\beta}.\, \sigma)\ t : \lceil \sigma \rceil \to \mathcal{R}}$$

Fun-Ci

$$\frac{\Gamma, z : \mathcal{S}_2^\flat \vdash_! t : \mathcal{S}_1^\flat}{\Gamma \vdash_! \mathsf{fun}\ (z)\ t : \mathcal{S}_2 \to \mathcal{S}_1}$$

Fun-Ii

$$\frac{\Gamma, z : \sharp \vdash_? t : \mathcal{R}}{\Gamma \vdash_? \mathsf{fun}\ (z)\ t : \sharp \to \mathcal{R}}$$

Var-C

Var-I

App-C

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \to \mathcal{S}_1 \qquad \Gamma \vdash_! t_2 : \mathcal{S}_2^\flat \qquad \mathcal{S}_1^\flat \leq^{\shortparallel}_p \mathcal{R}_1}{\Gamma \vdash_! t_1\ t_2 : \mathcal{R}_1}$$

App-I

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \to \mathcal{S}_1 \qquad \Gamma \vdash_! t_2 : \mathcal{S}_2^\flat}{\Gamma \vdash_? t_1\ t_2 : \mathcal{S}_1^\flat}$$

Let-C

Fun-Ce

$$\frac{\Gamma, z : \lceil\sigma\rceil^\flat \vdash_! t : \mathcal{S}_1^\flat \qquad \mathcal{S}_2^\flat \leq^{\shortparallel}_p \lceil\sigma\rceil^\flat}{\Gamma \vdash_! \mathsf{fun}\ (z : \exists\, \beta.\, \sigma)\ t : \mathcal{S}_2 \to \mathcal{S}_1}$$

Fun-Ie

Fun-Ci

Fun-Ii

$$\frac{\Gamma, z : \sharp \vdash_? t : \mathcal{R}}{\Gamma \vdash_? \mathsf{fun}\ (z)\ t : \sharp \to \mathcal{R}}$$

Propagation bidirectionnelle dans $F^{\Updownarrow}_{p}$

Var-C

Var-I

App-C

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \to \mathcal{S}_1 \Rightarrow t'_1 \quad \Gamma \vdash_! t_2 : \mathcal{S}_2^\flat \Rightarrow t'_2 \qquad \mathcal{S}_1^\flat \leq^{\shortparallel}_p \mathcal{R}_1}{\Gamma \vdash_! t_1\ t_2 : \mathcal{R}_1 \Rightarrow t_1\ (t_2 : \lfloor \mathcal{S}_2 \rfloor)}$$

App-I

$$\frac{\Gamma \vdash_? t_1 : \mathcal{S}_2 \to \mathcal{S}_1 \Rightarrow t'_1 \quad \Gamma \vdash_! t_2 : \mathcal{S}_2^\flat \Rightarrow t'_2}{\Gamma \vdash_? t_1\ t_2 : \mathcal{S}_1^\flat \Rightarrow t_1\ (t_2 : \lfloor \mathcal{S}_2 \rfloor)}$$

Let-C

Fun-Ce

$$\frac{\Gamma, z : \lceil \sigma \rceil^\flat \vdash_! t : \mathcal{S}_1^\flat \Rightarrow t' \qquad \mathcal{S}_2^\flat \leq^{\shortparallel}_p \lceil \sigma \rceil^\flat}{\Gamma \vdash_! \mathsf{fun}\ (z : \exists \beta.\, \sigma)\ t : \mathcal{S}_2 \to \mathcal{S}_1 \Rightarrow \mathsf{fun}\ (z)\ \mathsf{let}\ z = (z : \exists \beta.\, \sigma)\ \mathsf{in}\ t'}$$

Fun-Ie

Fun-Ci

Fun-Ii

$$\frac{\Gamma, z : \sharp \vdash_? t : \mathcal{R} \Rightarrow t'}{\Gamma \vdash_? \mathsf{fun}\ (z)\ t : \sharp \to \mathcal{R} \Rightarrow \mathsf{fun}\ (z)\ t'}$$

Présentation directe $F^{\Updownarrow}_{p}$

Var-C

$$\frac{\sigma' \in \Gamma \qquad \sigma' \leq^{\shortparallel}_{p} \rho}{\Gamma \vdash_{!} \mathsf{z} : \rho}$$

Var-I

$$\frac{\mathsf{z} : \sigma \in \Gamma \qquad \sigma \leq_{p} \rho}{\Gamma \vdash_{?} \mathsf{z} : \rho}$$

App-C

$$\frac{\Gamma \vdash_{?} \mathsf{t}_1 : \sigma_2 \to \sigma_1 \qquad \Gamma \vdash_{!} \mathsf{t}_2 : \sigma_2 \qquad \langle\Gamma\rangle(\sigma_1) \leq^{\shortparallel}_{p} \rho_1}{\Gamma \vdash_{!} \mathsf{t}_1\ \mathsf{t}_2 : \rho_1}$$

App-I

$$\frac{\Gamma \vdash_{?} \mathsf{t}_1 : \sigma_2 \to \rho_1 \qquad \Gamma \vdash_{!} \mathsf{t}_2 : \sigma_2}{\Gamma \vdash_{?} \mathsf{t}_1\ \mathsf{t}_2 : \rho_1}$$

Let-C

$$\frac{\Gamma \vdash_{?} \mathsf{t}_1 : \rho_1 \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\rho_1) \vdash_{!} \mathsf{t}_2 : \sigma_2}{\Gamma \vdash_{!} \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \sigma_2}$$

Let-I

$$\frac{\Gamma \vdash_{?} \mathsf{t}_1 : \rho_1 \qquad \Gamma, \mathsf{z} : \langle\Gamma\rangle(\rho_1) \vdash_{?} \mathsf{t}_2 : \rho_2}{\Gamma \vdash_{?} \mathsf{let}\ \mathsf{z} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2 : \rho_2}$$

Fun-Ci

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

Fun-Ie

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

Fun-Ce

$$\frac{\Gamma, \mathsf{z} : \sigma[\bar{\tau}/\bar{\beta}] \vdash_{!} \mathsf{t} : \sigma_1 \qquad \sigma_2 \leq^{\shortparallel}_{p} \sigma[\bar{\tau}/\bar{\beta}] \qquad \bar{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash_{!} \mathsf{fun}\ (\mathsf{z} : \exists\, \beta.\, \sigma)\ \mathsf{t} : \forall\, \bar{\alpha}.\, \sigma_2 \to \sigma_1}$$

Fun-Ii

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

Présentation directe $F^{\Updownarrow}_{p}$

Var-C

Var-I

App-C

$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \to \sigma_1 \qquad \Gamma \vdash_! t_2 : \sigma_2 \qquad \langle\Gamma\rangle(\sigma_1) \leq^{\shortparallel}_p \rho_1}{\Gamma \vdash_! t_1\ t_2 : \rho_1}$$

App-I

$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \to \rho_1 \qquad \Gamma \vdash_! t_2 : \sigma_2}{\Gamma \vdash_? t_1\ t_2 : \rho_1}$$

Let-C

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

Let-I

Fun-Ci

Fun-Ie

Fun-Ce

$$\frac{\Gamma, z : \sigma[\bar\tau/\bar\beta] \vdash_! t : \sigma_1 \qquad \sigma_2 \leq^{\shortparallel}_p \sigma[\bar\tau/\bar\beta] \qquad \bar\alpha \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash_! \mathsf{fun}\ (z : \exists\,\beta.\,\sigma)\ t : \forall\,\bar\alpha.\,\sigma_2 \to \sigma_1}$$

Fun-Ii

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

Présentation directe $F^{\Updownarrow}_{p}$

Var-C

Var-I

App-C

$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \to \sigma_1 \qquad \Gamma \vdash_! t_2 : \sigma_2 \qquad \langle\Gamma\rangle(\sigma_1) \leq^{\shortparallel}_p \rho_1}{\Gamma \vdash_! t_1\ t_2 : \rho_1}$$

App-I

$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \to \rho_1 \qquad \Gamma \vdash_! t_2 : \sigma_2}{\Gamma \vdash_? t_1\ t_2 : \rho_1}$$

Let-C

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

Let-I

Fun-Ci

Fun-Ie

Fun-Ce

$$\frac{\Gamma, z : \sigma[\bar\tau/\bar\beta] \vdash_! t : \sigma_1 \qquad \sigma_2 \leq^{\shortparallel}_p \sigma[\bar\tau/\bar\beta] \qquad \bar\alpha \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash_! \mathsf{fun}\ (z : \exists\,\beta.\,\sigma)\ t : \forall\,\bar\alpha.\,\sigma_2 \to \sigma_1}$$

Fun-Ii

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

Sucre syntaxique :

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

Var-C Var-I

App-C
$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \to \sigma_1 \qquad \Gamma \vdash_! t_2 : \sigma_2 \qquad \langle\Gamma\rangle(\sigma_1) \leq^{\shortparallel}_p \rho_1}{\Gamma \vdash_! t_1\ t_2 : \rho_1}$$

App-I
$$\frac{\Gamma \vdash_? t_1 : \sigma_2 \to \rho_1 \qquad \Gamma \vdash_! t_2 : \sigma_2}{\Gamma \vdash_? t_1\ t_2 : \rho_1}$$

Let-C
$$\frac{\Gamma \vdash_? t_1 : \rho_1 \qquad \Gamma, z : \langle\Gamma\rangle(\rho_1) \vdash_! t_2 : \sigma_2}{\Gamma \vdash_! \mathsf{let}\ z = t_1\ \mathsf{in}\ t_2 : \sigma_2}$$

Let-I Fun-Ci Fun-Ie

Fun-Ce
$$\frac{\Gamma, z : \sigma[\bar\tau/\bar\beta] \vdash_! t : \sigma_1 \qquad \sigma_2 \leq^{\shortparallel}_p \sigma[\bar\tau/\bar\beta] \qquad \bar\alpha \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash_! \mathsf{fun}\ (z : \exists\,\beta.\,\sigma)\ t : \forall\,\bar\alpha.\,\sigma_2 \to \sigma_1}$$

Fun-Ii
$$\frac{\Gamma, z : \tau \vdash_? t : \rho}{\Gamma \vdash_? \mathsf{fun}\ (z)\ t : \tau \to \rho}$$

Sucre syntaxique :

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

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 posibles pour la propagation, mais le noyau est partagé.

On retombe dans un cadre formel bien compris

- F^{η} pour la correction.
- Constraintes 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^ω ?
- 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^ω ?
- application de la technique au sous-typage (avec des contraintes de sous-typage de premier ordre) ?

Mais expressivité limité...

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` $: \forall\,\bar{\alpha}.\,\bar{\alpha} \to \bar{\alpha}$`) (open x) (open x))`

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

  `[ fun x x :` $\forall\,\bar{\alpha}.\,\bar{\alpha} \to \bar{\alpha}$ `]`
- Le système garde trace des polytypes qui peuvent être ouverts

  ~~`fun x (open x) (open x)`~~
- ML^F reprend cette approche mais en plus laissse 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` $: \forall\,\bar{\alpha}.\,\bar{\alpha} \to \bar{\alpha}$`) (open x) (open x))`

  `[ fun x x :` $\forall\,\bar{\alpha}.\,\bar{\alpha} \to \bar{\alpha}$ `]`
- Le système garde trace des polytypes qui peuvent être ouverts

  ~~`fun x (open x) (open x)`~~
- ML^F reprend cette approche mais en plus laissse le bleu implicite. Seul le rouge doit être indiqué explicitement.

ML^{F} intuitivement

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

où $id = \forall\, \alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

let $auto = \mathsf{fun}\ (x : id)\ x\ x\ :\ id \to id$

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

ML^{F} intuitivement

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

où $id = \forall\, \alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

let $auto = \mathsf{fun}\ (x : id)\ x\ x\ :\ id \to id$

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

let $choice\ \ x\ y = \mathsf{if}\ true\ \mathsf{then}\ x\ \mathsf{else}\ y\ :\ \forall\, \alpha.\, \alpha \to \alpha \to \alpha$

let $hard = choice\ id$

ML^{F} intuitivement

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

où $id = \forall\,\alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

let $auto = \mathsf{fun}\ (x : id)\ x\ x\ :\ id \to id$

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

let … x else y : $\forall\,\alpha.\, \alpha \to \alpha \to \alpha$

$id_\alpha \to id_\alpha \to id_\alpha$

id_α

let $hard = choice\ id$

: $\forall\,\alpha.(id_\alpha \to id_\alpha)$

Prendre une instance de id

ML^{F} intuitivement

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

où $id = \forall\,\alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

let $auto = \mathsf{fun}\ (x : id)\ x\ x\ :\ id \to id$

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

let $choose = \mathsf{fun}\ (x)\ (y)\ \mathsf{then}\ x\ \mathsf{else}\ y\ :\ \forall\,\alpha.\, \alpha \to \alpha \to \alpha$

$id \to id \to id$

id

let $hard = choice\ id$

$: \forall\,\alpha.(id_\alpha \to id_\alpha)$

$: (\forall\,\alpha.\, id_\alpha) \to (\forall\,\alpha.\, id_\alpha)$ Conserver id polymorphe

ML^{F} intuitivement

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

où $id = \forall\,\alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

let $auto = \mathsf{fun}\ (x : id)\ x\ x\ :\ id \to id$

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

let $choice\ \ x\ y = \mathsf{if}\ true\ \mathsf{then}\ x\ \mathsf{else}\ y\ :\ \forall\,\alpha.\, \alpha \to \alpha \to \alpha$

let $hard = choice\ id$

$: \forall\,\alpha.(id_\alpha \to id_\alpha)$

$: (\forall\,\alpha.\, id_\alpha) \to (\forall\,\alpha.\, id_\alpha)$

} sont incomparables

ML^{F} intuitivement

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

où $id = \forall\, \alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

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

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

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

let $hard = choice\ id$

$: \forall\, \alpha.(id_\alpha \to id_\alpha)$

$: (\forall\, \alpha.\, id_\alpha) \to (\forall\, \alpha.\, id_\alpha)$

$: \forall\, (\beta \geq id)\ \beta \to \beta$

$\beta \to \beta$

pour tout β qui est instance de id

ML^{F} intuitivement

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

où $id = \forall\,\alpha.\, id_\alpha$

et $id_\alpha = \alpha \to \alpha$

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

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

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

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

devrait s'écrire

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

let $hard = choice\ id$

$: \forall\,\alpha.(id_\alpha \to id_\alpha)$

$: (\forall\,\alpha.\, id_\alpha) \to (\forall\,\alpha.\, id_\alpha)$

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

Exemples

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

Exemples

- fun $(x : id)\ x\ x : \forall\ (\beta \exists id, \beta' \geq id)\ \beta \to \beta'$
  - $\forall\ (\beta \exists id)$ indique que le polymorphisme id de l'argument a pu être utilisé et ne permet pas de passer une instance de id.

Exemples

- fun $(x : id)\ x\ x : \forall\ (\beta \exists id, \beta' \geq id)\ \beta \to \beta'$
  - $\forall\ (\beta \exists id)$ indique que le polymorphisme id de l'argument a pu être utilisé et ne permet pas de passer une instance de id.
- $choice\ id : \forall\ (\beta \geq id)\ \beta \to \beta$
  - 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.

Exemples

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

Exemples

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

Exemples

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

Typing rules for ML

Var

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

Fun

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

App

$$\frac{\Gamma \vdash a_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash a_2 : \tau_2}{\Gamma \vdash a_1\ a_2 : \tau_1}$$

Let

$$\frac{\Gamma \vdash a_1 : \sigma \qquad \Gamma, x : \sigma \vdash a_2 : \tau}{\Gamma \vdash \mathsf{let}\ x = a_1 = a\ \mathsf{in}\ {}_2 : \tau}$$

Gen

$$\frac{\alpha \notin \mathsf{ftv}(\Gamma) \qquad \Gamma \vdash a : \sigma'}{\Gamma \vdash a : \forall\, \alpha \qquad .\, \sigma'}$$

Inst

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

Typing rules for ML$^{\mathsf{F}}$

Var

$$\frac{x : \sigma \in \Gamma}{\forall\,(Q)\ \Gamma \vdash x : \sigma}$$

Fun

$$\frac{\forall\,(Q)\ \Gamma, x : \tau_0 \vdash a : \tau}{\forall\,(Q)\ \Gamma \vdash \mathsf{fun}\ (x)\ a : \tau_0 \to \tau}$$

App

$$\frac{\forall\,(Q)\ \Gamma \vdash a_1 : \tau_2 \to \tau_1 \qquad \forall\,(Q)\ \Gamma \vdash a_2 : \tau_2}{\forall\,(Q)\ \Gamma \vdash a_1\ a_2 : \tau_1}$$

Let

$$\frac{\forall\,(Q)\ \Gamma \vdash a_1 : \sigma \qquad \forall\,(Q)\ \Gamma, x : \sigma \vdash a_2 : \tau}{\forall\,(Q)\ \Gamma \vdash \mathsf{let}\ x = a_1 = a\ \mathsf{in}\ {}_2 : \tau}$$

Gen

$$\frac{\alpha \notin \mathsf{ftv}(\Gamma) \qquad \forall\,(Q, \alpha \diamond \sigma)\ \Gamma \vdash a : \sigma'}{\forall\,(Q)\ \Gamma \vdash a : \forall\,\alpha \diamond \sigma.\, \sigma'}$$

Inst

$$\frac{\forall\,(Q)\ \Gamma \vdash a : \sigma \qquad \forall\,(Q)\ \sigma \leq \sigma'}{\forall\,(Q)\ \Gamma \vdash a : \sigma'}$$

Typing rules for ML$^{\mathsf{F}}$

Var

$$\frac{x : \sigma \in \Gamma}{\forall\,(Q)\ \Gamma \vdash x : \sigma}$$

Fun

$$\frac{\forall\,(Q)\ \Gamma, x : \tau_0 \vdash a : \tau}{\forall\,(Q)\ \Gamma \vdash \mathsf{fun}\ (x)\ a : \tau_0 \to \tau}$$

App

$$\frac{\forall\,(Q)\ \Gamma \vdash a_1 : \tau_2 \to \tau_1 \qquad \forall\,(Q)\ \Gamma \vdash a_2 : \tau_2}{\forall\,(Q)\ \Gamma \vdash a_1\ a_2 : \tau_1}$$

Let

$$\frac{\forall\,(Q)\ \Gamma \vdash a_1 : \sigma \qquad \forall\,(Q)\ \Gamma, x : \sigma \vdash a_2 : \tau}{\forall\,(Q)\ \Gamma \vdash \mathsf{let}\ x = a_1 = a\ \mathsf{in}\ {}_2 : \tau}$$

Gen

$$\frac{\alpha \notin \mathsf{ftv}(\Gamma) \qquad \forall\,(Q, \alpha \diamond \sigma)\ \Gamma \vdash a : \sigma'}{\forall\,(Q)\ \Gamma \vdash a : \forall\,\alpha \diamond \sigma.\, \sigma'}$$

Inst

$$\frac{\forall\,(Q)\ \Gamma \vdash a : \sigma \qquad \forall\,(Q)\ \sigma \leq \sigma'}{\forall\,(Q)\ \Gamma \vdash a : \sigma'}$$

Q liste les variables de types libres avec leur bornes.

Les types de ML^F

$$\begin{array}{lcll} \tau & = & \alpha \mid \tau_1 \to \tau_1 & \text{Monotypes} \\ \sigma & = & \tau \mid \bot \mid \forall\,(\alpha \geq \sigma_1)\,\sigma_2 \mid \forall\,(\alpha \exists\, \sigma_1)\,\sigma_2 & \text{Schémas de types} \end{array}$$

Signifie «tous les types»

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

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

  s'écrit :

$$\forall\,(\alpha)\,\forall\,(\beta \exists\, \forall\,(\alpha')\,\tau \to \alpha)\,\beta \to \alpha$$

Examples

$id \triangleq \forall\, \alpha.\, \alpha \to \alpha$

$id_\alpha \triangleq \alpha \to \alpha$

$$
\begin{array}{rcl}
\bot & \leq & \sigma \\
\forall\,(\alpha)\ \alpha \to \alpha & \leq & \alpha \to \alpha \\
\forall\,(\alpha)\ \alpha \to \alpha & \leq & \forall\,(\alpha, \alpha')\ (\alpha \to \alpha') \to (\alpha \to \alpha') \\
\forall\,(\beta \geq id)\ \beta \to \beta & \leq & \forall\,(\beta \exists\, id)\ \beta \to \beta \\
\forall\,(\beta \geq id)\ \beta \to \beta & \leq & \forall\,(\alpha)\ \forall\,(\beta \geq id_\alpha)\ \beta \to \beta \\
 & \leq & \forall\,(\alpha)\ id_\alpha \to id_\alpha
\end{array}
$$

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

$\approx \tau \to \forall\,(\alpha)\ \tau'$

$\leq \quad \forall\,(\alpha)\ \forall\,(\beta \exists\, \tau')\ \tau \to \beta \quad \leq$

En fait

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

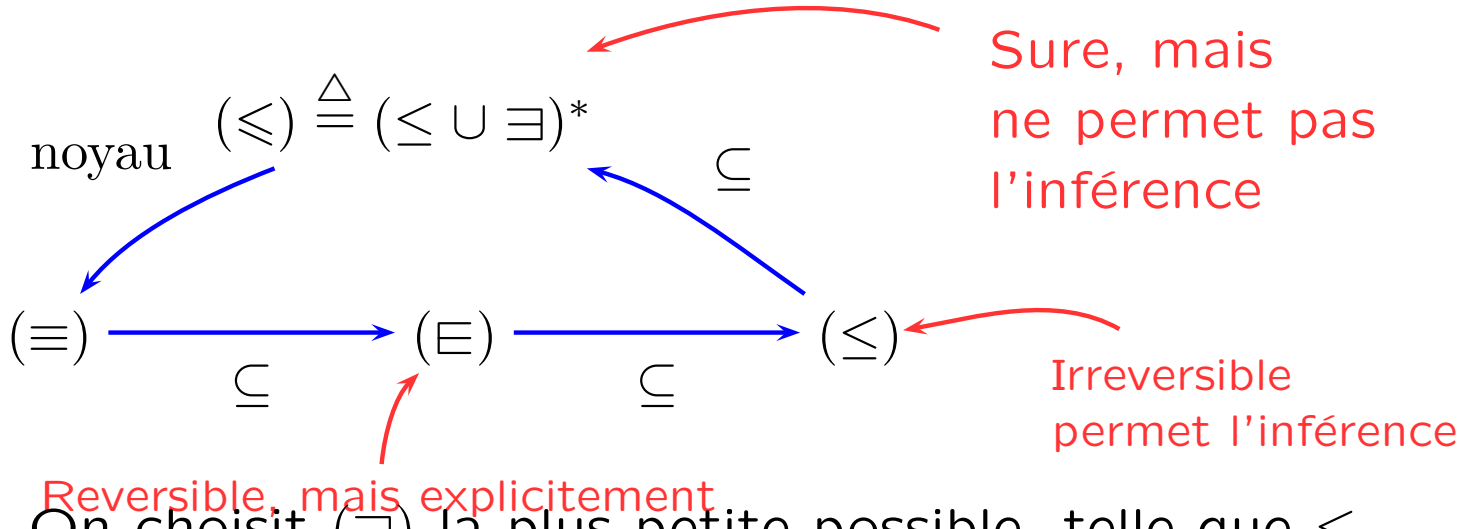
- Instance relation is defined under prefixes

$$\forall\,(Q)\;\sigma \leqslant \sigma'$$

Par des exemples

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

Décomposition de la relation d'instance



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

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

$$(\_ : \sigma) : \forall\ (\alpha \ni \sigma, \alpha' \ni \sigma)\ \alpha \to \alpha'$$

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 λ-calculus are equivalent and undecidable. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 176–185.

Algorithme

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

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

C'est-à-dire, φ' est une substitution la plus générale qui soit une instance de φ et satisfasse le problème de typage (*i.e.* telle que $\varphi'(\Gamma) \vdash \mathsf{t} : \varphi'(\rho)$). *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 $\exists W.(\varphi \wedge \Gamma \vdash \mathsf{t} : \rho) \rightsquigarrow \exists W'.\, \varphi'$. où W est l'ensemble des variables introduites par φ.

Algorithme

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

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

$$\frac{\begin{array}{c}\varphi \wedge \Gamma \vdash \mathsf{t}_1 : \sigma_2 \to \rho_1 \rightsquigarrow \varphi_1 \\ \varphi_1 \wedge \Gamma \vdash \mathsf{t}_2 : \sigma_2 \rightsquigarrow \varphi_2 \qquad \bar{\beta} \text{ fresh}\end{array}}{\varphi \wedge \Gamma \vdash \mathsf{t}_1\ (\mathsf{t}_2 : \exists\, \bar{\beta}.\, \sigma_2) : \rho_1 \rightsquigarrow \varphi_2}$$

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

$$\frac{\begin{array}{c}\varphi(\tau) = \beta' \to \beta \rightsquigarrow \varphi' \qquad \beta\beta' \text{ fresh} \\ \varphi' \wedge \Gamma, \mathsf{z} : \beta' \vdash \mathsf{t} : \beta \rightsquigarrow \varphi''\end{array}}{\varphi \wedge \Gamma \vdash \mathsf{fun}\ (\mathsf{z})\ \mathsf{t} : \tau \rightsquigarrow \varphi''}$$

$$\frac{\begin{array}{c}\varphi \wedge \Gamma \vdash \mathsf{t}_1 : \rho_1 \rightsquigarrow \varphi_1 \qquad \bar{\beta} \text{ fresh} \\ \varphi_1 \wedge \Gamma, \mathsf{z} : \langle\Gamma\rangle(\varphi_1(\sigma_1)) \vdash \mathsf{t}_2 : \forall\, \bar{\alpha}.\, \rho_2 \rightsquigarrow \varphi_2\end{array}}{\varphi_2 \wedge \Gamma \vdash \mathsf{let}\ \mathsf{z} = (\mathsf{t}_1 : \exists\, \bar{\beta}.\, \sigma_1)\ \mathsf{in}\ \mathsf{t}_2 : \rho_1}$$

Algorithme $(\varphi \wedge \Gamma \vdash \mathrm{t} : \rho) \rightsquigarrow \varphi'$

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