

# Récursion généralisée et inférence de types avec intersection

*Séminaire – 13 mai 2004*

Pascal Zimmer

INRIA Sophia Antipolis

# Motivation générale

Concevoir une base de langage:

- cœur fonctionnel
- couche objets
- sémantique bien définie, implémentable de façon réaliste
- inférence de types principaux à la ML

dans le but de rajouter d'autres paradigmes (migration, réactif)...

# Plan

Première partie:

- sémantique des langages à objets
- typage avec degrés
- implémentation, machine abstraite
- mixins

# Plan

Première partie:

- sémantique des langages à objets
- typage avec degrés
- implémentation, machine abstraite
- mixins

Seconde partie:

- types avec intersection
- calcul de Klop
- inférence de types
- extensions

# Première partie

## Réursion généralisée

# Sémantique des objets 1

## Sémantique par auto-application

- modèle initié par Kamin, 1988;  
référence: Abadi et Cardelli, 1996
- objet = collection de pré-méthodes:

$$o = [\dots, l = \zeta(\text{self}) b, \dots]$$

- appel de méthode:

$$o.l \Rightarrow b \{\text{self} \leftarrow o\}$$

- typage spécifique
- inférence de types principaux impossible

# Sémantique des objets 2

## Sémantique par enregistrements récurifs

- Cardelli 1988, Wand 1994, Cook 1994
- classe:

$$C = \lambda x_1 \dots \lambda x_n \lambda \text{self} \{l_1 = M_1, \dots, l_p = M_p\}$$

- objet:  $o = \text{fix} (CN_1 \dots N_n)$
- variables de rangée pour étendre l'objet
- pas de modification de l'état, car self lié à l'objet de départ
- modèle de typage d'OCAML

# Proposition de langage

- sémantique par enregistrements récurrents de Wand
- références à la ML pour stocker l'état de l'objet
- exemples:

$$\begin{aligned} point &= \lambda x \lambda self \\ &\quad \{ pos = \text{ref } x, \\ &\quad \quad move = \lambda y (self.pos := !self.pos + y) \} \end{aligned}$$
$$p = \text{fix } (point \ 4)$$
$$\begin{aligned} color\_point &= \lambda x \lambda c \lambda self \\ &\quad \{ point \ x \ self, \ color = \text{ref } c \} \end{aligned}$$

# Evaluation du point fixe

- Problème: comment évaluer le point fixe ?

$$\text{fix} = \lambda f (\text{let rec } x = fx \text{ in } x)$$

- En SML, seule construction autorisée:

$$\text{let rec } x = \lambda y N \text{ in } M$$

- Il nous faut un opérateur de récursion généralisée
- Mais certaines récursions sont dangereuses:

$$\text{let rec } x = xV \text{ in } M$$

$$\text{let rec } x = x + 1 \text{ in } M$$

# Systeme de types avec degrades

- Boudol, 2001
- degre = information booléenne dans le type des fonctions et dans le contexte de typage

$$\theta^d \rightarrow \tau$$

- 0 = “dangereux”, 1 = “sûr”
- intuitivement: valeur requise ou non lors de l'évaluation
- (let rec  $x = N$  in  $M$ ) est typable ssi  $N$  est typable avec un degre 1 pour  $x$
- (let rec  $x = fx$  in  $M$ ) est typable ssi  $f$  a pour type  $\theta^1 \rightarrow \tau$  (fonction dite “protectrice”)

# Degrés - exemples

- c'est le cas de:

$$point0 = \lambda self$$
$$\{ pos = \text{ref } 0,$$
$$move = \lambda y(\text{self}.pos := !\text{self}.pos + y) \}$$

- $\text{fix} = \lambda f(\text{let rec } x = fx \text{ in } x)$

a pour type:  $(\tau^1 \rightarrow \tau)^0 \rightarrow \tau$

- $\lambda self \{ x = 0, y = \text{self}.x \}$

a pour type:  $\{ \rho, x : \tau \}^0 \rightarrow \{ x : int, y : \tau \}$

où  $\rho$  est une variable de rangée

avec la contrainte  $\rho :: \{ x \}$

# Degrés - résultats

- réduction du sujet
- sûreté: l'évaluation d'un terme typable ne conduit jamais à une erreur (récursion, accès aux champs, applications...)
- algorithme d'inférence de types principaux, extension de celui de ML

# Algorithmes d'unification et d'inférence

- versions plus “réalistes” et efficaces
- travaillent sur des graphes (types récurifs)
- unification de degrés, enregistrements, types
- polymorphisme similaire à ML, sur variables de degré, rangée ou de type; généralisation pour:

$\text{let (rec) } x = V \text{ in } M$

- contraintes sur variables de rangée ( $\rho :: L$ ) et de degré;

exemple:  $\lambda f \lambda x (f x)$  a pour type

$(\theta^\alpha \rightarrow \tau)^\beta \rightarrow \theta^\gamma \rightarrow \tau$  avec  $\gamma \leq \alpha$

# Machine abstraite

- il faut pouvoir évaluer des termes de la forme

$$(\lambda \text{self } M) o$$

où  $o$  est une variable non encore évaluée, sachant que la valeur de  $\text{self}$  ne sera pas requise pour évaluer  $M$

- machines habituelles pour le  $\lambda$ -calcul ou ML ne permettent pas l'évaluation de la récursion généralisée

# Machine abstraite

$$\mathcal{M} = (S, \sigma, M, \xi)$$

- $S$ : pile de contrôle
- $\sigma$ : environnement
- $M$ : terme à évaluer
- $\xi$ : mémoire pour les valeurs récursives (et les références)
- ensemble de 11 règles de transition, dont la règle “magique”:

$$\begin{aligned} & (S :: (\sigma \lambda y M []), \rho :: \{x \mapsto \ell\}, x, \xi) \\ \rightarrow & (S, \sigma :: \{y \mapsto \ell\}, M, \xi) \quad \text{si } \xi(\ell) = \bullet \end{aligned}$$

# Machine abstraite

- correspondance opérationnelle
- déterminisme
- pas de réductions “silencieuses” infinies
- correction:  
si le terme de départ est typable, alors la machine et la sémantique du calcul passent par les mêmes réductions

# MLOBJ

<http://www-sop.inria.fr/mimosa/Pascal.Zimmer/mlobj.html>

Interpréteur dans le style d'OCAML...

# Mixins

- but: passer à l'ordre supérieur pour obtenir des constructions objets plus puissantes
- générateur:  $\lambda s \{ \dots \}$
- mixin: transformateur de générateur

$$C = \lambda x_1 \dots \lambda x_n \lambda g \lambda s \{ \dots \text{champs} \dots \text{méthodes} \dots \}$$

- instance ( $\lambda s \{ \}$  est le générateur de départ):

$$\text{fix } (C N_1 \dots N_n (\lambda s \{ \}))$$

- opérateur new:

$$\text{new} = \lambda m \text{fix } (m (\lambda s \{ \}))$$

# Mixins - définition

Implémentés par du sucre syntaxique.

## **mixin**

**var**  $l = N$

donnée modifiable

**cst**  $l = N$

donnée constante

**meth**  $l(\text{super}, \text{self}) = N$

méthode

**meth**  $l(\text{super}, \text{self}) \leftarrow N$

redéfinition de méthode

**inherit**  $N$

héritage

**without**  $l$

suppression de champ

**rename**  $l$  **as**  $l'$

renommage de champ

**end**

Appel de méthode:  $M\#l$

# Mixins - exemples

*point* =  $\lambda x$

**mixin**

**var** *pos* = *x*

**meth** *move* ...

**end**

*coloring* =  $\lambda c$

**mixin**

**var** *color* = *c*

**meth** *paint* ...

**end**

*colorPoint* =  $\lambda x \lambda c$

**mixin**

**inherit** *point* *x*

**inherit** *coloring* *c*

**end**

⇒ héritage multiple

# Mixins - exemples

*reset* =

**mixin**

**meth** *reset*(super, self) = self.*pos* := 0

**end**

*resetPoint* =  $\lambda x$

**mixin**

**inherit** *point* *x*

**inherit** *reset*

**end**

*resetColorPoint* =  $\lambda x \lambda c$

**mixin**

**inherit** *colorPoint* *x* *c*

**inherit** *reset*

**end**

⇒ partage de code

# Mixins - exemples

**mixin**

```
meth reset(super, self) ←  
     $\lambda d$  (super#reset; super#paint d)
```

**end**

- Le typage permet de distinguer quels mixins sont instantiables et lesquels ne le sont pas.
- En changeant le générateur de départ, on peut obtenir les initialiseurs.
- Mixins = valeurs de premier ordre  
⇒ une grande puissance expressive  
encore à explorer !

# Et ensuite ?

- fonctionnalités avancées: clonage, méthodes binaires... :

**meth**  $eq(\text{super}, \text{self}) = \lambda p (\text{self}.pos == p.pos)$

- opérationnellement, pas de problème
- typage: pas assez de polymorphisme !
- Système F ?  
inférence de types indécidable...
- types avec intersection ?  
inférence décidable à rang fini...

# Deuxième partie

## Inférence de types avec intersection

# Historique

- système  $\mathcal{D}$ : Coppo, Dezani, 1980; Pottinger, 1980
- typage principal: Coppo, Dezani et Venneri, 1980; Ronchi della Rocca et Venneri, 1984
- inférence: Ronchi della Rocca, 1988
- système  $\mathbb{I}$ : Kfoury et Wells, 1999
- système E: Carlier, Kfoury, Polakow et Wells, 2004

Motivation:

trouver un algorithme plus simple à exposer et à prouver

# Syntaxe des types

$$\tau, \sigma ::= t \mid \tau_1, \dots, \tau_n \rightarrow \sigma$$

- conjonction seulement à gauche de la flèche
- séquence vide notée  $\omega$
- $\tau_1, \dots, \tau_n \rightarrow \sigma$ : type d'une fonction qui attend un argument possédant *tous* les types  $\tau_i$

# Règles de typage

$$\frac{}{x : \tau \vdash x : \tau} \text{(Typ Id)}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma \setminus x \vdash \lambda x M : \Gamma(x) \rightarrow \tau} \text{(Typ } \lambda)$$

$$\frac{\Gamma \vdash M : \tau_1, \dots, \tau_n \rightarrow \sigma \quad \forall i, \Gamma_i \vdash N : \tau_i}{\Gamma, \Gamma_1, \dots, \Gamma_n \vdash MN : \sigma} \text{(Typ Appl Gen)} \quad (n \geq 1)$$

$$\frac{\Gamma \vdash M : \omega \rightarrow \sigma \quad \Gamma_1 \vdash N : \tau_1}{\Gamma, \Gamma_1 \vdash MN : \sigma} \text{(Typ Appl } \omega)$$

# Exemples

- $\vdash I : t \rightarrow t$   
( $I = \lambda x x$ )
- $\vdash \mathbf{2} : (t_1 \rightarrow t_2), (t_2 \rightarrow t_3) \rightarrow t_1 \rightarrow t_3$   
( $\mathbf{2} = \lambda f \lambda x f(fx)$ )
- $\vdash \Delta : t_1, (t_1 \rightarrow t_2) \rightarrow t_2$   
( $\Delta = \lambda x (xx)$ )
- $\vdash K : t \rightarrow \omega \rightarrow t$   
( $K = \lambda x \lambda y x$ )
- $\not\vdash \Omega : ?$                        $\not\vdash Kx\Omega : ?$   
( $\Omega = \Delta\Delta$ )

# Propriétés

- Réduction du sujet: Si  $M \rightarrow M'$ , alors

$$\Gamma \vdash M : \tau \implies \Gamma \vdash M' : \tau$$

- Théorème: Un terme  $M$  est typable dans  $\mathcal{D}$  si et seulement si  $M$  est fortement normalisable (i.e. ssi il ne possède pas de réduction infinie).

# Propriétés

- Réduction du sujet: Si  $M \rightarrow M'$ , alors

$$\Gamma \vdash M : \tau \implies \Gamma \vdash M' : \tau$$

- Théorème: Un terme  $M$  est typable dans  $\mathcal{D}$  si et seulement si  $M$  est fortement normalisable (i.e. ssi il ne possède pas de réduction infinie).
- Algorithme: essayer de normaliser fortement, puis typer.
- Problème: ne marche plus pour un calcul étendu (récursion...)
- On a le type, mais pas l'arbre de typage...

# Exemple

$$M = F(\lambda u \Delta(uu))$$

avec  $F = \lambda x \lambda y y$  et  $\Delta = \lambda x (xx)$

# Exemple

$$M = F(\lambda u \Delta(uu))$$

avec  $F = \lambda x \lambda y y$  et  $\Delta = \lambda x (xx)$

- Première étape:  
annoter chaque variable et chaque application par  
une variable de type fraîche.

$$M^t = (F^t (\lambda u (\Delta^t (u : t_4 u : t_5) : t_6) : t_7)) : t_8$$

avec  $F^t = \lambda x \lambda y (y : t_0)$

et  $\Delta^t = \lambda x (x : t_1 x : t_2) : t_3$

# Exemple - $F(\lambda u \Delta(uu))$

- Deuxième étape:  
pour chaque application  $(M^t N^t) : t$ , créer la  
contrainte:

$$\text{Typ}(N^t) \rightarrow t \perp \text{Typ}(M^t) [ftv(N^t)]$$

# Exemple - $F(\lambda u \Delta(uu))$

- Deuxième étape:  
pour chaque application  $(M^t N^t) : t$ , créer la  
contrainte:

$$Typ(N^t) \rightarrow t \perp Typ(M^t) [ftv(N^t)]$$

$$\left\{ \begin{array}{ll} (t_4, t_5 \rightarrow t_7) \rightarrow t_8 & \perp \quad \omega \rightarrow t_0 \rightarrow t_0 & [t_1, \dots, t_7], \\ t_6 \rightarrow t_7 & \perp \quad t_1, t_2 \rightarrow t_3 & [t_4, t_5, t_6], \\ t_5 \rightarrow t_6 & \perp \quad t_4 & [t_5], \\ t_2 \rightarrow t_3 & \perp \quad t_1 & [t_2] \end{array} \right\}$$

(En ML, on aurait ajouté  $t_4 \perp t_5$  et  $t_1 \perp t_2$ ).

# Exemple - $F(\lambda u \Delta(uu))$

Résolution de:

$$t_6 \rightarrow t_7 \perp t_1, t_2 \rightarrow t_3 [t_4, t_5, t_6]$$

Nouveau système:

$$\left\{ \begin{array}{l} (t_4^1, t_4^2, t_5^1, t_5^2 \rightarrow t_3) \rightarrow t_8 \perp \omega \rightarrow t_0 \rightarrow t_0 [T], \\ t_6^2 \rightarrow t_3 \perp t_6^1 [t_4^2, t_5^2, t_6^2] \\ t_5^1 \rightarrow t_6^1 \perp t_4^1 [t_5^1], \\ t_5^2 \rightarrow t_6^2 \perp t_4^2 [t_5^2] \end{array} \right.$$

avec  $T = \{t_3, t_4^1, t_4^2, t_5^1, t_5^2, t_6^1, t_6^2\}$

Ces équations correspondent au terme:

$$F(\lambda u (uu)(uu))$$

# Exemple - $F(\lambda u \Delta(uu))$

Résolution de:

$$(t_4^1, t_4^2, t_5^1, t_5^2 \rightarrow t_3) \rightarrow t_8 \perp \omega \rightarrow t_0 \rightarrow t_0 [T]$$

Ne pas “effacer” l’argument, car il doit être typable !  
Nouveau système:

$$\left\{ \begin{array}{l} t_6^2 \rightarrow t_3 \quad \perp \quad t_6^1 \quad [t_4^2, t_5^2, t_6^2], \\ t_5^1 \rightarrow t_6^1 \quad \perp \quad t_4^1 \quad [t_5^1], \\ t_5^2 \rightarrow t_6^2 \quad \perp \quad t_4^2 \quad [t_5^2] \end{array} \right\}$$

Ces équations correspondent aux termes:

$$I \text{ et } \lambda u (uu)(uu)$$

et non pas  $I$  seul

# $\Lambda_{\kappa}$ -calcul

- Inspiré par Klop, 1980.
- Syntaxe:

$$M, N ::= x \mid MN \mid \lambda x M \mid [M, N]$$

- Réduction:

Pour  $x \in fv(M)$ :

$$[\lambda x M, N_1, \dots, N_n] N \longrightarrow_{\kappa} [M\{x \mapsto N\}, N_1, \dots, N_n]$$

Pour  $x \notin fv(M)$ :

$$[\lambda x M, N_1, \dots, N_n] N \longrightarrow_{\kappa} [M, N_1, \dots, N_n, N]$$

# $\Lambda_{\mathcal{K}}$ -calcul

- $\mathcal{WN}_{\mathcal{K}} = \mathcal{SN}_{\mathcal{K}}$ : les termes normalisables sont fortement normalisables
- $\mathcal{SN}_{\Lambda} = \Lambda \cap \mathcal{SN}_{\mathcal{K}}$ : ils correspondent aux termes fortement normalisables du  $\lambda$ -calcul
- Ajout de la règle de typage:

$$\frac{\Gamma_1 \vdash M_1 : \tau \quad \Gamma_2 \vdash M_2 : \sigma}{\Gamma_1, \Gamma_2 \vdash [M_1, M_2] : \tau} \text{(Typ Forget)}$$

# Règles de réduction

Etat du système:  $(\mathcal{E}, \Pi)$  où

- $\mathcal{E}$  est un ensemble de contraintes
- $\Pi$  est un squelette de preuve, qui va évoluer en un arbre de typage valide

# Règles de réduction

Etat du système:  $(\mathcal{E}, \Pi)$  où

- $\mathcal{E}$  est un ensemble de contraintes
- $\Pi$  est un squelette de preuve, qui va évoluer en un arbre de typage valide

Règle pour  $n \geq 1$ :

$$(\{\tau \rightarrow t \perp t_1, \dots, t_n \rightarrow \sigma [T]\} \cup \mathcal{E}, \Pi) \longrightarrow (S(\mathcal{E}), S(\Pi))$$

$$\text{avec } S = \{t_i \mapsto \langle \tau \rangle^i, \langle T \rangle^i\}_{1 \leq i \leq n} :: \{t \mapsto \sigma, \emptyset\} :: D(n, T)$$

$(R_n)$

# Règles de réduction

Règle pour  $n = 0$ :

$$(\{\tau \rightarrow t \perp \omega \rightarrow \sigma [T]\} \cup \mathcal{E}, \Pi) \longrightarrow (S(\mathcal{E}), S(\Pi))$$

avec  $S = \{t \mapsto \sigma, \emptyset\}$

$(R_0)$

Règle finale:

$$(\{\tau \perp t\} \cup \mathcal{E}, \Pi) \longrightarrow_f (S(\mathcal{E}), S(\Pi)) \quad \text{avec } S = \{t \mapsto \tau\}$$

$(R_f)$

# Résultats

- Théorème: Un terme  $M$  est typable si et seulement si le système initial correspondant à  $M$  converge.

# Résultats

- Théorème: Un terme  $M$  est typable si et seulement si le système initial correspondant à  $M$  converge.
- Théorème: Si  $M$  est typable, alors le squelette de preuve final est un arbre de typage valide pour  $M$ .

# Résultats

- Théorème: Un terme  $M$  est typable si et seulement si le système initial correspondant à  $M$  converge.
- Théorème: Si  $M$  est typable, alors le squelette de preuve final est un arbre de typage valide pour  $M$ .
- Théorème: Cet arbre de typage est *principal*.

# Rang

Définition syntaxique sur les types...

- rang 0: types usuels sans intersection
- rang 1: vide
- rang  $r \geq 2$ : il existe une conjonction non triviale sous  $r - 1$  flèches

Exemple:

$(t_1 \rightarrow t_2), (\omega \rightarrow t_3) \rightarrow t_1 \rightarrow t_3$  de rang 3

# Algorithme à rang fini

- On se fixe un rang maximal  $r$ .
- Pour tout état intermédiaire  $(\mathcal{E}, \Pi)$ , vérifier que  $\text{rang}(\Pi) \leq r$ .
- Sinon, le terme n'est pas typable au rang  $r$ .

# Algorithme à rang fini

- On se fixe un rang maximal  $r$ .
- Pour tout état intermédiaire  $(\mathcal{E}, \Pi)$ , vérifier que  $\text{rang}(\Pi) \leq r$ .
- Sinon, le terme n'est pas typable au rang  $r$ .

Propriété: L'algorithme à rang fini *termine toujours*.

Conséquence: L'inférence de types à rang fini est *décidable*.

# TYPI

<http://www-sop.inria.fr/mimosa/Pascal.Zimmer/typi.html>

Implémentation directe de l'algorithme...

# Variante

Que se passe-t-il si on utilise la règle générale aussi pour  $n = 0$  ?

$$(\{\tau \rightarrow t \perp t_1, \dots, t_n \rightarrow \sigma [T]\} \cup \mathcal{E}, \Pi) \longrightarrow (S(\mathcal{E}), S(\Pi))$$

avec  $S = \{t_i \mapsto \langle \tau \rangle^i, \langle T \rangle^i\}_{1 \leq i \leq n} :: \{t \mapsto \sigma, \emptyset\} :: D(n, T)$

$(R_n)$

- Conduit à “effacer” des contraintes ou des sous-arbres par  $D(0, T)$
- Correspondance avec le système de types  $\mathcal{D}\Omega$  (Krivine) ou  $\lambda\cap$  (Barendregt)

$$\overline{\vdash M : \omega} \text{ (Typ } \omega)$$

# Variante

- Propriété: La variante de l'algorithme converge ssi le terme est normalisable.
- Proposition: Un terme est typable dans  $\mathcal{D}\Omega$  par un type non trivial ssi il possède une forme normale de tête.
- Caractérisation des termes normalisables
- Corollaire: Si l'algorithme converge, alors le terme est typable.
- Réciproque non vraie (exemple:  $x\Omega$ )

# Systeme II

- Systeme propose par Kfoury et Wells (variante: Systeme E avec Carlier)
- Les types peuvent contenir des *variables d'expansion*:

$$\begin{aligned}\bar{\psi} &::= \alpha \mid (\psi \rightarrow \bar{\psi}) \\ \psi &::= \bar{\psi} \mid (\psi \wedge \psi') \mid (F\psi)\end{aligned}$$

- Algorithme qui resoud des contraintes similaires et retourne un arbre de typage

# Systeme II

- Correspondance variables d'expansion / territoire:

$$F_T \longleftrightarrow T = \{v \mid F_T \in \text{E-path}(v, \Gamma_{\text{II}}(M))\}$$

- Les deux algorithmes effectuent les mêmes opérations, pas forcément dans le même ordre, si on ignore les variables d'expansion  
→ correspondance opérationnelle
- Utilisée pour éviter de redémontrer certains résultats (principalité, rang fini)

# Références

L'expression

$$(\lambda r (r := ["chaîne"]; \text{hd}(! r) + 1)) (\text{ref } [])$$

est typable, mais son exécution conduit à une erreur..

# Références

L'expression

$$(\lambda r (r := ["chaîne"]; \text{hd}(! r) + 1)) (\text{ref } [])$$

est typable, mais son exécution conduit à une erreur..

Solution similaire à celle du polymorphisme en ML: introduire la conjonction seulement pour les *valeurs* (Davies et Pfenning).

$$\frac{\Gamma \vdash V : A \quad \Gamma \vdash V : B}{\Gamma \vdash V : A \wedge B}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

# Références

- Distinguer les types des termes-variables et des applications:  $t_v$  et  $t_@$
- Syntaxe étendue pour les types:

$$t_b ::= t_v \mid t_b \text{ ref} \mid \text{cte} \mid t_b \text{ list}$$

$$\tau, \sigma ::= t_v \mid \tau \text{ ref} \mid \text{cte} \mid \tau \text{ list} \mid t_@ \mid t_b, \dots, t_b \rightarrow \tau$$

- Equations décomposables:

$$\tau \rightarrow t_@ \perp t_{b_1}, \dots, t_{b_n} \rightarrow \sigma \quad [T]$$

# Références

$$(\{\tau \rightarrow t_{@} \stackrel{\perp}{=} t_{b_1}, \dots, t_{b_n} \rightarrow \sigma [T]\} \cup \mathcal{E}, \Pi) \longrightarrow (S(\mathcal{E}), S(\Pi))$$

$$\text{avec } S = \begin{cases} mgu(t_{b_i}, \langle \tau \rangle^i, \langle T \rangle^i)_{1 \leq i \leq n} :: \{t_{@} \mapsto \sigma, \emptyset\} :: D(n, T) & \text{si } \textit{ValueType}(\tau) \\ mgu(t_{b_i}, \tau, T)_{1 \leq i \leq n} :: \{t_{@} \mapsto \sigma, \emptyset\} & \text{sinon} \end{cases}$$

# Références

$$(\{\tau \rightarrow t_{@} \stackrel{\perp}{=} t_{b_1}, \dots, t_{b_n} \rightarrow \sigma [T]\} \cup \mathcal{E}, \Pi) \longrightarrow (S(\mathcal{E}), S(\Pi))$$

$$\text{avec } S = \begin{cases} mgu(t_{b_i}, \langle \tau \rangle^i, \langle T \rangle^i)_{1 \leq i \leq n} :: \{t_{@} \mapsto \sigma, \emptyset\} :: D(n, T) & \text{si } \textit{ValueType}(\tau) \\ mgu(t_{b_i}, \tau, T)_{1 \leq i \leq n} :: \{t_{@} \mapsto \sigma, \emptyset\} & \text{sinon} \end{cases}$$

mais en plus il faut imposer un ordre de résolution des contraintes, correspondant grosso modo à de l'appel par valeur...

# Récursion

- Ajout d'un opérateur  $\mu x M$
- Solution: inférence comme pour  $M$ , puis algorithme d'unification
- Modification du système de types:

$$\frac{\Gamma, x : \sigma_1, \dots, x : \sigma_n \vdash M : \tau}{\Gamma \vdash \mu x M : \tau} \text{(REC)} \quad \text{avec } \forall i \sigma_i \equiv \tau$$

- Egalité modulo commutativité et contraction:

$$\dots, \tau_1, \tau_2, \dots \rightarrow \sigma \equiv \dots, \tau_2, \tau_1, \dots \rightarrow \sigma$$

$$\dots, \tau, \tau, \dots \rightarrow \sigma \equiv \dots, \tau, \dots \rightarrow \sigma$$

# Futur

- intégrer les types avec intersection dans le langage de MLOBJ
- méthodes polymorphes dans MLOBJ
- étudier de plus près l'expressivité des mixins
- étendre le langage avec d'autres paradigmes

**The end**