Récursion généralisée et inférence de types avec intersection

Soutenance de thèse – 29 avril 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)...

INRIA

Plan

Première partie:

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

INRIA

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

INRIA

Première partie

Récursion généralisée

Sémantique des objets 1

Sémantique par auto-application

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

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

- 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 récursifs

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

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

- objet: $o = \text{fix} \; (C N_1 \ldots N_n)$
- 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 récursifs de Wand
- références à la ML pour stocker l'état de l'objet
- exemples:

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

$$p = \text{fix } (point\ 4)$$

$$color\_point = \lambda x \lambda c \lambda \text{self} \\ \{point\ x\ \text{self}, color = \text{ref } c\}$$

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:

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

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

INRIA

Système de types avec degrés

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

$$\theta^{d} \to \tau$$

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

Degrés - exemples

- c'est le cas de:

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

- $\text{fix} = \lambda f(\text{let rec } x = fx \text{ in } x)$
  a pour type: $(\tau^1 \to \tau)^0 \to \tau$
- $\lambda \text{self}\{x = 0, y = \text{self}.x\}$
  a pour type: $\{\rho, x : \tau\}^0 \to \{x : int, y : \tau\}$
  où ρ 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

INRIA

Algorithmes d'unification et d'inférence

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

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

- contraintes sur variables de rangée ($\rho :: L$) et de degré;
  exemple: $\lambda f \lambda x (f\, x)$ a pour type
  $(\theta^{\alpha} \to \tau)^{\beta} \to \theta^{\gamma} \to \tau$ avec $\gamma \leq \alpha$

INRIA

Machine abstraite

- il faut pouvoir évaluer des termes de la forme

$$(\lambda \mathrm{self}\, M)\; o$$

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

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

Machine abstraite

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

- S: pile de contrôle
- σ: environnement
- M: terme à évaluer
- ξ: mémoire pour les valeurs récursives (et les références)

- ensemble de 11 règles de transition, dont la règle “magique”:

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

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

INRIA

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: $\lambda s\,\{\ldots\}$
- mixin: transformateur de générateur

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

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

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

- 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 $move \ldots$
end

$coloring = \lambda c$
mixin
var $color = c$
meth $paint \ldots$
end

$colorPoint = \lambda x \lambda c$
mixin
inherit $point\ x$
inherit $coloring\ c$
end

$\Rightarrow$ héritage multiple

Mixins - exemples

$$reset =$$
mixin
meth $reset(\text{super}, \text{self}) = \text{self}.pos := 0$
end

$resetPoint = \lambda x$
mixin
inherit $point\ x$
inherit $reset$
end

$resetColorPoint = \lambda x \lambda c$
mixin
inherit $colorPoint\ x\ c$
inherit $reset$
end

$\Rightarrow$ partage de code

Mixins - exemples

mixin

meth $reset(\text{super}, \text{self}) \leftarrow$

$\lambda d\ (\text{super}\#reset;\ \text{super}\#paint\ d)$

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
  $\Rightarrow$ une grande puissance expressive
  encore à explorer !

Et ensuite ?

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

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

- 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

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

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

INRIA

Règles de typage

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

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

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

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

Exemples

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

Propriétés

- Réduction du sujet: Si $M \to M'$, 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 $M \to M'$, 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...

INRIA

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.

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

avec $F^{\mathbf{t}} = \lambda x \lambda y (y : t_0)$
et $\Delta^{\mathbf{t}} = \lambda x\ (x : t_1\ x : t_2) : t_3$

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

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

$$Typ(N^{\mathbf{t}}) \to t \ \doteq\ Typ(M^{\mathbf{t}}) \ \ [ftv(N^{\mathbf{t}})]$$

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

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

$$Typ(N^{\mathbf{t}}) \to t \ \perp\!\!\!=\ Typ(M^{\mathbf{t}}) \ \ [ftv(N^{\mathbf{t}})]$$

$$\left\{\begin{array}{rcll} (t_4, t_5 \to t_7) \to t_8 & \perp\!\!\!= & \omega \to t_0 \to t_0 & [t_1, \ldots, t_7], \\ t_6 \to t_7 & \perp\!\!\!= & t_1, t_2 \to t_3 & [t_4, t_5, t_6], \\ t_5 \to t_6 & \perp\!\!\!= & t_4 & [t_5], \\ t_2 \to t_3 & \perp\!\!\!= & t_1 & [t_2] \end{array}\right\}$$

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

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

Résolution de:

$$t_6 \to t_7 \ \doteq\ t_1, t_2 \to t_3 \ \ [t_4, t_5, t_6]$$

Nouveau système:

$$\begin{cases} (t_4^1, t_4^2, t_5^1, t_5^2 \to t_3) \to t_8 & \doteq & \omega \to t_0 \to t_0 & [T], \\ t_6^2 \to t_3 & \doteq & t_6^1 & [t_4^2, t_5^2, t_6^2] \\ t_5^1 \to t_6^1 & \doteq & t_4^1 & [t_5^1], \\ t_5^2 \to t_6^2 & \doteq & t_4^2 & [t_5^2] \end{cases}$$

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:

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

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

$$\left\{ \begin{array}{lcll} t_6^2 \to t_3 & \doteq & t_6^1 & [t_4^2, t_5^2, t_6^2], \\ t_5^1 \to t_6^1 & \doteq & t_4^1 & [t_5^1], \\ t_5^2 \to t_6^2 & \doteq & t_4^2 & [t_5^2] \end{array} \right\}$$

Ces équations correspondent aux termes:

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

et non pas I seul

Λ_κ-calcul

- Inspiré par Klop, 1980.
- Syntaxe:

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

- Réduction:

Pour $x \in fv(M)$:

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

Pour $x \notin fv(M)$:

$$[\lambda x M, N_1, \ldots, N_n]\, N \longrightarrow_\kappa [M, N_1, \ldots, N_n, N]$$

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

- $\mathcal{WN}_{\mathcal{K}} = \mathcal{SN}_{\mathcal{K}}$: les termes normalisables sont fortements normalisables
- $\mathcal{SN}_{\Lambda} = \Lambda \cap \mathcal{SN}_{\mathcal{K}}$: ils correspondent aux termes fortement normalisables du λ-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
- Π 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
- Π est un squelette de preuve, qui va évoluer en un arbre de typage valide

Règle pour $n \geq 1$:

$$(\{\tau \to t \doteq t_1, \ldots, t_n \to \sigma \ [T]\} \cup \mathcal{E},\ \Pi) \longrightarrow (S(\mathcal{E}),\ S(\Pi))$$

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)

Règles de réduction

Règle pour $n = 0$:

$$(\{\tau \to t \doteq \omega \to \sigma \ [T]\} \cup \mathcal{E},\ \Pi) \longrightarrow (S(\mathcal{E}),\ S(\Pi))$$
$$\text{avec } S = \{t \mapsto \sigma, \emptyset\}$$

(R_0)

Règle finale:

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

(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.

INRIA

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*.

INRIA

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; mesure la “profondeur” du polymorphisme.

INRIA

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; mesure la “profondeur” du polymorphisme.

  Propriété: L’algorithme à rang fini *termine toujours*.
  Conséquence: L’inférence de types à rang fini est *décidable*.

INRIA

Autres résultats

- Implémentation de l'algorithme: TYPI

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

- Variante: remplacement de (R_0) par la règle générale (R_n); équivalent au système de types $\mathcal{D}\Omega$, avec la règle: $\dfrac{}{\vdash M : \omega}\,(\text{Typ } \omega)$
- Extension aux références (introduction de la conjonction seulement pour les valeurs, comme en ML; moins de liberté sur l'ordre de résolution)
- Extension à la récursion $\mu x M$ (unification supplémentaire en fin d'algorithme)

  dans le but de typer MLOBJ ...

INRIA

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

INRIA

The end

Variante

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

$$(\{\tau \to t \doteq t_1, \ldots, t_n \to \sigma \ [T]\} \cup \mathcal{E},\ \Pi) \longrightarrow (S(\mathcal{E}),\ S(\Pi))$$
$$\text{avec } S = \{t_i \mapsto \langle\tau\rangle^i, \langle T\rangle^i\}_{1\leq i\leq n} :: \{t \mapsto \sigma, \emptyset\} :: D(n, T)$$

(R_n)

- 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)

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

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: $x\Omega$)

Système $\mathbb{I}$

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

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

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

Système $\mathbb{I}$

- Correspondance variables d'expansion / territoire:

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

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

Références

L'expression

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

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

Références

L'expression

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

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).

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

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

Références

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

$$t_b ::= t_v \mid t_b\, ref \mid cte \mid t_b\, list$$

$$\tau, \sigma ::= t_v \mid \tau\, ref \mid cte \mid \tau\, list \mid t_@ \mid t_b, \ldots, t_b \rightarrow \tau$$

- Equations décomposables:

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

Références

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

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

Références

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

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

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:

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

- Egalité modulo commutativité et contraction:

$$\ldots, \tau_1, \tau_2, \ldots \to \sigma \quad \equiv \quad \ldots, \tau_2, \tau_1, \ldots \to \sigma$$

$$\ldots, \tau, \tau, \ldots \to \sigma \quad \equiv \quad \ldots, \tau, \ldots \to \sigma$$

Rang

$$inc(0) = 0$$
$$inc(n) = n + 1 \quad \text{pour } n > 0$$

$$rank(t) = 0$$
$$rank(\tau \to \sigma) = \max(inc(rank(\tau)), rank(\sigma))$$
$$rank(\tau_1, \ldots, \tau_n \to \sigma) =$$
$$\max(inc(\max(1, rank(\tau_1), \ldots, rank(\tau_n))), rank(\sigma))$$
$$\text{pour } n \neq 1$$

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

Algorithme à rang fini

- On se fixe un rang maximal r.
- Pour tout état intermédiaire $(\mathcal{E}, \Pi)$, vérifier que $rang(\Pi) \leq r$.
- 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 $rang(\Pi) \leq r$.
- 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*.