Cours 3

Inférence de type
Polymorphisme

Jean-Jacques.Levy@inria.fr

http://jeanjacqueslevy.net

secrétariat de l'enseignement:

Catherine Bensoussan

cb@lix.polytechnique.fr

Laboratoire d'Informatique de l'X

Aile 00, LIX

tel: 34 67

http://w3.edu.polytechnique.fr/informatique

Références

- Programming Languages, Concepts and Constructs, Ravi Sethi, 2nd edition, 1997.
- Theories of Programming Languages, J. Reynolds, Cambridge University Press, 1998.
- Type Systems for Programming Languages, Benjamin C. Pierce, Cours University of Pennsylvania, 2000.
- Programming Languages: Theory and Practice, Robert Harper, Cours Carnegie Mellon University, Printemps 2000.
- Notes du cours de DEA “Typage et programmation”, Xavier Leroy, Cours de DEA, Décembre 1999, `http://pauillac.inria.fr/~xleroy/dea`.

Plan

1. Typage à la Church
2. Inférence de type
3. Unification
4. Listes polymorphes
5. Polymorphisme

Rappel: règles de typage monomorphe

$\rho, x : t \vdash x : t$

$$\frac{\rho, x : t \vdash M : t'}{\rho \vdash \lambda x : t . M : t \to t'}$$

$$\frac{\rho \vdash M : t \to t' \qquad \rho \vdash N : t}{\rho \vdash M N : t'}$$

$\rho \vdash \underline{n} : \mathtt{int}$

$$\frac{\rho \vdash M : \mathtt{int} \qquad \rho \vdash N : \mathtt{int}}{\rho \vdash M \otimes N : \mathtt{int}}$$

$$\frac{\rho \vdash M : \mathtt{int} \qquad \rho \vdash N : t \qquad \rho \vdash N' : t}{\rho \vdash \mathtt{ifz}\, M \,\mathtt{then}\, N \,\mathtt{then}\, N' : t}$$

$$\frac{\rho, x : t \vdash M : t}{\rho \vdash \mu x : t . M : t}$$

$$\frac{\rho \vdash M : t \qquad \rho, x : t \vdash N : t'}{\rho \vdash \mathtt{let}\ x : t = M \ \mathtt{in}\ N : t'}$$

Présentation de PCF typé à la Church

Termes sans annotations de type. Les règles de typage sont

$$\vdash \rho, x : t \vdash x : t$$

$$\frac{\rho, x : t \vdash M : t'}{\rho \vdash \lambda x.M : t \to t'} \qquad \frac{\rho \vdash M : t \to t' \quad \rho \vdash N : t}{\rho \vdash MN : t'}$$

$$\rho \vdash \underline{n} : \mathtt{int} \qquad \frac{\rho \vdash M : \mathtt{int} \quad \rho \vdash N : \mathtt{int}}{\rho \vdash M \otimes N : \mathtt{int}}$$

$$\frac{\rho \vdash M : \mathtt{int} \quad \rho \vdash N : t \quad \rho \vdash N' : t}{\rho \vdash \mathtt{ifz}\, M \,\mathtt{then}\, N \,\mathtt{then}\, N' : t}$$

$$\frac{\rho, x : t \vdash M : t}{\rho \vdash \mu x.M : t}$$

$$\frac{\rho \vdash M : t \quad \rho, x : t \vdash N : t'}{\rho \vdash \mathtt{let}\ x = M\ \mathtt{in}\ N : t'}$$

Exercice 1 **Montrer que le théorème d'unicité n'est plus vrai.**

Inférence de type

Ecrire le type de l'argument et des résultats des fonctions peut être pénible. Informatif parfois, mais souvent très redondant.

Par exemple dans

$$\texttt{let}\ f : \texttt{int} \to \texttt{int} = \lambda x : \texttt{int}\,.x\ \texttt{in}\ \ldots$$

ou même

$$\texttt{let}\ f(x : \texttt{int}) : \texttt{int} = x\ \texttt{in}\ \ldots$$

C'est plus simple d'écrire

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

ou

$$\texttt{let}\ f(x) = x\ \texttt{in}\ \ldots$$

et d'avoir le système qui calcule automatiquement le type (s'il existe).

Règles de typage monomorphe – bis

Dans M, toutes les variables libres ou liées sont distinctes (toujours possible par α-conversion). On associe à chaque variable x une variable de type α_x et à chaque sous-terme N de M une variable de type α_N.

Pour M, on génère un système d'équations $C(M)$ de la manière suivante.

M	$C(M)$
x	$\{\alpha_M = \alpha_x\}$
$\lambda x.N$	$\{\alpha_M = \alpha_x \to \alpha_N\} \cup C(N)$
$N P$	$\{\alpha_N = \alpha_P \to \alpha_M\} \cup C(N) \cup C(P)$
$\underline{n}$	$\{\alpha_M = \texttt{int}\}$
$N \otimes P$	$\{\alpha_M = \texttt{int} = \alpha_N = \alpha_P\} \cup C(N) \cup C(P)$
$\texttt{ifz}\, N \,\texttt{then}\, P \,\texttt{then}\, P'$	$\{\alpha_M = \alpha_P = \alpha_{P'},\ \alpha_N = \texttt{int}\} \cup C(N) \cup C(P) \cup C(P')$
$\mu x.N$	$\{\alpha_M = \alpha_N,\ \alpha_M = \alpha_x\} \cup C(N)$
$\texttt{let}\ x = N\ \texttt{in}\ P$	$\{\alpha_M = \alpha_P,\ \alpha_x = \alpha_N\} \cup C(N) \cup C(P)$

Exemple

$M = \mathtt{let}\ x = \underline{3}\ \mathtt{in}\ x + \underline{1}$

$\alpha_M = \alpha_2,\ \alpha_x = \alpha_3$

$\alpha_3 = \mathtt{int}$

$\alpha_2 = \mathtt{int} = \alpha_4 = \alpha_1$

$\alpha_4 = \alpha_x$

Solution $\alpha_M = \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \mathtt{int}$

$\lambda x.\lambda y.x$

$\alpha_M = \alpha_x \rightarrow \alpha_1$

$\alpha_1 = \alpha_y \rightarrow \alpha_2$

$\alpha_2 = \alpha_x$

Solution $\alpha_M = \alpha_x \rightarrow \alpha_y \rightarrow \alpha_x$

Exercice 2 **Essayer pour** `fact`**,** $\mathtt{let}\ f = \lambda x.x\ \mathtt{in}\ f(f\underline{2})$ **et** $\mathtt{let}\ f = \lambda x.x\ \mathtt{in}\ (f\,f)\underline{2}$**.**

Exercice 3 **Donner les équations générées pour les listes.**

Equations sur l'algèbre des types

Trouver un type polymorphe revient à résoudre le système d'équations précédent.

Théorème 1 $\vdash M : t$ ssi $C(M)$ admet une solution.

Ce système admet toujours une solution la plus générale: le type principal de M.

Comment la calculer?

$\Rightarrow$ théorie de l'unification (logique, 1960), utilisée dans la démonstration automatique pour la méthode dite de résolution.

L'unification sert à trouver des solutions dans des algèbres libres. Plusieurs algorithmes: Robinson (exponentiel, et très simple), Huet, Martelli, Montanari (quasi linéaire et simple), Patterson (linéaire, mais compliqué).

Dans ce cours, nous ne considérerons l'unification que sur l'ensemble des types.

Substitutions et types

$$
\begin{array}{rcll}
t, t' & ::= & \alpha & \text{variable de type} \\
& | & \texttt{int} & \text{les entiers naturels } \mathbf{N} \\
& | & t \rightarrow t' & \text{les fonctions de } T \text{ dans } T'
\end{array}
$$

Une substitution σ est une fonction des variables de type dans les types. On écrira $\sigma = [\alpha_1 \backslash t_1, \alpha_2 \backslash t_2, \cdots \alpha_n \backslash t_n]$ quand son graphe est fini. On l'étend naturellement comme fonction des types dans les types. On écrit $t\sigma$ pour le terme obtenu en appliquant σ à t. De même, pour la composition $\sigma \circ \sigma'$ de deux substitutions σ et σ'.

Posons $t \leq t'$ ss'il existe σ tel que $t' = t\sigma$
et $\sigma \leq \sigma'$ ss'il existe ϕ tel que $\sigma' = \sigma \circ \phi$.

Remarque: les types ont une structure de treillis vis à vis de $\leq$.

De même, tout système d'équations sur les types admet une solution minimale au sens de $\leq$, si une telle solution existe.

Unification

$\mathtt{mgu}(C)$ **est la plus petite solution du système d'équations** C **si elle existe. Le résultat est donc** `echec` **ou une substitution. (***most general unifier***)**

Algorithme de Robinson

$$
\begin{array}{rcll}
\mathtt{mgu}(\emptyset) & = & \mathtt{identite} & \\
\mathtt{mgu}(\{\alpha = \alpha\} \cup C) & = & \mathtt{mgu}(C) & \\
\mathtt{mgu}(\{\alpha = t\} \cup C) & = & \mathtt{mgu}(C[\alpha \backslash t]) \circ [\alpha \backslash t] & \textbf{si } \alpha \notin FV(t) \\
\mathtt{mgu}(\{t = \alpha\} \cup C) & = & \mathtt{mgu}(C[\alpha \backslash t]) \circ [\alpha \backslash t] & \textbf{si } \alpha \notin FV(t) \\
\mathtt{mgu}(\{t_1 \to t_2 = t'_1 \to t'_2\} \cup C) & = & \mathtt{mgu}(\{t_1 = t'_1,\ t_2 = t'_2\} \cup C) & \\
\textbf{sinon} \quad \mathtt{mgu}(C) & = & \mathtt{echec} &
\end{array}
$$

Exercice 4 **Montrer que cet algorithme termine toujours (Indication compter le nombre de variables). Quelle est sa complexité?**

Type principal

Théorème 2 **Le type t trouvé pour M en résolvant $C(M)$ par l'algorithme d'unification est principal, c'est-à-dire que si $\vdash M : t'$, alors $t \leq t'$.**

Exemples

$I = \lambda x.x : \alpha \to \alpha$
$K = \lambda x.\lambda y.x : \alpha \to \beta \to \alpha$
$S = \lambda x.\lambda y.\lambda z.xz(yz) : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$
$B = \lambda f.\lambda g.\lambda x.g(fx) : (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$

Polymorphisme

Quelques fonctions générales peuvent être réutilisables.

Par exemple $\lambda x : \texttt{int} . x$ a le même code que $\lambda x : \texttt{char} . x$ ou $\lambda x : \texttt{string} . x$

De même $\texttt{append}\, x\, y$ concatène deux listes sans se soucier du type de leurs éléments.

En ne précisant pas les types, on peut utiliser les fonctions plusieurs fois

Ecrire une fois, utiliser partout (R.Harper)

Listes polymorphes

Dans l'extension de PCF avec les listes, on a pour tout α

Terme	Type
`nil`	$\mathtt{list}(\alpha)$
`hd`	$\mathtt{list}(\alpha) \to \alpha$
`tl`	$\mathtt{list}(\alpha) \to \mathtt{list}(\alpha)$
$::$	$\alpha \to \mathtt{list}(\alpha) \to \mathtt{list}(\alpha)$

Ni `hd` ni `tl` ni $::$ ne sont des termes de PCF étendu. Mais $\mathtt{hd}(M)$, $\mathtt{tl}(M)$, $M :: N$ sont des termes de PCF.

En C ou en Java, on peut faire des listes de caractères ou d'*Object*s. Mais il faut faire des conversions explicites pour passer du type $\mathtt{list}(\alpha)$ à ces listes plus spécifiques. En C, ca ne coûte rien, mais ce n'est pas sûr. En Java, il y a vérification dynamique du type quand on passe du type $\mathtt{list}(\mathit{Object})$ au type plus spécifique $\mathtt{list}(\alpha)$.

Types polymorphes

Problème Si on veut utiliser les listes de tout type librement dans PCF, il faut pouvoir parler du type $\texttt{list}(\alpha)$ où α est une variable de type.

Comment avoir des termes de type $\texttt{list}(\alpha)$ sans coercions ni vérifications de type dynamique?

$\Rightarrow$ **Types polymorphes**

$\texttt{let append} = \mu f.\lambda x.\lambda y.\,\texttt{ifnil}\,x\;\texttt{then}\;y\;\texttt{else}\;(\texttt{hd}\,x) :: f(\texttt{tl}\,x)\;y\;\texttt{in}$
$\texttt{let}\;\ell_0 = 1 :: 2 :: 3 :: \texttt{nil in}$
$\texttt{let}\;\ell_1 = 4 :: 5 :: \texttt{nil in}$
$\texttt{let}\;\ell_2 = 6 :: 7 :: 8 :: \texttt{nil in}$
$\texttt{let}\;\ell'_0 = \ell_0 :: \ell_1 :: \ell_2 :: \texttt{nil in}$
$\texttt{let}\;\ell'_1 = \ell_2 :: \texttt{nil in}$
$\texttt{hd}(\texttt{append}\,\ell_0\;\ell_1) + \texttt{hd}(\texttt{hd}(\texttt{append}\,\ell'_0\;\ell'_1))$

Credo

Polymorphisme

$+$

Typage statique

$\Rightarrow$

Généralité $+$ **Sureté** $+$ **Efficacité**

Types polymorphes fonctionnels

Les fonctions manipulant des listes sont de type polymorphe.

$\texttt{let map} = \mu\,\texttt{map}\,.\lambda f.\lambda x.$
$\quad \texttt{ifnil}\,x\;\texttt{then nil else}\;f(\texttt{hd}\,x) :: \texttt{map}\,f(\texttt{tl}\,x)\;\texttt{in}$
$\texttt{map}(\lambda x.x+1)(3 :: 4 :: \texttt{nil})$

Rappel: en abrégé, cette fonction peut aussi s'écrire

$\texttt{letrec map}\,f\;x =$
$\quad \texttt{ifnil}\,x\;\texttt{then nil else}\;f(\texttt{hd}\,x) :: \texttt{map}\,f(\texttt{tl}\,x)\;\texttt{in}$
$\texttt{map}(\lambda x.x+1)(3 :: 4 :: \texttt{nil})$

est de type $(\alpha \to \beta) \to \texttt{list}(\alpha) \to \texttt{list}(\beta)$

Même des fonctions (moins utiles) ne travaillant pas sur des listes peuvent avoir des types polymorphes.

$\texttt{let}\;I = \lambda x.x\;\texttt{in}\ldots$
$\texttt{let}\;K = \lambda x.\lambda y.x\;\texttt{in}\ldots$
$\texttt{let}\;S = \lambda x.\lambda y.\lambda z.xz(yz)\;\texttt{in}\ldots$
$\texttt{let comp} = \lambda f.\lambda g.\lambda x.f(gx)\;\texttt{in}\ldots$

Quels sont les lois de typage de tels termes?

Règles de typage polymorphe

$$\frac{t \leq \tau}{\rho, x : \tau \vdash x : t}$$ **instanciation**

$$\frac{\rho, x : t \vdash M : t'}{\rho \vdash \lambda x.M : t \rightarrow t'} \qquad \frac{\rho \vdash M : t \rightarrow t' \quad \rho \vdash N : t}{\rho \vdash MN : t'}$$

$$\rho \vdash \underline{n} : \texttt{int} \qquad \frac{\rho \vdash M : \texttt{int} \quad \rho \vdash N : \texttt{int}}{\rho \vdash M \otimes N : \texttt{int}}$$

$$\frac{\rho \vdash M : \texttt{int} \quad \rho \vdash N : t \quad \rho \vdash N' : t}{\rho \vdash \texttt{ifz}\, M \;\texttt{then}\; N \;\texttt{then}\; N' : t}$$

$$\frac{\rho, x : t \vdash M : t}{\rho \vdash \mu x.M : t}$$

$$\frac{\rho \vdash M : t \quad \rho, x : \mathit{Gen}(t, \rho) \vdash N : t'}{\rho \vdash \texttt{let}\; x = M \;\texttt{in}\; N : t'}$$ **généralisation**

Légère variation sur les règles monomorphes.

Types polymorphes

$$\begin{array}{rcll} \tau, \tau' & ::= & \forall \alpha_1, \alpha_2, \cdots \alpha_n.t \quad (n \geq 0) & \textbf{schéma de type} \\ t, t' & ::= & \alpha & \textbf{variable de type} \\ & | & \texttt{int} & \textbf{les entiers naturels } \mathbf{N} \\ & | & t \rightarrow t' & \textbf{les fonctions de } T \textbf{ dans } T' \end{array}$$

Variables de type libres

$FV(\alpha) = \{\alpha\}$ $\quad$ $FV(\forall \alpha_1, \alpha_2, \cdots \alpha_n.t) = FV(t) - \{\alpha_1, \alpha_2, \cdots \alpha_n\}$

$FV(\texttt{int}) = \emptyset$ $\quad$ $FV(t \rightarrow t') = FV(t) \cup FV(t')$

$FV(\emptyset) = \emptyset$ $\quad$ $FV(\rho, x : \tau) = FV(\rho) \cup FV(\tau)$

Généralisation

$Gen(t, \rho) = \forall \alpha_1, \alpha_2, \cdots \alpha_n.t$ **où** $\{\alpha_1, \alpha_2, \cdots \alpha_n\} = FV(t) - FV(\rho)$

Instanciation

$t \leq \forall \alpha_1, \alpha_2, \cdots \alpha_n.t'$

ss'il existe $t_1, t_2, \cdots t_n$ **tels que** $t = t'[\alpha_1 \backslash t_1, \alpha_2 \backslash t_2, \cdots \alpha_n \backslash t_n]$

Exemples

$\mathtt{let}\ x = \underline{3}\ \mathtt{in}\ x + \underline{1}$

$$\dfrac{\rho \vdash \underline{3} : \mathtt{int} \qquad \dfrac{\rho, x : \mathtt{int} \vdash x : \mathtt{int} \qquad \rho, x : \mathtt{int} \vdash \underline{1} : \mathtt{int}}{\rho, x : \mathtt{int} \vdash x + \underline{1} : \mathtt{int}}}{\rho \vdash \mathtt{let}\ x = \underline{3}\ \mathtt{in}\ x + \underline{1} : \mathtt{int}}$$

$\lambda x.x$

$\lambda x.\lambda y.x$

$$\dfrac{x : \alpha \vdash x : \alpha}{\vdash \lambda x.x : \alpha \to \alpha} \qquad\qquad \dfrac{\dfrac{x : \alpha, y : \beta \vdash x : \beta}{x : \alpha \vdash \lambda y.x : \alpha \to \beta}}{\vdash \lambda x.\lambda y.x : \alpha \to \beta \to \alpha}$$

$\mathtt{let}\ f = \lambda x.x\ \mathtt{in}\ f\underline{2}$

$$\dfrac{\dfrac{x : \alpha \vdash x : \alpha}{\vdash \lambda x.x : \alpha \to \alpha} \qquad \dfrac{f : \forall \alpha.\alpha \to \alpha \vdash f : \mathtt{int} \to \mathtt{int} \qquad f : \forall \alpha.\alpha \to \alpha \vdash \underline{2} : \mathtt{int}}{f : \forall \alpha.\alpha \to \alpha \vdash f\ \underline{2} : \mathtt{int}}}{\vdash \mathtt{let}\ f = \lambda x.x\ \mathtt{in}\ f\ \underline{2} : \mathtt{int}}$$

Exercice 5 **En appliquant les règles précédentes, trouver les types de**

$\texttt{let}\ f = \lambda x.x\ \texttt{in}\ f(f\underline{2})$

$\texttt{let}\ f = \lambda x.x\ \texttt{in}\ (f\ f)\underline{2}$

$\texttt{let}\ x = \underline{2}\ \texttt{in}\ \texttt{let}\ y = x + \underline{1}\ \texttt{in}\ \underline{2} \times x + y$

$\texttt{let}\ f = \lambda x.x \times x + \underline{1}\ \texttt{in}\ f\underline{4} + f\underline{5}$

$\texttt{let}\ f = \lambda x.x \times x + \underline{1}\ \texttt{in}\ f(fx)$

$\texttt{let}\ \texttt{fact} = \mu f.\lambda x.\texttt{ifz}\ x\ \texttt{then}\ \underline{1}\ \texttt{else}\ x \times f(x - \underline{1})\ \texttt{in}\ \texttt{fact}\ \underline{5}$

Extension avec les listes

$$\frac{t \leq \forall\alpha.\,\texttt{list}(\alpha)}{\rho \vdash \texttt{nil} : t}\ \textbf{nil} \qquad\qquad \frac{M : t \quad N : \texttt{list}(t)}{\rho \vdash M :: N : \texttt{list}(t)}\ \textbf{cons}$$

$$\frac{\rho \vdash M : \texttt{list}(t') \quad \rho \vdash N : t \quad \rho \vdash N' : t}{\rho \vdash \texttt{ifnil}\, M\ \texttt{then}\ N\ \texttt{then}\ N' : t}\ \textbf{cond'}$$

$$\frac{\rho \vdash M : \texttt{list}(t)}{\rho \vdash \texttt{hd}(M) : t}\ \textbf{hd} \qquad\qquad \frac{\rho \vdash M : \texttt{list}(t)}{\rho \vdash \texttt{tl}(M) : \texttt{list}(t)}\ \textbf{tl}$$

Et si on veut autoriser `hd`, `tl` dans le langage, comme citoyen de 1ère classe, on remplace les 2 dernières règles par

$$\frac{t \leq \forall\alpha.\,\texttt{list}(\alpha) \rightarrow \alpha}{\rho \vdash \texttt{hd} : t}\ \textbf{hd} \qquad\qquad \frac{t \leq \forall\alpha.\,\texttt{list}(\alpha) \rightarrow \texttt{list}(\alpha)}{\rho \vdash \texttt{tl} : t}\ \textbf{tl}$$

Exercices avec les listes

$\lambda x.\, \mathtt{hd}(x)$

$$\dfrac{\dfrac{\dfrac{\mathtt{list}(\alpha) \to \alpha \leq \forall \alpha.\, \mathtt{list}(\alpha) \to \alpha}{x : \mathtt{list}(\alpha) \vdash \mathtt{hd} : \mathtt{list}(\alpha) \to \alpha} \qquad x : \mathtt{list}(\alpha) \vdash x : \alpha}{x : \mathtt{list}(\alpha) \vdash \mathtt{hd}(x) : \alpha}}{\vdash \lambda x.\, \mathtt{hd}(x) : \mathtt{list}(\alpha) \to \alpha}$$

Exercice 6 **Typer `map`.**

Quelques théorèmes

Proposition 1 Si $\rho \vdash M : t$, alors $\rho \vdash M : t[\alpha \backslash t']$

Théorème 2 **[Type principal]** Chaque terme M a un type principal. Tous ses autres types s'obtiennent par substitution.

Théorème 3 `let` $x = M$ `in` N est typé ssi $M[x \backslash N]$ est typé.

En TD

- finir l'évaluateur symbolique de PCF
- finir l'interpréteur avec environnements de PCF pour l'appel par valeur. Montrer que son écriture est bien plus courte que celle de l'évaluateur symbolique.
- faire le vérificateur de type

A la maison et prochaine fois

- commencer un algorithme d'inférence de type; rajouter les listes
- équivalence de types
- langages impératifs et méthode des invariants

Interpréteur

Donner la valeur et dessiner son arbre de syntaxe abstraite pour les termes:

$\texttt{let } I = \lambda x.x \texttt{ in let } \Delta = \lambda x.xx \texttt{ in } II$

$\texttt{let } S = \lambda x.\lambda y.\lambda z.x\ z(y\ z) \texttt{ in let } K = \lambda x.\lambda y.x \texttt{ in } SKK$

$\texttt{let } I = \lambda x.x \texttt{ in } (\lambda x.xx)(\lambda x.xI)$

$\texttt{let fact} = \lambda x.$
$\quad \texttt{let } f = \mu f.\lambda x.\lambda r.\texttt{ ifz } x \texttt{ then } r \texttt{ else } f(x-1)(x*r) \texttt{ in}$
$\quad f\ x\ 1 \texttt{ in}$
$\texttt{fact } \underline{5}$