Cours 4

Inférence de type polymorphe — Mémoire

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. http://cm.bell-labs.com/who/ravi/teddy/
- Theories of Programming Languages, J. Reynolds, Cambridge University Press, 1998. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/jcr/www/
- Type Systems for Programming Languages, Benjamin C. Pierce, Cours University of Pennsylvania, 2000. http://www.cis.upenn.edu/~bcpierce/
- Programming Languages: Theory and Practice, Robert Harper, Cours Carnegie Mellon University, Printemps 2000. http://www.cs.cmu.edu/~rwh/
- Notes du cours de DEA "Typage et programmation", Xavier Leroy, Cours de DEA, Décembre 1999, http://pauillac.inria.fr/~xleroy/dea

Plan

1. Inférence de type polymorphe
2. Algorithme W (Damas, Milner, Tofte)
3. Références
4. Tableaux
5. Typage des références
6. Analyse statique

Règles de typage polymorphe – Rappel

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

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

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

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

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

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

Le polymorphisme est induit par les déclarations $\mathtt{let}\; x = M \;\mathtt{in}\; N$.

Types polymorphes

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

Variables de type libres

$$
\begin{array}{ll}
FV(\alpha) = \{\alpha\} & FV(\forall \alpha_1, \alpha_2, \cdots \alpha_n . t) = FV(t) - \{\alpha_1, \alpha_2, \cdots \alpha_n\} \\
FV(\mathtt{int}) = \emptyset & FV(t \rightarrow t') = FV(t) \cup FV(t') \\
FV(\emptyset) = \emptyset & FV(\rho, x : \tau) = FV(\rho) \cup FV(\tau)
\end{array}
$$

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** $t = t'[\alpha_1 \backslash t_1, \alpha_2 \backslash t_2, \cdots \alpha_n \backslash t_n]$

Substitutions et types

Une substitution σ est une fonction des variables de type dans les types. On écrit $\sigma = [\alpha_1 \backslash t_1, \alpha_2 \backslash t_2, \cdots \alpha_n \backslash t_n]$ quand son graphe est fini, et on pose *domain*$(\sigma) = \{\alpha_1, \alpha_2, \cdots \alpha_n\}$**.**

Une substitution est étendue naturellement en fonction des types dans les types. La composition des substitutions σ et σ' est notée $\sigma \circ \sigma'$.

Posons $\sigma \leq \sigma'$ ss'il existe ϕ tel que $\sigma' = \sigma \circ \phi$. S'il existe, $\mathtt{mgu}(t = t')$ est le plus petit σ tel que $\sigma(t) = \sigma(t')$.

Enfin, une substitution ϕ s'étend aussi naturellement à tout environnement ρ comme suit:

$$\phi(\{x_1 : t_1,\ x_2 : t_2,\ \ldots x_n : t_n\} = \{x_1 : \phi(t_1),\ x_2 : \phi(t_2),\ \ldots x_n : \phi(t_n)\}$$

Inférence de type en PCF

Damas-Milner-Tofte ont un algorithme W combinant unification et règles de typage

Entrée de l'algorithme W:

- M terme à typer,
- ρ un environnement de typage,

Sortie de W:

- le type t inféré pour M,
- la substitution à effectuer sur les variables de ρ donnant ρ' tel que $\rho' \vdash M : t$,

Algorithme W

Donnés: ρ **et** M**. Chercher** t **et** ϕ **tels que** $\phi(\rho) \vdash M : t$

$W(\rho + \{x:\tau\}, x) = (t, \texttt{identite})$ $\qquad$ (t instance de τ par des variables fraiches)

$W(\rho, \lambda x.M) = (\phi(\alpha) \to t', \phi)$ **où**

$\qquad W(\rho + \{x:\alpha\}, M) = (t', \phi)$ $\qquad$ (α variable fraiche)

$W(\rho, MN) = (\sigma(\alpha), \sigma \circ \phi' \circ \phi)$ **où**

$\qquad W(\rho, M) = (t, \phi) \qquad W(\phi(\rho), N) = (t', \phi') \qquad \sigma = \texttt{mgu}(\phi'(t) = t' \to \alpha)$

$\qquad$ (α variable fraiche) **Retourner échec si** σ **n'existe pas.**

$W(\rho, \texttt{let } x = M \texttt{ in } N) = (t', \phi' \circ \phi)$ **où**

$\qquad W(\rho, M) = (t, \phi) \qquad W(\phi(\rho) + \{x : Gen(t, \phi(\rho))\}, N) = (t', \phi')$

$W(\rho, \underline{n}) = (\texttt{int}, \texttt{identite})$

$W(\rho, M \otimes N) = (\texttt{int}, \sigma' \circ \phi' \circ \sigma \circ \phi)$ **où**

$\qquad W(\rho, M) = (t, \phi) \qquad \sigma = \texttt{mgu}(t = \texttt{int})$

$\qquad W((\sigma \circ \phi)(\rho), N) = (t', \phi') \qquad \sigma' = \texttt{mgu}(t' = \texttt{int})$

$\qquad$ **Retourner échec si** σ **ou** σ' **n'existe pas.**

Exercice 1 **Donner l'expression de** W **pour** `ifz` **et** μ**.**

Exemples

$\lambda x.x + x$

$$\begin{aligned}
W(\emptyset, \lambda x.x + x) &= (\texttt{int} \to \texttt{int}, [\alpha\backslash\, \texttt{int}]) \\
W(\{x : \alpha\}, x + x) &= (\texttt{int}, [\alpha\backslash\, \texttt{int}]) \\
W(\{x : \alpha\}, x) &= (\alpha, \texttt{identite}) \\
\sigma &= \texttt{mgu}(\alpha = \texttt{int}) = [\alpha\backslash\, \texttt{int}] \\
W(\{x : \texttt{int}\}, x) &= (\texttt{int}, \texttt{identite}) \\
\sigma' &= \texttt{mgu}(\texttt{int} = \texttt{int}) = \texttt{identite}
\end{aligned}$$

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

$$\begin{aligned}
W(\emptyset, \texttt{let}\ f = \lambda x.x\ \texttt{in}\ f\ \underline{2}) &= (\texttt{int}, [\beta\backslash\, \texttt{int}, \gamma\backslash\, \texttt{int}]) \\
W(\emptyset, \lambda x.x) &= (\alpha \to \alpha, \texttt{identite}) \\
W(\{x : \alpha\}, x) &= (\alpha, \texttt{identite}) \\
W(\{f : \forall\alpha.\alpha \to \alpha\}, f\ \underline{2}) &= (\texttt{int}, [\beta\backslash\, \texttt{int}, \gamma\backslash\, \texttt{int}]) \\
W(\{f : \forall\alpha.\alpha \to \alpha\}, f) &= (\beta \to \beta, \texttt{identite}) \\
W(\{f : \forall\alpha.\alpha \to \alpha\}, \underline{2}) &= (\texttt{int}, \texttt{identite}) \\
\sigma &= \texttt{mgu}(\beta \to \beta = \texttt{int} \to \gamma) = [\beta\backslash\, \texttt{int}, \gamma\backslash\, \texttt{int}]
\end{aligned}$$

Proposition 1 [correction]
$W(\rho, M) = (t, \phi)$ **implique** $\phi(\rho) \vdash M : t$

Proposition 2 [complétude]
$\phi(\rho) \vdash M : t$ **implique** $W(\rho, M) = (t', \phi')$ **pour un** t' **et** ϕ'.

Proposition 3 [type principal]
$W(\rho, M) = (t, \phi)$ **et** $\phi'(\rho) \vdash M : t'$ **impliquent** $t \leq t'$ **et** $\phi \leq \phi'$.

Corollaire 4 $\emptyset \vdash M : t$ **ssi** $W(\emptyset, M) = (t', \phi')$ **et** $t' \leq t$.

Exercice 2 **Typer** $\lambda x.\lambda y.x$, $\lambda x.\lambda y.y$, $\lambda x.\lambda y.xz(yz)$, $\lambda f.\lambda g.\lambda x.f(gx)$.

Exercice 3 **Montrer que** $\lambda x.xx$, $\lambda x.x(y(x))$ **ne sont pas typables.**

Exercice 4 **Montrer que** $\texttt{let}\ f = \lambda x.x\ \texttt{in}\ (ff)\underline{2}$ **est typable, mais que** $(\lambda f.(ff)\underline{2})(\lambda x.x)$ **ne l'est pas.**

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

Exercice 6 **Etendre l'algorithme W aux listes.**

Mémoire

PCF et Références

Termes

M, N, P	$::=$	$\ldots$	voir cours précédents
	$\mid$	$\texttt{ref}\ M$	création
	$\mid$	$!M$	valeur
	$\mid$	$M := N$	modification de valeur
	$\mid$	$()$	valeur vide

Valeurs

V, V'	$::=$	$\underline{n}$	constante entière
	$\mid$	$\lambda x.M$	abstraction
	$\mid$	ℓ	location
	$\mid$	$()$	valeur vide

Exemples

$\texttt{let}\ c = \texttt{ref}\ \underline{0}\ \texttt{in}\ \texttt{let}\ x = c := !c + 1\ \texttt{in}\ !c$

$\texttt{let}\ c = \texttt{ref}\ \underline{0}\ \texttt{in}\ \texttt{let}\ f = (\lambda x.\,\texttt{let}\ y = c := !c + 1\ \texttt{in}\ x)\ \texttt{in}\ f(\underline{4}) + f(\underline{5}) + !c$

$\texttt{let}\ c = \texttt{ref}\ \underline{0}\ \texttt{in}\ \texttt{let}\ f = (\lambda x.\,\texttt{let}\ y = !c\ \texttt{in}\ \texttt{let}\ z = c := x\ \texttt{in}\ y)\ \texttt{in}\ f(\underline{2}) + f(\underline{3})$

Etat mémoire

$$s \quad ::= \quad [\ell_1 = V_1, \ell_2 = V_2, \ldots, \ell_n = V_n] \qquad \textbf{(}\ell_i \textbf{ tous distincts)}$$

$$\mathit{domain}(s) = \{\ell_1, \ell_2, \ldots \ell_n\}$$

Règles de réductions

β	$\langle (\lambda x.M)V,\ s\rangle \rightarrow \langle M[x\backslash V],\ s\rangle$
op	$\langle \underline{m} \otimes \underline{n},\ s\rangle \rightarrow \langle \underline{m \otimes n},\ s\rangle$
cond1	$\langle \mathtt{ifz}\,\underline{0}\ \mathtt{then}\ M\ \mathtt{else}\ N,\ s\rangle \rightarrow \langle M,\ s\rangle$
cond2	$\langle \mathtt{ifz}\,\underline{n}\ \mathtt{then}\ M\ \mathtt{else}\ N,\ s\rangle \rightarrow \langle N,\ s\rangle \qquad (n \neq 0)$
μ	$\langle \mu x.M,\ s\rangle \rightarrow \langle M[x\backslash \mu x.M],\ s\rangle$
β_ℓ	$\langle \mathtt{let}\ x = V\ \mathtt{in}\ N,\ s\rangle \rightarrow \langle N[x\backslash V],\ s\rangle$
alloc	$\langle \mathtt{ref}\ V,\ s\rangle \rightarrow \langle \ell,\ s + [\ell = V]\rangle \qquad (\ell \notin \mathit{domain}(s))$
deref	$\langle !\ell,\ s\rangle \rightarrow \langle s(\ell),\ s\rangle$
assign	$\langle \ell := V,\ s\rangle \rightarrow \langle (),\ s[\ell\backslash V]\rangle$

On passe aux contextes les réductions précédentes. Ainsi

$$\langle M,\ s\rangle \rightarrow \langle M',\ s'\rangle \quad \Rightarrow \quad \langle C[M],\ s\rangle \rightarrow \langle C[M'],\ s'\rangle$$

Exemple

$$\begin{aligned}
&\langle \texttt{let}\ c = \texttt{ref}\ \underline{0}\ \texttt{in}\ \texttt{let}\ x = c := !c + \underline{1}\ \texttt{in}\ !c,\ \ \rangle \\
&\qquad \to \langle \texttt{let}\ c = \ell\ \texttt{in}\ \texttt{let}\ x = c := !c + \underline{1}\ \texttt{in}\ !c,\ [\ell = \underline{0}]\rangle \\
&\qquad \to \langle \texttt{let}\ x = \ell := !\ell + \underline{1}\ \texttt{in}\ !\ell,\ [\ell = \underline{0}]\rangle \\
&\qquad \to \langle \texttt{let}\ x = \ell := \underline{0} + \underline{1}\ \texttt{in}\ !\ell,\ [\ell = \underline{0}]\rangle \\
&\qquad \to \langle \texttt{let}\ x = \ell := \underline{1}\ \texttt{in}\ !\ell,\ [\ell = \underline{0}]\rangle \\
&\qquad \to \langle \texttt{let}\ x = ()\ \texttt{in}\ !\ell,\ [\ell = \underline{1}]\rangle \\
&\qquad \to \langle !\ell,\ [\ell = \underline{1}]\rangle \\
&\qquad \to \langle \underline{1},\ [\ell = \underline{1}]\rangle
\end{aligned}$$

Exercice 7 **Donner les règles d'inférence exactes pour dériver la relation $\langle C[M],\, s\rangle \to \langle C[M'],\, s'\rangle$ à partir de $\langle M,\, s\rangle \to \langle M',\, s'\rangle$.**

Séquence

On dérive les instructions classiques de séquence.

$M \;;\; N$

$\quad \equiv \texttt{let}\ x = M\ \texttt{in}\ N$ **où** $x \notin FV(M)$

$\texttt{while}\ M\ \texttt{do}\ N$

$\quad \equiv \mu x.\,\texttt{ifz}\,M\ \texttt{then}\ ()\ \texttt{else}\ N; x$

$\texttt{for}\ x = M\ \texttt{to}\ N\ \texttt{do}\ P$

$\quad \equiv \texttt{let}\ f = \mu f.\lambda x.\lambda y.\,\texttt{ifz}\ x \ominus y\ \texttt{then}\ P; f(x + \underline{1})\ y\ \texttt{else}\ ()\ \texttt{in}\ f M N$

où $\underline{m} \ominus \underline{n} = \begin{cases} \underline{0} \text{ si } m \leq n \\ \underline{1} \text{ si } m > n \end{cases}$

Remarque: on peut dériver

$$\frac{\langle M,\, s\rangle \to \langle V,\, s'\rangle}{\langle M; N,\, s\rangle \to^{*} \langle N,\, s'\rangle}$$

puisque $\langle M; N,\, s\rangle = \langle \texttt{let}\ x = M\ \texttt{in}\ N,\, s\rangle \to \langle \texttt{let}\ x = V\ \texttt{in}\ N,\, s'\rangle \to \langle N,\, s'\rangle$ comme $x \notin FV(N)$.

Programmation impérative

Dans $M; N$, le résultat de M est sans intérêt (cf. le codage avec `let`). Ce qui importe est de faire M, puis d'évaluer N. Intuitivement, M a un **effet** par exemple sur la mémoire.

Beaucoup de langages de programmation distinguent les expressions dont seuls comptent les effets, les **instructions**, des autres expressions. Leur syntaxe définit séparément expressions et instructions. (cf. Algol, Pascal, Java). Par exemple en C ou Java, une expression e devient l'instruction $e;$ en la post-fixant par point-virgule. Ces langages sont souvent appelés langages **impératifs**.

D'autres langages donnent un résultat à toute expression, même si un type spécial `unit` ou `void` est inventé quand on ne s'intéresse pas vraiment au résultat. Ce sont les langages **applicatifs**. Par exemple Lisp, ML, Haskell.

Remarque: le théorème de Church-Rosser (confluence) disparait avec les références.

Tableaux

Termes

M, N, P	$::=$	$\ldots$	
	$\mid$	$[\mid M_1; M_2; \ldots; M_n \mid]$	**création**
	$\mid$	$M[N]$	**valeur d'un élément**
	$\mid$	$M[N] \leftarrow P$	**modification de valeur**
	$\mid$	$\mathtt{length}\, M$	**taille d'un tableau**

Exemples

$\mathtt{let}\ p = \lambda v.\lambda v'.$
$\quad \mathtt{let}\ m = \mathtt{ref}(0)\ \mathtt{in}$
$\quad \mathtt{for}\ i = 0\ \mathtt{to}\ \mathtt{length}\, v - 1\ \mathtt{do}$
$\qquad m := !m + v[i] * v'[i];$
$\quad !m\ \mathtt{in}$
$p\,[\mid 1; 2; 3 \mid]\ [\mid 10; 20; 30 \mid]$

Règles de réductions pour les tableaux

alloc$_t$ $\langle [|\ V_1; V_2; \ldots V_n\ |],\ s\rangle \rightarrow \langle \ell,\ s + [\,\ell = [|\ V_1; V_2; \ldots V_n\ |]\,]\rangle \quad (\ell \notin \mathit{domain}(s))$

accès $\langle \ell[\underline{m}],\ s\rangle \rightarrow \langle V_i,\ s\rangle$ **si** $s(\ell) = [|\ V_1; V_2; \ldots V_n\ |]$ **et** $0 \leq m < n$

$\qquad\qquad\qquad\qquad$ `err` **sinon**

assign $\langle \ell[\underline{m}] \leftarrow V,\ s\rangle \rightarrow \langle (),\ s[\ell[m] \backslash V]\rangle$ **si** $s(\ell) = [|\ V_1; V_2; \ldots V_n\ |]$ **et** $0 \leq m < n$

$\qquad\qquad\qquad\qquad\qquad$ `err` **sinon**

length $\langle \mathtt{length}\,\ell,\ s\rangle \rightarrow \langle \underline{n},\ s\rangle$ **si** $s(\ell) = [|\ V_1; V_2; \ldots V_n\ |]$

Typage polymorphe des références

$\mathtt{ref}(t)$ et `unit` sont deux nouveaux types.

Les règles de typage polymorphe sont étendues en donnant les types suivants aux 4 nouvelles opérations

$$
\begin{aligned}
() &: \mathtt{unit} \\
\mathtt{ref} &: \forall \alpha . \alpha \rightarrow \mathtt{ref}(\alpha) \\
! &: \forall \alpha . \mathtt{ref}(\alpha) \rightarrow \alpha \\
:= &: \forall \alpha . \mathtt{ref}(\alpha) \rightarrow \alpha \rightarrow \mathtt{unit}
\end{aligned}
$$

On peut donc typer

$$\mathtt{let}\ r = \mathtt{ref\,nil}\ \mathtt{in}\ r := \underline{3} :: \mathtt{nil}; (\mathtt{hd}\ !r)\ \underline{2}$$

$$
\begin{aligned}
r &: \forall \alpha . \mathtt{ref}(\mathtt{list}(\alpha)) \\
r &: \mathtt{ref}(\mathtt{list}(\mathtt{int})) \\
r &: \mathtt{ref}(\mathtt{list}(\mathtt{int} \rightarrow \mathtt{int}))
\end{aligned}
$$

Pourtant il y a une erreur à l'exécution!
Interaction entre effets mémoire et instanciation.

Typage polymorphe des tableaux

Exercice 8 Inventer un système de typage pour les tableaux.

Exercice 9 Montrer que le même problème existe avec les tableaux.

Exercice 10 Trouver un contrexemple au typage avec les références et sans les listes.

Intérêt des types

Le système de types de PCF avec références ne respecte plus le théorème de progression.

Aussi le cas de l'héritage dans Eiffel, ou des tableaux en Java.

Les deux théorèmes de préservation des types (*subject reduction*) et de progression sont deux bons thermomètres pour un système de type. Si non respectés, le type d'un terme peut changer après réduction, ou un terme typé peut se bloquer ou provoquer une erreur avant d'obtenir une valeur. Dans un langage typé, les programmes ne finissent pas par *segmentation fault*.

Remarque: un langage typé n'empêche pas de faire des vérifications de bornes dans l'accès aux éléments d'un tableau ou des tests de division par zéro (exceptions $\neq$ erreurs).

Analyse statique de programmes

Statiquement, c'est à dire avant d'exécuter un programme, on peut essayer de déterminer si l'accès aux tableaux ne fera pas de débordements.

Exemples:

$\texttt{let}\ x = [|\ 4; 5; 6\ |]\ \texttt{in}\ x[2]$
$\texttt{let}\ x = 2\ \texttt{in}\ [|\ 4; 5; 6\ |]\ [x]$
$\texttt{let}\ s = \lambda v.$
$\quad \texttt{let}\ m = \texttt{ref}(0)\ \texttt{in}$
$\quad \texttt{for}\ i = 0\ \texttt{to}\ \texttt{length}\ v - 1\ \texttt{do}$
$\quad\quad m := !m + v[i];$
$\quad !m\ \texttt{in}$
$s\ [|\ 1; 2; 3\ |]$

Il faut faire une analyse statique d'intervalles. Cela peut être compliqué. Inéquations linéaires (cf. Cousot-Halbwachs, Bourdoncle, Deutsch). En tout état de cause, une telle analyse est impossible dans un langage non typé.

Le typage est une première forme d'analyse statique.

En TD

- implémenter l'algorithme W
- ajouter les références à l'évaluateur symbolique de PCF
- finir l'interpréteur et ajouter les références

A la maison et prochaine fois

- Trouver une solution pour le typage des références
- Essayer de comprendre le typage des références en Caml
- Typage des références
- Assertions de correction
- Correction faible, correction forte
- *L-values* et *R-values*
- Alias