Cours 2
Langages fonctionnels typés

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. Déclarations
2. Portée – Liaison lexicale
3. Interpréteurs et environnements
4. Types
5. Typage statique, typage dynamique
6. Vérification de type
7. Extension avec des listes

PCF et déclarations

Termes

$$
\begin{array}{lcll}
M, N, P & ::= & \ldots & \textbf{voir cours précédent} \\
& \mid & \texttt{let}\ x = M\ \texttt{in}\ N & \textbf{déclaration}
\end{array}
$$

Exemples

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

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

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

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

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

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

Déclarations

La notation anonyme des fonctions est souvent cachée dans l'instruction `let`, on écrit plutôt

$$\texttt{let } f\ x = M \texttt{ in } N \quad \textbf{pour} \quad \texttt{let } f = \lambda x.M \texttt{ in } N$$

La déclaration `let` n'est pas non plus nécessaire puisqu'on a

$$\texttt{let } x = M \texttt{ in } N \quad \textbf{identique à} \quad (\lambda x.N)M$$

mais on verra plus tard des différences dans les lois de typage.

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

correspond en ML à

```
let f = function x -> x*x+1 in f 4 + f 5 ;;
```

ou en Java à

```
class Test {
  static int f (int x) { return x * x + 1; }
  public static void main (String[ ] args) {
    System.out.println (f(4) + f(5));
  }
}
```

Blocs — Portée d'une déclaration

```
int f (int x)
{
    return x*x + 1;
}

int main (int argc, char *args[])
{
    return f(4) + f(5);
}
```

Certains langages comme Pascal ou Modula ont des blocs plus structurés autorisant la définition de fonctions locales dans les fonctions. Les blocs peuvent être emboîtés, comme en PCF où un `let` peut être contenu dans tout sous-terme.

Dans $\texttt{let}\ x = M\ \texttt{in}\ N$, la déclaration de la variable locale x porte sur tout N.

Variables locales – globales

$\texttt{let fact} =$

$$\lambda x.\boxed{\begin{array}{l}\texttt{let } f = \\ \quad \mu f.\boxed{\lambda x.\boxed{\lambda r.\boxed{\texttt{ifz } x \texttt{ then } r \texttt{ else } f(x-1)(x*r)}^{r}}^{x}}^{f} \\ \texttt{in} \\ \quad \boxed{f\ x\ 1}^{f}\end{array}}^{x}$$

$\texttt{in}$

$$\boxed{\texttt{fact } \underline{5}}^{\texttt{fact}}$$

En PCF, un bloc correspond à la portée de toute déclaration.

PCF avec déclarations récursives

Termes

$$\begin{array}{lcll} M, N, P & ::= & \ldots & \textbf{voir cours précédent} \\ & | & \texttt{letrec}\ x = M\ \texttt{in}\ N & \textbf{déclaration récursive} \end{array}$$

Nous n'utiliserons pas cette extension, car on peut écrire

$$\texttt{letrec}\ x = M\ \texttt{in}\ N \quad \textbf{pour} \quad \texttt{let}\ x = \mu x.M\ \texttt{in}\ N$$

Exercice 1 **Ecrire l'expression de $\mu x.M$ en terme de `letrec`. L'appliquer au cas de la factorielle.**

Syntaxe abstraite de PCF avec déclaration

```
type nom = string;;

type term =
...
| Let of nom * term * term;;
```

Variables libres

$$FV(\texttt{let } x = M \texttt{ in } N) = FV((\lambda x.N)M) = FV(M) \cup (FV(N) - \{x\})$$

Exercice 2 **Calculer** $FV(\texttt{letrec } x = M \texttt{ in } N)$

Variables liées

$$BV(\texttt{let } x = M \texttt{ in } N) = BV((\lambda x.N)M) = BV(M) \cup BV(N) \cup \{x\}$$

Valeurs (inchangé)

$$
\begin{array}{rcll}
V, V' & ::= & x & \text{variable} \\
& \mid & \lambda x.M & \text{abstraction} \\
& \mid & \underline{n} & \text{constante entière}
\end{array}
$$

Substitution et réductions

On définit $(\mathtt{let}\ y = M\ \mathtt{in}\ N)[x \backslash P]$ à partir de $((\lambda y.N)M)[x \backslash P]$, ainsi que ses réductions.

$$
\begin{array}{lll}
\alpha_\ell & \mathtt{let}\ x = M\ \mathtt{in}\ N \equiv \mathtt{let}\ x' = M\ \mathtt{in}\ N[x \backslash x'] & (x' \notin FV(N)) \\
\beta_\ell & \mathtt{let}\ x = V\ \mathtt{in}\ N \rightarrow N[x \backslash V] &
\end{array}
$$

Remarque

- On exige une valeur pour réduire un `let`

Exercice 3 Expliciter $(\mathtt{let}\ y = M\ \mathtt{in}\ N)[x \backslash P]$

Exercice 4 Donner la règle de réduction de `letrec`

Sémantique opérationnelle de l'appel par valeur

On étend la SOS (*structural operational semantics*) de l'appel par valeur pour tenir compte du `let`

$$\vdash \lambda x.M = \lambda x.M \qquad\qquad \frac{\vdash M[x\backslash\mu x.M] = V}{\vdash \mu x.M = V}$$

$$\frac{\vdash M = \lambda x.P \qquad \vdash N = V' \qquad \vdash P[x\backslash V'] = V}{\vdash M\,N = V}$$

$$\vdash \underline{n} = \underline{n} \qquad\qquad \frac{\vdash M = \underline{m} \qquad \vdash N = \underline{n}}{\vdash M \otimes N = \underline{m \otimes n}}$$

$$\frac{\vdash M = \underline{0} \qquad \vdash M = V}{\vdash \texttt{ifz}\, M \,\texttt{then}\, N \,\texttt{then}\, N' = V} \qquad \frac{\vdash M = \underline{n} \qquad \vdash N = V' \qquad (n \neq 0)}{\vdash \texttt{ifz}\, M \,\texttt{then}\, N \,\texttt{then}\, N' = V'}$$

$$\frac{\vdash M = V' \qquad \vdash N[x\backslash V'] = V}{\vdash \texttt{let}\ x = M\ \texttt{in}\ N = V}$$

Environnements et SOS *bigstep*

On donne un deuxième argument à la fonction d'évaluation, l'environnement ρ qui décrit la liaison des variables.

ρ est une liste d'association entre les noms des variables locales et leur valeur.

On redéfinit les règles de l'appel par valeur

$$\rho \vdash \lambda x.M = \lambda x.M[\rho] \qquad \frac{\rho, x = V \vdash M = V}{\rho \vdash \mu x.M = V}$$

$$\frac{\rho \vdash M = \lambda x.P[\rho'] \qquad \rho \vdash N = V' \qquad \rho', x = V' \vdash P = V}{\rho \vdash M\,N = V}$$

$$\rho \vdash \underline{n} = \underline{n} \qquad \frac{\rho \vdash M = \underline{m} \qquad \rho \vdash N = \underline{n}}{\rho \vdash M \otimes N = \underline{m \otimes n}}$$

$$\frac{\rho \vdash P = \underline{0} \qquad \rho \vdash M = V}{\rho \vdash \texttt{ifz}\, P\, \texttt{then}\, M\, \texttt{then}\, N = V} \qquad \frac{\rho \vdash P = \underline{n} \qquad \rho \vdash N = V \qquad (n \neq 0)}{\rho \vdash \texttt{ifz}\, P\, \texttt{then}\, M\, \texttt{then}\, N = V'}$$

$$\frac{\rho \vdash M = V' \qquad \rho, x = V' \vdash N = V}{\rho \vdash \texttt{let}\ x = M\ \texttt{in}\ N = V} \qquad \rho, x = V \vdash x = V$$

Environnements

On peut aussi voir ρ comme une fonction des noms dans les valeurs.

$$\begin{array}{lcll} V & ::= & x \mid \underline{n} \mid (\lambda x.P)[\rho] & \textbf{valeurs} \\ \rho & ::= & (x_1 = V_1), (x_2 = V_2), \cdots (x_n = V_n) & x_i \textbf{ distincts } (n \geq 0) \end{array}$$

avec $\rho[x\backslash V](x) = V$ **et** $\rho[x\backslash V](y) = \rho(y)$ **sinon.**

$$\rho \vdash \lambda x.M = \lambda x.M[\rho] \qquad \rho \vdash x = \rho(x) \qquad \frac{\rho[x\backslash V] \vdash M = V}{\rho \vdash \mu x.M = V}$$

$$\frac{\rho \vdash M = \lambda x.P[\rho'] \quad \rho \vdash N = V' \quad \rho'[x\backslash V'] \vdash P = V}{\rho \vdash M N = V}$$

$$\rho \vdash \underline{n} = \underline{n} \qquad \frac{\rho \vdash M = \underline{m} \quad \rho \vdash N = \underline{n}}{\rho \vdash M \otimes N = \underline{m \otimes n}}$$

$$\frac{\rho \vdash P = \underline{0} \quad \rho \vdash M = V}{\rho \vdash \texttt{ifz}\, P \texttt{ then } M \texttt{ then } N = V} \qquad \frac{\rho \vdash P = \underline{n} \quad \rho \vdash N = V \quad (n \neq 0)}{\rho \vdash \texttt{ifz}\, P \texttt{ then } M \texttt{ then } N = V}$$

Environnements et récursivité

La règle pour $\mu x.M$ est imprécise. Par exemple $\vdash \mu x.x = V$ pour tout V. Dans cette règle, la valeur est définie en fonction d'un environnement qui est récursif.

On peut voir ρ comme une fonction des noms dans les valeurs.

$$
\begin{array}{lcll}
V & ::= & x \mid \underline{n} \mid (\lambda x.P)[\rho] & \textbf{valeurs} \\
\rho & ::= & (x_1 = V_1), (x_2 = V_2), \cdots (x_n = V_n) & x_i \textbf{ distincts } (n \geq 0) \\
 & & \mid r \mid \mu r.\rho &
\end{array}
$$

On modifie certaines des règles.

$$\rho \vdash \mu x.\underline{n} = \underline{n} \qquad \rho \vdash \mu x.\lambda y.M = \lambda y.M[\mu r.\rho[x \backslash \lambda y.M[r]]]$$

$$\frac{\rho[r \backslash \mu r.\rho] \vdash x = V}{\mu r.\rho \vdash x = V} \qquad \rho \vdash x = \rho(x)$$

Exercices sur les environnements

Exercice 5 Trouver la règle du `letrec`

Exercice 6 Trouver une solution avec des valeurs récursives.

Exercice 7 Refaire la SOS sans environnements récursifs pour la règle μ, mais deux règles pour l'application.

Exercice 8 (en TD) Ecrire en ML la fonction `eval M rho` calculant V si $\rho \vdash M = V$ et où `rho` est une liste d'association représentant l'environnement ρ.

Exercice 9 Faire la SOS pour l'appel par nom avec des environnements, et calculer `eval M rho`.

Types

Langages de programmation typés

En PCF, si on écrit $\underline{2}\,\underline{1}$ ou $\underline{2}(\underline{1})$, c'est une erreur. De même, pour $(\lambda x.x) + \underline{1}$, ou $(\lambda f.\lambda g.\lambda x.f(gx))(\lambda x.x\underline{1})(\lambda x.\underline{2})$

De manière générale, seules les opérations bien définies sur certains domaines sont autorisées. Diffère de l'assembleur (ou de C) où toutes les adresses mémoire sont (ou peuvent être considérées comme) aussi des nombres.

Credo Typage $\Rightarrow$ sureté

Langages fortement typés Caml, Java, Scheme, Lisp

Langages typés Pascal

Langages faiblement typés C, C++

Langages non typés Assembleur

Intérêt des types

- pas de violation mémoire (*core dump, segmentation fault, etc.*)
- pas de comportements indéfinis
- pas de comportements dépendant de la machine

- pas de confusions de valeurs (coercions)
- pas de pointeurs en l'air (*dangling pointer*)
- pas de libération de mémoire occupée, ou de pertes de mémoire libre

- pas d'états bloqués
- pas de transitions hasardeuses
- pas de dépendances du modèle de la machine

Types en PCF

$$\begin{array}{rcll} t, t' & ::= & \texttt{int} & \textbf{les entiers naturels } \mathbb{N} \\ & | & t \to t' & \textbf{les fonctions de } T \textbf{ dans } T' \end{array}$$

Termes de PCF typé

$$\begin{array}{rcll} M, N, P & ::= & x & \textbf{variable} \\ & | & \lambda x : t . M & \textbf{abstraction} \\ & | & M\,N & \textbf{application} \\ & | & \underline{n} & \textbf{constante entière} \\ & | & M \otimes N & \otimes \in \{+, -, \times, \div\} \\ & | & \texttt{ifz}\; M \;\texttt{then}\; N \;\texttt{then}\; N' & \textbf{conditionnelle} \\ & | & \mu x : t . M & \textbf{définition récursive} \\ & | & \texttt{let}\; x : t = M \;\texttt{in}\; N & \textbf{déclaration} \end{array}$$

Abréviation $t \to t' \to t''$ **pour** $t \to (t' \to t'')$

Exemples

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

$\mathtt{let}\ x : \mathtt{int} = \underline{2}\ \mathtt{in}\ \mathtt{let}\ y : \mathtt{int} = x + \underline{1}\ \mathtt{in}\ \underline{2} \times x + y$

$\mathtt{let}\ f : \mathtt{int} \to \mathtt{int} = \lambda x : \mathtt{int}\,.x \times x + \underline{1}\ \mathtt{in}\ f\underline{4} + f\underline{5}$

$\mathtt{let}\ f : \mathtt{int} \to \mathtt{int} = \lambda x : \mathtt{int}\,.x \times x + \underline{1}\ \mathtt{in}\ f(fx)$

$\mathtt{let}\ f : \mathtt{int} \to \mathtt{int} = \lambda x : \mathtt{int}\,.x \times x + \underline{1}\ \mathtt{in}$

$\quad \mathtt{let}\ g : (\mathtt{int} \to \mathtt{int}) \to \mathtt{int} \to \mathtt{int} = \lambda f : \mathtt{int} \to \mathtt{int}\,.\lambda x : \mathtt{int}\,.f(fx)\ \mathtt{in}$

$\qquad g f x$

$\mathtt{let}\ \mathtt{fact} : \mathtt{int} \to \mathtt{int} = \mu f : \mathtt{int} \to \mathtt{int}\,.$

$\quad \lambda x : \mathtt{int}\,.\mathtt{ifz}\, x\ \mathtt{then}\ \underline{1}\ \mathtt{else}\ x \times f(x - \underline{1})\ \mathtt{in}$

$\qquad \mathtt{fact}\,\underline{5}$

Remarques

- la notation $\mathtt{let}\ f(x : t) : t' = M\ \mathtt{in}\ N$ pour $\mathtt{let}\ f : t \to t' = \lambda x : t.M\ \mathtt{in}\ N$ est plus courte

- Idem pour $\mathtt{letrec}\ x : t = M\ \mathtt{in}\ N$ pour $\mathtt{let}\ x : t = \mu x : t.M\ \mathtt{in}\ N$

Plus uniforme de garder l'annotation de type à l'introduction de chaque nouvelle variable.

Règles de typage

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

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

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

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

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

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

Exercice 10 Donner la règle du `letrec`

Quelques propriétés

Théorème 3 **[unicité]** Si $\rho \vdash M : t$ et $\rho \vdash M : t'$, on a $t = t'$

Théorème 4 **[*subject reduction*]** Si $\rho \vdash M : t$ et $M \to N$, on a $\rho \vdash N : t$

Disons que $M \to \texttt{err}$ si M contient un sous-terme de la forme $\underline{n}N$, $(\lambda x.M) \otimes \underline{n}$ ou $\underline{m} \otimes (\lambda x.N)$.

Corollaire 5 **[sureté]** Si $\rho \vdash M : t$ et si $M \to^* N$, alors $N \not\to \texttt{err}$

Théorème 6 **[progression]** Si $\vdash M : t$, alors soit M est une valeur, soit $M \to N$.

Dans le lambda calcul (c'est à dire sans la règle μ),

Théorème 7 **[normalisation forte]** Si $\rho \vdash M : t$, toutes les réductions issues de M sont finies.

Ce dernier théorème est particulièrement dur à démontrer. Une démonstration a été faite par Sanchis 60, une autre par Tait et Martin-Löf.

Correspondance de Curry-Howard

Soient $I = \lambda x.x$, $K = \lambda x.\lambda y.x$ et $S = \lambda x.\lambda y.\lambda z.xz(yz)$.

Les types de I, K et S sont de la forme

$$\vdash I : \alpha \to \alpha$$
$$\vdash K : \alpha \to \beta \to \alpha$$
$$\vdash S : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$$

Or 3 premiers axiomes de la logique propositionnelle sont

$$\vdash A \supset A$$
$$\vdash A \supset B \supset A$$
$$\vdash (A \supset B \supset C) \supset (A \supset B) \supset A \supset C$$

Types $\leftrightarrow$ Formules
Termes $\leftrightarrow$ Preuves !!!

Présentation de PCF typé à la Church

Termes sans annotations de type. Les règles de typage sont

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

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

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

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

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

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

Exercice 11 **Montrer qu'alors le théorème d'unicité n'est plus vrai.**

Typage statique – Typage dynamique

Il est facile d'écrire un programme qui vérifie ou calcule le type d'une expression. On dit que l'expression est typée **statiquement**. Au cours de l'exécution, on peut oublier les types et exécuter **surement** et **efficacement** du PCF.

Certains langages (Lisp, Scheme) ne vérifient les types qu'à l'exécution (**Typage dynamique**). Les termes autorisés sont plus généraux, mais il n'y a aucune garantie d'absence d'erreur à l'exécution. Rien de dramatique ne peut se passer. L'exécution est plus lente. (Une partie de Java fonctionne ainsi).

Exercice 12 Donner un exemple de terme de PCF non typable statiquement, mais typable dynamiquement (sans erreur d'exécution).

Exercice 13 Ecrire un vérificateur de type pour PCF typé.

Extension des types avec des listes

t, t'	$::=$	`int`	les entiers naturels $\mathbf{N}$
	$\mid$	$t \to t'$	les fonctions de T dans T'
	$\mid$	$\mathtt{list}(t)$	listes de T

Termes de PCF typé avec listes

M, N, P	$::=$	$\ldots$	cf. PCF typé
	$\mid$	$\mathtt{hd}\, M$	tête de liste
	$\mid$	$\mathtt{tl}\, M$	queue de liste
	$\mid$	`nil`	liste vide
	$\mid$	$M :: N$	constructeur
	$\mid$	$\mathtt{ifnil}\, M \;\mathtt{then}\; N \;\mathtt{then}\; N'$	cond. sur les listes

Exercice 14 Sait-on écrire simplement des règles de typage pour ce langage?

En TD

- analyseur syntaxique pour PCF donné
- finir l'évaluateur symbolique de PCF
- écrire un 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

La prochaine fois

- équivalence de types
- polymorphisme
- unification et inférence de type