Logiciel certifié: Mini-ML: évaluation et typage

Joëlle Despeyroux

INRIA Sophia-Antipolis

Module
"Méthodes formelles et fiabilité du logiciel"
Dea d'informatique, Nice, automne 2000

Mini-ML - exemples

Mini-ML: λ-calcul avec constantes + polymorphisme (*let*), *case* et fonctions d'ordre supérieur recursives.

Exemples:

$let\ x = 0\ in\ let\ x = s(x)\ in\ x$

$let\ f = lam\ x.x\ in\ (f\ f)$

$$\begin{array}{l} let\ pred = lam\ x.(case\ x\ of\ 0 \Rightarrow 0 \mid s(x') \ \Rightarrow x') \\ in\ let\ twice = lam\ f.lam\ x.(f(f\ x)) \\ \quad in\ ((twice\ pred)\ 2) \end{array}$$

$fix\ double = lam\ x.\ case\ x\ of\ 0 \Rightarrow 0 \mid s(x') \ \Rightarrow s(s(double\ x'))$

$$\begin{array}{l} letrec\ plus = \\ \quad (lam\ x.lam\ y.case\ x\ of\ 0 \Rightarrow y \mid s(x') \ \Rightarrow s(plus\ x'\ y)) \\ in\ (plus\ 2\ 3) \\ \equiv let\ plus = (fix\ Plus = \\ \qquad lam\ x.lam\ y.case\ x\ of\ 0 \Rightarrow y \mid s(x') \ \Rightarrow s(Plus\ x'\ y)) \\ \quad in\ (plus\ 2\ 3) \end{array}$$

Syntaxe

Expressions:

$$
\begin{array}{lll}
exp\ e & ::= & x \mid 0 \mid (s\ e) \mid lam\ x.e \mid (e_1\ e_2) \mid let\ x = e_1\ in\ e_2 \mid \\
 & & fix\ x = e \mid (case\ e_1\ of\ 0 \Rightarrow e_2 \mid (s\ x) \Rightarrow e_3) \mid \\
 & & letrec\ x = e_1\ in\ e_2
\end{array}
$$

$x \in \mathcal{V}ar$

sortes exp, var
sous sortes var $\subset$ exp
constructeurs

$$
\begin{array}{lll}
var & : & \rightarrow \text{var} \\
0 & : & \rightarrow \text{exp} \\
s & : \text{exp} & \rightarrow \text{exp} \\
app & : \text{exp} \times \text{exp} & \rightarrow \text{exp} \\
lam, fix & : \text{var} \times \text{exp} & \rightarrow \text{exp} \\
let, letrec & : \text{exp} \times \text{var} \times \text{exp} & \rightarrow \text{exp} \\
case & : \text{exp} \times \text{exp} \times \text{var} \times \text{exp} & \rightarrow \text{exp}
\end{array}
$$

Note. Dans les expressions $(fix\ x = e)$ et $(letrec\ x = e\ in\ e_1)$, e est une lambda expression.

LF

LF est un système de types pur (PTS), donné par

- Sortes: $\mathcal{S} = \{\text{type}, \text{kind}\}$.
- Axiomes: $\mathcal{A} = \{\text{type} : \text{kind}\}$.
- Règles: $\mathcal{R} = \{(\text{type}, \text{type}), (\text{type}, \text{kind})\}$.

Termes: $M ::= x \mid c \mid \lambda x : M.M' \mid (M\ M') \mid \Pi x : M.M'$

Règles:

$ax \quad \vdash c : s \quad (c : s) \in \mathcal{A}$

$$var\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\ x \notin \Gamma$$

$$aff\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}\ x \notin \Gamma$$

$$lam\ \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash \lambda x : A.M\ :\ \Pi x : A.B}$$

$$app\ \frac{\Gamma \vdash M : \Pi x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B[N/x]}$$

$$prod\ \frac{\Gamma \vdash A\ :\ s_1 \qquad \Gamma \vdash B\ :\ s_2}{\Gamma \vdash \Pi x : A.B\ :\ s_2}\ (s_1, s_2) \in \mathcal{R}$$

$$conv\ \frac{\Gamma \vdash A\ :\ B \qquad \Gamma \vdash B'\ :\ s \qquad B =_\beta B'}{\Gamma \vdash A\ :\ B'}$$

Syntaxe abstraite d'ordre supérieur (hoas)

Church(40): 'on peut représenter toute construction liante d'un langage en utilisant la λ-abstraction'.

Ex. Mini-ML en LF:

$exp : type$

$0 : exp$

$s : exp \to exp$

$app : exp \to exp \to exp$

Fonction de représentation de Mini-ML dans LF:

$\lceil\_\rceil$: Mini-ML $\to$ LF

$\lceil 0 \rceil = 0$

$\lceil s(0) \rceil = (s \ \lceil 0 \rceil)$

$\lceil (e \ e') \rceil = (app \ \lceil e \rceil \ \lceil e' \rceil)$

Les variables de Mini-ML: représentées par les vars de LF.
$\to$ les vars liées dans Mini-ML le seront dans LF.

$\lceil x \rceil = x$

$lam : (exp \to exp) \to exp$
$\lceil lam\ x.e \rceil = (lam\ \lambda x : exp.\lceil e \rceil)$

$let : exp \to (exp \to exp) \to exp$
$\lceil let\ x = e'\ in\ e \rceil = (let\ \lceil e' \rceil\ \lambda x : exp.\lceil e \rceil)$

Hoas et induction: problèmes

Dans CCind, on ne peut pas implémenter le λ-calcul simple par le type suivant:
$Inductive\ L : Set := lam : (L \to L) \to L \mid app : L \to L \to L$

Une telle déclaration est interdite parceque L ne formerait pas un ensemble. D'un point de vue technique, ce qui est interdit est ici la première occurence de L dans le type de lam, parceque c'est une occurence "négative".

Si cette déclaration était permise, on aurait trop de termes. Par exemple, le terme suivant serait un terme légal de L. Or il ne représente aucun terme du λ-calcul.

$(lam\ \lambda x : L.Case\ x\ of$
$\quad (*\ lam\ *)\ \lambda e : (L \to L).(lam\ e)$
$\quad (*\ app\ *)\ \lambda x, y : L.(app\ x\ y))$

On aurait aussi des termes non normalisables (i.e. des termes dont l'évaluation ne termine pas). Par exemple:

$(app\ \Delta\ \Delta)$ où $\Delta = (lam\ \lambda x : L.(app\ x\ x))$.
$(app\ \Delta\ \Delta) \hookrightarrow (app\ \Delta\ \Delta)$.

Hoas et induction: solutions

Il faut distinguer l'espace des fonctions décrivant les termes (de L ou des preuves sur L), de l'espace des fonctions général (comprenant des opérateurs de récursion).

- Dans CCind. Introduire *lam* et *app* sans définir L par un type inductif. Définir les termes corrects par une propriété inductive, qui donnera le principe d'induction voulu (LPAR'94 et TLCA'95),
- dans LF. On décrit bien les langages, sans types inductifs. L'induction et la récursion ne sont pas internalisées (i.e. ne sont pas formalisées au niveau de LF),
- définir une nouvelle théorie des types avec deux "$\Rightarrow$", par exemple basée sur la logique modale. Travaux récents et en cours.

Mini-ML en LF

$\lceil \_ \rceil$: **Mini-ML** $\rightarrow$ **LF**
$\lceil 0 \rceil = 0$
$\lceil s(0) \rceil = (s\ \lceil 0 \rceil)$
$\lceil x \rceil = x$
$\lceil lam\ x.e \rceil = (lam\ \lambda x : exp.\lceil e \rceil)$
$\lceil (e\ e') \rceil = (app\ \lceil e \rceil\ \lceil e' \rceil)$
$\lceil let\ x = e'\ in\ e \rceil = (let\ \lceil e' \rceil\ \lambda x : exp.\lceil e \rceil)$
$\lceil case\ e_1\ of\ 0 \Rightarrow e_2 \mid (s\ x) \Rightarrow e_3 \rceil = (case\ \lceil e_1 \rceil\ \lceil e_2 \rceil\ \lambda x : exp.\lceil e_3 \rceil)$
$\lceil fix\ x = e \rceil = (fix\ \lambda x : exp.\lceil e \rceil)$

Syntaxe:
$exp : type$
$0 : exp$
$s : exp \rightarrow exp$
$lam, fix : (exp \rightarrow exp) \rightarrow exp$
$app : exp \rightarrow exp \rightarrow exp$
$let : exp \rightarrow (exp \rightarrow exp) \rightarrow exp$
$case : exp \rightarrow exp \rightarrow (exp \rightarrow exp) \rightarrow exp$

Exemple:
$\lceil fix\ plus = lam\ x.\ lam\ y.$
$\quad case\ x\ of\ 0\ \Rightarrow\ y \mid (s\ x')\ \Rightarrow s(plus\ x'\ y) \rceil =$
$(fix\ \lambda plus : exp.(lam\ \lambda x : exp.\ (lam\ \lambda y : exp.$
$\quad (case\ x\ y\ (\lambda x' : exp.\ s(plus\ x'\ y))))))$

Formes canoniques

But: adéquation de la représentation d'un langage dans LF.
Remarques:

- $(c\ e_1 \cdots e_n)$ représente un terme $(c\ t_1 \cdots t_n)$
- $(\lambda x : exp.x\ 0) = 0$ représente 0
- $(let\ e\ f) = (let\ e\ \lambda x : exp.(f\ x))$ représente $let\ x = e\ in\ (f\ x)$

Définition. forme canonique = forme '$\beta\eta$-longue':
(def dans λ_C)

- Un terme canonique de type a est soit une variable, soit un terme de la forme $(c\ e_1 \cdots e_n)$ où c est une constante et les e_i sont canoniques.
- Un terme canonique de type $a \rightarrow b$ est de la forme $\lambda x : a.e$ où e est un terme canonique de type b.

Définition M est canonique de type A dans Γ, noté $\Gamma \vdash M \Uparrow A$.
[...]
Thm M est valide dans LF $\Rightarrow \exists M'$ canonique $M' =_{\beta\eta} M$.
Thm $\Gamma \vdash M \Uparrow A \Rightarrow \Gamma \vdash M : A$

Adéquation de la syntaxe

Lemme $\forall \Gamma = x_1 \cdots x_n : exp$ et $\forall e$ expression de Mini-ML tq $FV(e) \subseteq \{x_1 \ldots x_n\}$, $\Gamma \vdash \lceil e \rceil \Uparrow exp$.

Inverse de $\lceil \_ \rceil$:
$\lfloor \_ \rfloor$: LF $\rightarrow$ Mini-ML
$\lfloor 0 \rfloor = 0$
$\lfloor s(0) \rfloor = (s\ \lfloor 0 \rfloor)$
$\lfloor x \rfloor = x$
$\lfloor (lam\ \lambda x : exp.e) \rfloor = lam\ x.\lfloor e \rfloor)$
$\lfloor (app\ e\ e') \rfloor = (\lfloor e \rfloor\ \lfloor e' \rfloor)$
$\lfloor (let\ e'\ \lambda x : exp.e) = let\ x = \ \lfloor e' \rfloor\ in\ \lfloor e \rfloor$
$\lfloor (case\ e_1\ e_2\ \lambda x : exp.e_3) = case\ \lfloor e_1 \rfloor\ of\ 0 \Rightarrow \lfloor e_2 \rfloor \mid (s\ x) \Rightarrow \lfloor e_3 \rfloor$
$\lfloor (fix\ \lambda x : exp.e) = fix\ x = \lfloor e \rfloor$

Lemme $\forall \Gamma = x_1, \cdots x_n : exp\ tq\ \Gamma \vdash e \Uparrow exp$, $\lfloor e \rfloor$ est définie et $\lceil \lfloor e \rfloor \rceil = e$.
Lemme $\forall e$ expression de Mini-ML à vars libres dans $x_1 \cdots x_n$, $\lfloor \lceil e \rceil \rfloor = e$.

Lemme (subst) $\lceil e[e'/x] \rceil = \lceil e \rceil [\lceil e' \rceil / x]$.

Thm (adequation) $\exists \lceil \_ \rceil$ bijection compositionelle (i.e. vérifiant subst) entre les termes de Mini-ML à vars libres dans $\{x_1 \cdots x_n\}$ et les termes (canoniques) M dans LF tq $x_1 \cdots x_n : exp \vdash M \Uparrow exp$.

Évaluation - spécification

valeurs: $valeur\ v ::= 0 \mid (s\ v) \mid lam\ x.e$

évaluation:

$$0 : 0 \hookrightarrow 0 \qquad s : \frac{e \hookrightarrow v}{(s\ e) \hookrightarrow (s\ v)} \qquad lam : lam\ x.e \hookrightarrow lam\ x.e$$

$$app : \frac{e_1 \hookrightarrow lam\ x.e \quad e_2 \hookrightarrow v_2 \quad e[v_2/x] \hookrightarrow v}{(e_1\ e_2) \hookrightarrow v} \quad \text{– appel par valeur}$$

$$let : \frac{e' \hookrightarrow v \qquad e[v/x] \hookrightarrow v'}{let\ x = e'\ in\ e \hookrightarrow v'}$$

$$fix : \frac{e[fix\ x = e/x] \hookrightarrow v}{fix\ x = e \hookrightarrow v}$$

$$case_0 : \frac{e_1 \hookrightarrow 0 \qquad e_2 \hookrightarrow v}{case\ e_1\ of\ 0 \Rightarrow e_2 \mid (s\ x) \Rightarrow e_3 \hookrightarrow v}$$

$$case_s : \frac{e_1 \hookrightarrow (s\ v') \qquad e_3[v'/x] \hookrightarrow v}{case\ e_1\ of\ 0 \Rightarrow e_2 \mid (s\ x) \Rightarrow e_3 \hookrightarrow v}$$

Alternatives:

$$app_n : \frac{e_1 \hookrightarrow lam\ x.e \qquad e[e_2/x] \hookrightarrow v}{(e_1\ e_2) \hookrightarrow v} \quad \text{– appel par nom}$$

$$let_n : \frac{e[e'/x] \hookrightarrow v}{let\ x = e'\ in\ e \hookrightarrow v}$$

Remarque. $(let\ x = e'\ in\ e) \equiv (lam\ x.e\ \ e')$ pour l'évaluation.

Évaluation - exemples

$$\frac{lam\ x.x \hookrightarrow lam\ x.x \qquad 0 \hookrightarrow 0 \qquad x[0/x] \equiv 0 \hookrightarrow 0}{(lam\ x.x\ \ 0) \hookrightarrow 0}\ app$$

Soit $pred = lam\ x.\ (case\ x\ of\ 0 \Rightarrow 0 \mid s(x')\ \ \Rightarrow x')$

$$\frac{pred \hookrightarrow pred \quad \dfrac{0 \hookrightarrow 0}{s(0) \hookrightarrow s(0)} \quad \dfrac{\dfrac{0 \hookrightarrow 0}{s(0) \hookrightarrow s(0)} \quad x'[0/x'] = 0 \hookrightarrow 0}{case\ s(0)\ of\ 0 \Rightarrow 0 \mid s(x') \Rightarrow x' \hookrightarrow 0}}{(pred\ s(0)) \hookrightarrow 0}$$

Soient

$P = lam\ x.lam\ y.case\ x\ of\ 0 \Rightarrow y \mid s(x') \Rightarrow s(plus\ x'\ y))$

$P' = lam\ x.lam\ y.case\ x\ of\ 0 \Rightarrow y \mid s(x') \Rightarrow$
$\quad s((fix\ plus = P)\ x'\ y))$

$P'_2 = lam\ y.case\ 2\ of\ 0 \Rightarrow y \mid s(x') \Rightarrow s((fix\ plus = P)\ x'\ y))$

$$\frac{\dfrac{P[(fix\ plus = P)/plus]] \hookrightarrow P'}{(fix\ plus = P) \hookrightarrow P'} \quad \dfrac{(P'\ 2) \hookrightarrow P'_2 \quad P'_2[3/y] \hookrightarrow 5}{((P'\ 2)\ 3) \hookrightarrow 5}}{let\ plus = (fix\ plus = P)\ in\ (plus\ 2\ 3) \hookrightarrow 5}$$

Théorème: l'évaluation donne une valeur

Rappel: $valeur\ v ::= 0 \mid (s\ v) \mid lam\ x.e$

valeurs:

$val_0 : \quad 0 : val$

$val_s : \quad \dfrac{e : val}{s(e) : val}$

$val_\lambda : \quad lam\ x.e : val$

Thm $\forall e : exp,\ v : valeur.\ e \hookrightarrow v \Rightarrow v : val.$

Preuve. par induction sur la structure de la preuve de $e \hookrightarrow v$:

- cas 0. $0 \hookrightarrow 0$. Par val_0 on a $0 : val$.
- cas s. $\dfrac{e \hookrightarrow v}{(s\ e) \hookrightarrow (s\ v)}$
  Par hyp. d'ind. sur la preuve de $e \hookrightarrow v$ on a $v : val$.
  par val_s on a $(s\ v) : val$.
- cas app. $\dfrac{e_1 \hookrightarrow lam\ x.e \quad e_2 \hookrightarrow v_2 \quad e[v_2/x] \hookrightarrow v}{(e_1\ e_2) \hookrightarrow v}$
  Par hyp. d'ind. sur la preuve de $e[v_2/x] \hookrightarrow v$ on a $v : val$.
- les autres cas sont similaires au cas app.

$\square$

Évaluation - en Sémantique Naturelle

valeurs: $valeur\ v ::= 0 \mid (s\ v) \mid [lam\ x.e; \rho] \mid [fix\ x = e; \rho]$

évaluation:

$$var : \dfrac{\rho \vdash_{val} var\ x \hookrightarrow v}{\rho \vdash var\ x \hookrightarrow v}$$

$$0 : \rho \vdash 0 \hookrightarrow 0 \qquad s : \dfrac{\rho \vdash e \hookrightarrow v}{\rho \vdash (s\ e) \hookrightarrow (s\ v)}$$

$$lam : \rho \vdash lam\ x.e \hookrightarrow [lam\ x.e; \rho]$$

$$app : \dfrac{\rho \vdash e_1 \hookrightarrow [lam\ x.e; \rho_1] \quad \rho \vdash e_2 \hookrightarrow v_2 \quad (\rho_1, x \mapsto v_2) \vdash e \hookrightarrow v}{\rho \vdash (e_1\ e_2) \hookrightarrow v}$$

$$let : \dfrac{\rho \vdash e' \hookrightarrow v' \qquad (\rho, x \mapsto v') \vdash e \hookrightarrow v}{\rho \vdash let\ x = e'\ in\ e \hookrightarrow v}$$

$$fix : \dfrac{(\rho, x \mapsto [fix\ x = e; \rho]) \vdash e \hookrightarrow v}{\rho \vdash fix\ x = e \hookrightarrow v}$$

$$case_0 : \dfrac{\rho \vdash e_1 \hookrightarrow 0 \qquad \rho \vdash e_2 \hookrightarrow v}{\rho \vdash case\ e_1\ of\ 0 \Rightarrow e_2 \mid (s\ x) \Rightarrow e_3 \hookrightarrow v}$$

$$case_s : \dfrac{\rho \vdash e_1 \hookrightarrow (s\ v') \qquad (\rho, x \mapsto v') \vdash e_3 \hookrightarrow v}{\rho \vdash case\ e_1\ of\ 0 \Rightarrow e_2 \mid (s\ x) \Rightarrow e_3 \hookrightarrow v}$$

Évaluation en SN (suite)

Recherche dans l'environnement:

Set val is

$$var_1 : (\rho, x \mapsto v) \vdash x \hookrightarrow v$$

$$var_0 : \frac{\rho \vdash x \hookrightarrow v}{(\rho, y \mapsto v) \vdash x \hookrightarrow v} \quad x \neq y$$

$$fix : \frac{(\rho, y \mapsto [e; (\rho_1, y \mapsto [fix\ y = e; \rho_1])]) \vdash x \hookrightarrow v}{\rho, y \mapsto [fix\ y = e; \rho_1] \vdash x \hookrightarrow v}$$

end val

Alternative pour la récursion - fold

valeurs: sans $[fix\ x = e; \rho]$.

$$valeur\ v ::= 0 \mid (s\ v) \mid [lam\ x.e; \rho]$$

évaluation:

$$fix : \frac{fold(x, e, \rho) \vdash e \hookrightarrow v}{\rho \vdash fix\ x = e \hookrightarrow v}$$

$$val\_fix : \frac{x \mapsto [e; fold(x, e, \rho)].\rho \vdash x \hookrightarrow v}{fold(x, e, \rho) \vdash x \hookrightarrow v}$$

à comparer avec l'évaluation précédente:

$$fix : \frac{(\rho, x \mapsto [fix\ x = e; \rho]) \vdash e \hookrightarrow v}{\rho \vdash fix\ x = e \hookrightarrow v}$$

$$val\_fix : \frac{(\rho, y \mapsto [e; (\rho_1, y \mapsto [fix\ y = e; \rho_1])]) \vdash x \hookrightarrow v}{\rho, y \mapsto [fix\ y = e; \rho_1] \vdash x \hookrightarrow v}$$

Alternative pour la récursion: letrec

expressions: $letrec$ remplace fix.

$$exp\ e\ ::= \cdots \mid letrec\ x = e_1\ in\ e_2$$

valeurs sans $[fix\ x = e; \rho]$:

$$valeur\ v ::= 0 \mid (s\ v) \mid [lam\ x.e; \rho]$$

évaluation:

$$letrec : \frac{(\rho, x \mapsto v_1) \vdash e_1 \hookrightarrow v_1 \qquad (\rho, x \mapsto v_1) \vdash e_2 \hookrightarrow v_2}{\rho \vdash letrec\ x = e_1\ in\ e_2 \hookrightarrow v_2}$$

Note. v_1 est une structure bouclée.

Théorème: l'évaluation donne une valeur

Rappel: $valeur\ v ::= 0 \mid (s\ v) \mid [lam\ x.e, \rho] \mid [fix\ x = e, \rho]$

valeurs:

$val_0 : \quad 0 : val$

$val_s : \quad \dfrac{e : val}{s(e) : val}$

$val_\lambda : \quad [lam\ x.e, \rho] : val$

$val_f : \quad [fix\ x = e, \rho] : val$

Thm $\forall e : exp,\ \rho : env,\ v : valeur.\ \rho \vdash e \hookrightarrow v \Rightarrow v : val.$

Preuve. par induction sur la structure de la preuve de $e \hookrightarrow v$:

- cas 0. $\rho \vdash 0 \hookrightarrow 0$. Par val_0 on a $0 : val$.
- cas s. $\dfrac{\rho \vdash e \hookrightarrow v}{\rho \vdash (s\ e) \hookrightarrow (s\ v)}$
  Par hyp. d'ind. sur la preuve de $\rho \vdash e \hookrightarrow v$ on a $v : val$.
  par val_s on a $(s\ v) : val$.
- cas app.
  $$\frac{\rho \vdash e_1 \hookrightarrow [lam\ x.e; \rho_1] \quad \rho \vdash e_2 \hookrightarrow v_2 \quad (\rho_1, x \mapsto v_2) \vdash e \hookrightarrow v}{\rho \vdash (e_1\ e_2) \hookrightarrow v}$$
  Par hyp. d'ind. sur la preuve de $(\rho_1, x \mapsto v_2) \vdash e \hookrightarrow v$ on a $v : val$.
- les autres cas sont similaires au cas app.

$\square$

Évaluation en LF

$$\begin{array}{lll}
\textbf{eval} & : & exp \to exp \to \mathrm{type} \\
ev_0 & : & (eval\ 0\ 0) \\
ev_s & : & \forall e, v : exp.\ (eval\ e\ v) \to (eval\ (s\ e)(s\ v)) \\
ev_lam & : & \forall e : exp \to exp.\ (eval\ lam\ e\ \ lam\ e) \\
ev_app & : & \forall e_1, e_2, v_2, v : exp.\forall e : exp \to exp.\ (eval\ e_1\ lam\ e) \to \\
 & & (eval\ e_2\ v_2) \to (eval\ (e\ v_2)\ v) \to (eval\ (app\ e_1\ e_2)\ v) \\
ev_let & : & \forall e' : exp.\forall e : exp \to exp.\forall v : exp. \\
 & & (eval\ (e\ e')\ v) \to (eval\ (let\ e'\ e)\ v) \\
ev_fix & : & \forall e : exp \to exp.\forall v : exp.\ (eval\ (e\ (fix\ e))\ v) \to \\
 & & (eval\ (fix\ e)\ v) \\
ev_case_0 & : & \forall e_1, e_2, v : exp.\forall e_3 : exp \to exp.\ (eval\ e_1\ 0) \to \\
 & & (eval\ e_2\ v) \to (eval\ (case\ e_1\ e_2\ e_3)\ v) \\
ev_case_s & : & \forall e_1, e_2, v', v : exp.\forall e_3 : exp \to exp.\ (eval\ e_1\ (s\ v')) \to \\
 & & (eval\ (e_3\ v')\ v) \to (eval\ (case\ e_1\ e_2\ e_3)\ v)
\end{array}$$

Note. dans ev_app: $(e\ v_2)$ implémente $(e\ x)[v_2/x]$.
De même pour let et $case_s$.

Un exemple typique en LF

La β-réduction dans le λ-calcul simple:

$$\begin{array}{lll}
\textbf{red} & : & L \to L \to \mathrm{type} \\
red_\beta & : & \forall e, v : L\ (red\ (app\ (lam\ e)\ v)\ (e\ v)) \\
red_\lambda & : & \forall e, e' : L \to L. \\
 & & (\forall x : L.(red\ x\ x) \to (red\ (e\ x)\ (e'\ x))) \to \\
 & & (red\ (lam\ e)\ (lam\ e')) \\
red_a & : & \forall m, n, m', n' : L.\ (red\ m\ m') \to (red\ n\ n') \to \\
 & & (red\ (app\ m\ n)\ (app\ m'\ n'))
\end{array}$$

Évaluation en LF - exemple 1

$$\frac{lam\ x.x \hookrightarrow lam\ x.x \qquad 0 \hookrightarrow 0 \qquad x[0/x] = 0 \hookrightarrow 0}{(lam\ x.x\ \ 0) \hookrightarrow 0}\ app$$

$(ev\_app\ \_\ \_\ \_\ \_\ \_\ (ev\_lam\ \lambda x : exp.x)\ \ ev\_0\ \ ev\_0) :$
$(eval\ (app\ (lam\ \lambda x.x)\ 0)\ 0)$

Rappel: Isomorphisme de Curry-Howard:

– termes $\equiv$ preuves (dérivations)

– types $\equiv$ propositions

Note: $ev\_0 : (eval\ \ 0\ \ 0) = (eval\ \ (\lambda x : exp.x\ 0)\ \ 0)$.

Évaluation en LF - exemple 2

$$pred = lam\ x.(case\ x\ of\ 0\ \Rightarrow\ 0 \mid (s\ x')\ \Rightarrow x'))$$

$$\frac{pred \hookrightarrow lam\ x.e \qquad \dfrac{0 \hookrightarrow 0}{s(0) \hookrightarrow s(0)} \qquad \dfrac{\dfrac{0 \hookrightarrow 0}{s(0) \hookrightarrow s(0)} \quad 0 \hookrightarrow 0}{case\ s(0)\ of\ 0 \Rightarrow 0 \mid s(x') \Rightarrow 0}}{(pred\ s(0)) \hookrightarrow 0}$$

$$pred = (lam\ \lambda x : exp.(case\ x\ 0\ \lambda x' : exp.x')) = (lam\ e)$$

$(ev\_app\ pred\ e\ (s\ 0)\ (s\ 0)\ 0$
$\quad (ev\_lam\ e)$
$\quad (ev\_s\ 0\ 0\ ev\_0)$
$\quad (ev\_case\_s\ (s\ 0)\ 0\ \lambda x' : exp.x'\ 0\ 0\ (ev\_s\ 0\ 0\ ev\_0)\ ev\_0)) :$
$(eval\ (app\ pred\ (s\ 0))\ 0)$

Adéquation de l'évaluation

$\lceil \_ \rceil$: preuves de $(\hookrightarrow) \rightarrow$ termes de type $(eval\ e\ v)$

$$\lceil \frac{\overset{p}{e \hookrightarrow v}}{(s\ e) \hookrightarrow (s\ v)} \rceil = (ev\_s\ \lceil e \rceil\ \lceil v \rceil\ \lceil p \rceil)$$

Lemme Soient e et v deux expressions Mini-ML fermées et p une preuve de $e \hookrightarrow v$. On a $\vdash \lceil p \rceil \Uparrow (eval\ \lceil e \rceil\ \lceil v \rceil)$.

Lemme $\forall e, v, p$ termes dans LF tq $\vdash e \Uparrow exp$, $\vdash v \Uparrow exp$ et $\vdash p \Uparrow (eval\ e\ v)$, $\exists! e', v'$ expressions dans Mini-ML et $\exists! p'$ preuve de $e \hookrightarrow v$ tq $\lceil e' \rceil = e$, $\lceil v' \rceil = v$ et $\lceil p' \rceil = p$.

Thm (adéquation) $\exists \lceil \_ \rceil$ bijection entre les preuves de $e \hookrightarrow v$ pour e et v expressions fermées de Mini-ML et les termes (canoniques) p dans LF tq $\vdash p \Uparrow (eval\ \lceil e \rceil\ \lceil v \rceil)$

Typage de Damas-Milner (DM)

$$\begin{array}{lll} types & t & ::= \alpha \mid t \rightarrow t' \\ schemas\ de\ types & \sigma & ::= \uparrow t \mid forall\ \alpha.\sigma \end{array}$$

typage

$$var\ \frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$inst\ \frac{\Gamma \vdash e : forall\ \alpha.\sigma}{\Gamma \vdash e : \sigma[t/\alpha]} \qquad inst'\ \frac{\Gamma \vdash e : \uparrow t}{\Gamma \vdash e : t}$$

$$gen\ \frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : forall\ \alpha.\sigma}\ \alpha \notin FV(\Gamma) \qquad gen'\ \frac{\Gamma \vdash e : t}{\Gamma \vdash e : \uparrow t}$$

$$lam\ \frac{\Gamma, x :\uparrow t \vdash e : t'}{\Gamma \vdash lam\ x.e\ :\ t \rightarrow t'}$$

$$app\ \frac{\Gamma \vdash e : t' \rightarrow t \qquad \Gamma \vdash e' : t'}{\Gamma \vdash (e\ e') : t}$$

$$let\ \frac{\Gamma \vdash e' : \sigma' \qquad \Gamma, (x : \sigma') \vdash e : \sigma}{\Gamma \vdash let\ x = e'\ in\ e : \sigma}$$

Typage DM - exemple

$$\dfrac{\dfrac{\dfrac{x : a \vdash x : a}{\vdash lam\ x.x : a \to a}}{\vdash lam\ x.x : \forall a.a \to a} \qquad \dfrac{\dfrac{\Gamma \vdash f : \forall a.a \to a}{\Gamma \vdash f : (b \to b) \to (b \to b)} \qquad \dfrac{\Gamma \vdash f : \forall a.a \to a}{\Gamma \vdash f : b \to b}}{\Gamma \vdash (f\ f) : b \to b}}{\vdash let\ f = lam\ x.x\ in\ (f\ f) : b \to b}$$

où $\Gamma = (f : \forall a.a \to a)$

Typage DM - algorithme

Normalisation des preuves (typage style déduction naturelle):

– on réduit les *gen* immédiatement suivies d'*inst*.

– on fait les *inst* et les *gen* le plus tôt possible.

$$var\ \frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : t} \quad t = inst(\sigma)$$

$$lam\ \frac{\Gamma, x :\ t \vdash e : t'}{\Gamma \vdash lam\ x.e\ :\ t \to t'}$$

$$app\ \frac{\Gamma \vdash e : t' \to t \qquad \Gamma \vdash e' : t'}{\Gamma \vdash (e\ e') : t}$$

$$let\ \frac{\Gamma \vdash e' : t' \qquad \Gamma, (x : \sigma') \vdash e : \sigma}{\Gamma \vdash let\ x = e'\ in\ e : \sigma} \quad \sigma' = gen(\Gamma, t')$$

Typage en Sémantique Naturelle

$$var\ \frac{\Gamma \vdash_{typ} x : \sigma}{\Gamma \vdash x : t}\quad t = inst(\sigma)$$

$$0\ \Gamma \vdash 0 : nat \qquad\qquad s\ \frac{\Gamma \vdash e : nat}{\Gamma \vdash (s\ e) : nat}$$

$$lam\ \frac{\Gamma, x :\ t \vdash e : t'}{\Gamma \vdash lam\ x.e\ :\ t \to t'}$$

$$app\ \frac{\Gamma \vdash e : t' \to t \qquad \Gamma \vdash e' : t'}{\Gamma \vdash (e\ e') : t}$$

$$let\ \frac{\Gamma \vdash e' : t' \qquad \Gamma, (x : \sigma') \vdash e : \sigma}{\Gamma \vdash let\ x = e'\ in\ e : \sigma}\quad \sigma' = gen(\Gamma, t')$$

$$fix\ \frac{\Gamma, x : t \vdash e : t}{\Gamma \vdash fix\ x = e : t}$$

Set *typ* is

$$var_1\ \Gamma, x : \sigma \vdash x : \sigma \qquad\qquad var_0\ \frac{\Gamma \vdash x : \sigma}{\Gamma, y : \sigma' \vdash x : \sigma}\quad x \neq y$$

end *typ*

Typage DM en déduction naturelle

$$inst\ \frac{e : forall\ \alpha.\sigma}{e : \sigma[t/\alpha]} \qquad\qquad inst'\ \frac{e :\uparrow t}{e : t}$$

$$gen\ \frac{e : \sigma}{e : forall\ \alpha.\sigma}\quad (*) \qquad\qquad gen'\ \frac{e : t}{e :\uparrow t}$$

$$lam\ \frac{\begin{array}{c}(x : t)\\ e : t'\end{array}}{lam\ x.e\ :\ t \to t'}\quad (**)$$

$$app\ \frac{e : t' \to t \qquad e' : t'}{(e\ e') : t}$$

$$let\ \frac{e' : \sigma' \qquad \begin{array}{c}(x : \sigma')\\ e : \sigma\end{array}}{let\ x = e'\ in\ e : \sigma}\quad (**)$$

“side conditions”:

$(*)$ α n’est libre dans aucune hypothèse vivante.

$(**)$ x n’apparait libre dans aucune hypothèse vivante.

“parametric judgment”: $lam\ x.e : t \to t'$ dépend de x

“hypothetical judgment”: $e : t'$ dépend de $(x : t)$

Typage DM en LF

$$
\begin{array}{lll}
ty & : & \mathrm{type} \\
sty & : & \mathrm{type} \\
\uparrow & : & ty \to sty \\
exp & : & \mathrm{type} \\
\hookrightarrow & : & ty \to ty \to ty \\
forall & : & (ty \to sty) \to sty \\
lam & : & (exp \to exp) \to exp \\
app & : & exp \to exp \to exp \\
let & : & exp \to (exp \to exp) \to exp \\
type & : & exp \to ty \to \mathrm{type} \\
type_s & : & exp \to sty \to \mathrm{type} \\
inst & : & \forall e : exp.\forall \sigma : (ty \to sty).\forall t : ty. \\
 & & (type_s\ e\ forall\ \sigma) \to (type_s\ e\ \sigma(t)) \\
inst' & : & \forall e : exp.\forall t : ty.\ (type_s\ e\ \uparrow t) \to (type\ e\ t) \\
gen & : & \forall e : exp.\forall \sigma : (ty \to sty). \\
 & & (\forall t : ty.(type_s\ e\ \sigma(t))) \to (type_s\ e\ forall\ \sigma) \\
gen' & : & \forall e : exp.\forall t : ty.\ (type\ e\ t) \to (type_s\ e\ \uparrow t) \\
lam & : & \forall f : exp \to exp.\forall t, t' : ty. \\
 & & (\forall x : exp.(type\ x\ t) \to (type\ (f\ x)\ t')) \to \\
 & & (type\ (lam\ f)\ (t \hookrightarrow t')) \\
app & : & \forall e, e' : exp.\forall t, t' : ty. \\
 & & (type\ e\ (t' \hookrightarrow t)) \to (type\ e'\ t') \to (type\ (app\ e\ e')\ t) \\
let & : & \forall e' : exp.\forall e : exp \to exp.\forall \sigma', \sigma : sty.\ (type_s\ e'\ \sigma') \to \\
 & & (\forall x : exp.(type_s\ x\ \sigma') \to (type_s\ (e\ x)\ \sigma)) \to \\
 & & (type_s\ (let\ e'\ e)\ \sigma)
\end{array}
$$

Typage DM en LF - exemple

$$
\begin{array}{l}
(let\ \_\ \_\ \_\ \_ \\
\quad (gen\ \_\ \lambda a : ty.(a\ to\ a) \\
\quad\quad \lambda a : ty.(lam\ \lambda x : exp.x\ \_\ \_\ \lambda x : exp.\lambda p : (type\ x\ a).p)) \\
\quad \lambda f : exp.\lambda p : (type\ f\ \forall \lambda a : ty.(a\ to\ a)). \\
\quad\quad (app\ \_\ \_\ \_\ \_ \\
\quad\quad\quad (inst\ f\ \lambda a : ty.(a\ to\ a)\ (b \hookrightarrow b)\ p) \\
\quad\quad\quad (inst\ f\ \lambda a : ty.(a\ to\ a)\ b\ p)) \\
: (type\ (let\ (lam\ \lambda x : exp.x)\ \lambda f : exp.(app\ f\ f))\ \ (b \hookrightarrow b))
\end{array}
$$

$$
\dfrac{
\dfrac{
\dfrac{\begin{array}{c}(x : a) \\ x : a\end{array}}{lam\ x.x : a \to a}
}{lam\ x.x : \forall a.a \to a}
\qquad
\dfrac{
\dfrac{\begin{array}{c}(f : \forall a.a \to a) \\ f : \forall a.a \to a\end{array}}{f : (b \to b) \to (b \to b)}
\quad
\dfrac{\begin{array}{c}(f : \forall a.a \to a) \\ f : \forall a.a \to a\end{array}}{f : b \to b}
}{(f\ f) : b \to b}
}{let\ f = lam\ x.x\ in\ (f\ f) : b \to b}
$$

Typage FP (F. Pfenning)

Typage sans schéma de type.

$types\ \ t\ \ ::= nat \mid \alpha \mid t \to t'$

Règles:

$$0\ \frac{}{0\ :\ nat} \qquad s\ \frac{e\ :\ nat}{(s\ e)\ :\ nat}$$

$$lam\ \frac{\begin{array}{c}(x:t)\\ e:t'\end{array}}{lam\ x.e\ :\ t \to t'}\ (*)$$

$$app\ \frac{e:t' \to t \qquad e':t'}{(e\ e'):t}$$

$$let\ \frac{e':t' \quad e[e'/x]:t}{let\ x = e'\ in\ e:t}$$

$$fix\ \frac{\begin{array}{c}(x:t)\\ e:t\end{array}}{fix\ x = e\ :\ t}\ (*)$$

'side conditions':

$(*)$ x n'apparait libre dans aucune hypothèse vivante.

Typage FP en LF

$$\begin{array}{lcl}
ty & : & \mathrm{type} \\
exp & : & \mathrm{type} \\
nat & : & ty \\
\hookrightarrow & : & ty \to ty \to ty \\
lam & : & (exp \to exp) \to exp \\
app & : & exp \to exp \to exp \\
let & : & exp \to (exp \to exp) \to exp \\
type & : & exp \to ty \to \mathrm{type} \\
0 & : & nat \\
s & : & nat \to nat \\
lam & : & \forall f : exp \to exp. \forall t, t' : ty. \\
 & & (\forall x : exp.(type\ x\ t) \to (type\ (f\ x)\ t')) \to \\
 & & (type\ (lam\ f)\ (t \hookrightarrow t')) \\
app & : & \forall e, e' : exp. \forall t, t' : ty. \\
 & & (type\ e\ (t' \hookrightarrow t)) \to (type\ e'\ t') \to (type\ (app\ e\ e')\ t) \\
let & : & \forall e' : exp. \forall e : exp \to exp. \forall t', t : ty. \\
 & & (type\ e'\ t') \to (type\ (e\ e')\ t) \to (type\ (let\ e'\ e)\ t) \\
fix & : & \forall e : exp \to exp. \forall t : ty. \\
 & & (\forall x : exp.(type\ x\ t) \to (type\ (e\ x)\ t)) \to \\
 & & (type\ (fix\ e)\ t)
\end{array}$$

Typage FP en LF - exemple

$$
\begin{array}{l}
(let\ \_\ \_\ \_\ \_\\
\quad (lam\ \lambda x : exp.x\ \_\ \_\ \lambda x : exp.\lambda p : (type\ x\ a).p)\\
\quad (app\ \_\ \_\ \_\ \_\\
\qquad (lam\ \_\ \_\ \_\ \lambda x : exp.\lambda p : (type\ x\ (b \hookrightarrow b)).p)\\
\qquad (lam\ \_\ \_\ \_\ \lambda x : exp.\lambda p : (type\ x\ b).p)\\
: (type\ (let\ (lam\ \lambda x : exp.x)\ \lambda f : exp.(app\ f\ f))\ \ (b \hookrightarrow b))
\end{array}
$$

$$
\dfrac{\dfrac{\begin{matrix}(x:a)\\ x:a\end{matrix}}{lam\ x.x : a \to a} \qquad \dfrac{\dfrac{\begin{matrix}(x:b\to b)\\ x:b\to b\end{matrix}}{lam\ x.x : (b\to b)\to(b\to b)} \quad \dfrac{\begin{matrix}(x:b)\\ x:b\end{matrix}}{lam\ x.x : b\to b}}{(lam\ x.x\ \ lam\ x.x) : b\to b}}{let\ f = lam\ x.x\ in\ (f\ f) : b \to b}
$$

Preuve de la conservation des types (SRT)

Données: les règles d'évaluation (spécification) et les règles de typage FP. Théorème à prouver:

$$(e \hookrightarrow v) \wedge (e : t) \Rightarrow (v : t)$$

Preuve: par induction sur la structure de la preuve.
On a besoin d'un lemme de substitution.

Cas 0.

Hypothèses: $0 \hookrightarrow 0 \quad et \quad 0 : nat$

Preuve: immédiate: $0 : nat$ par hypothèse.

Cas s.

Hypothèses: $\dfrac{e \hookrightarrow v}{(s\ e) \hookrightarrow (s\ v)} \quad et \quad \dfrac{e : nat}{(s\ e) : nat}$

Preuve: immédiate: $v : nat$ par hypothèse d'induction.

D'où $(s\ v) : nat$ par la règle du typage de s.

Cas lam.

Hypothèses: $lam\ x.e \hookrightarrow lam\ x.e \quad et \quad \dfrac{\begin{matrix}(x:t)\\ e:t'\end{matrix}}{lam\ x.e\ :\ t \to t'}$

Preuve: immédiate: $lam\ x.e\ :\ t \to t'$ par hypothèse.

Cas app.

Hypothèses:

$$\frac{e_1 \hookrightarrow lam\ x.e \quad e_2 \hookrightarrow v_2 \quad e[v_2/x] \hookrightarrow v}{(e_1\ e_2) \hookrightarrow v} \qquad \frac{e_1 : t' \to t \quad e_2 : t'}{(e_1\ e_2) : t}$$

Preuve: Par hypothèse d'induction on a:

$lam\ x.e : t' \to t\ [H_1]$ et $v_2 : t'\ [H_2]$.

Le lemme d'inversion de la règle du typage de lam est: $(lam\ x.e : t' \to t) \Rightarrow (x : t' \Rightarrow e : t)\ [inv_{lam}]$.

L'application de inv_{lam} à H_1 nous donne: $x : t' \Rightarrow e : t\ [H_3]$.

On a besoin du lemme de substitution suivant:

Si $v : t'$ et $(x : t' \Rightarrow e : t)$ alors $e[v/x] : t\ [Subst]$.

L'application de $Subst$ à H_2 et H_3 donne $e[v_2/x] : t$.

D'où par hypothèse d'induction: $v : t$.

Cas let.

Hypothèses: $\dfrac{e[e'/x] \hookrightarrow v}{let\ x = e'\ in\ e \hookrightarrow v} \quad et \quad \dfrac{e' : t' \quad e[e'/x] : t}{let\ x = e'\ in\ e : t}$

Preuve: immédiate: $v : t$ par hypothèse d'induction.

Cas fix.

Hypothèses: $\dfrac{e[(fix\ x = e)/x] \hookrightarrow v}{fix\ x = e \hookrightarrow v} \quad et \quad \dfrac{\begin{matrix}(x : t)\\ e : t\end{matrix}}{fix\ x = e\ :\ t}$

Preuve: par hypothèse on a $fix\ x = e\ :\ t$ et $(x : t \Rightarrow e : t)$.

D'où par le lemme de substitution: $e[(fix\ x = e)/x]\ :\ t$.

D'où par hypothèse d'induction: $v : t$.

Preuve de la conservation des types (SRT) sur les spécifications en LF

Données: les règles d'évaluation en LF et les règles de typage FP, en LF. Théorème à prouver dans LF:

$$\forall e, v.\ (eval\ e\ v) \Rightarrow \forall t.\ (type\ e\ t) \Rightarrow (type\ v\ t)$$

Preuve: par induction sur la structure de la preuve de $(eval\ e\ v)$, et inversion de la définition de $type$.
En LF, on n'a plus besoin d'un lemme de substitution.

Cas 0.

Hypothèses: $(eval\ 0\ 0) \quad et \quad (type\ 0\ nat)$

Preuve: immédiate: $(type\ 0\ nat)$ par hypothèse.

Cas s.

Hypothèses: $\begin{matrix}(eval\ e\ v) \to (eval\ (s\ e)\ (s\ v))\ et \\ (type\ e\ nat) \to (type\ (s\ e)\ nat)\end{matrix}$

Preuve: immédiate: $(type\ v\ nat)$ par hypothèse d'induction.

D'où $(type\ \ (s\ v)\ nat)$ par définition du typage de s.

Cas lam.

Hypothèses: $\begin{matrix}(eval\ (lam\ e)\ (lam\ e))\ et \\ \forall t, t' : ty.\ (\forall x : exp.(type\ x\ t) \to (type\ (e\ x)\ t')) \to \\ (type\ (lam\ e)\ (t\ to\ t'))\end{matrix}$

Preuve: immédiate: $(type\ (lam\ e)\ (t\ to\ t'))$ par hypothèse.

Cas app.

Hypothèses:

$(eval\ e_1\ (lam\ e)) \rightarrow (eval\ e_2\ v_2) \rightarrow (eval\ (e\ v_2)\ v) \rightarrow$
$(eval\ (app\ e_1\ e_2)\ v)\ et$
$\forall t, t' : ty.$
$(type\ e_1\ (t'\ to\ t)) \rightarrow (type\ e_2\ t') \rightarrow (type\ (app\ e_1\ e_2)\ t)$

Preuve: par hypothèse d'induction on a:

$(type\ (lam\ e)\ (t'\ to\ t))$ $[H_1]$ et $(type\ v_2\ t')$ $[H_2]$.

Lemme d'inversion: $(type\ (lam\ e)\ (t\ to\ t')) \Rightarrow$
$\forall x : exp.\ (type\ x\ t) \rightarrow (type\ (e\ x)\ t')\ [inv_{lam}]$

L'application de inv_{lam} à H_1 donne:

$\forall x : exp.\ (type\ x\ t') \rightarrow (type\ (e\ x)\ t)\ [H_3]$.

L'application de H_3 à v_2 et à H_2 donne $(type\ (e\ v_2)\ t)$.

D'où par hypothèse d'induction: $(type\ v\ t)$.

Cas let.

Hypothèses:

$(eval\ (e\ e')\ v) \rightarrow (eval\ (let\ e'\ e)\ v) \quad et$
$\forall t', t : ty.\ (type\ e'\ t') \rightarrow (type\ (e\ e')\ t) \rightarrow (type\ (let\ e'\ e)\ t)$

Preuve: immédiate: $(type\ v\ t)$ par hypothèse d'induction.

Cas fix.

Hypothèses:

$(eval\ (e\ (fix\ e))\ v) \rightarrow (eval\ (fix\ e)\ v) \quad et$
$\forall t : ty.\ (\forall x : exp. (type\ x\ t) \rightarrow (type\ (e\ x)\ t)) \rightarrow (type\ (fix\ e)\ t)$

Preuve: la preuve est similaire au cas de l'application:

Par hypothèse on a $(type\ (fix\ e)\ t)$ $[H_1]$ et
$(\forall x : exp.\ (type\ x\ t) \rightarrow (type\ (e\ x)\ t))$ $[H_2]$.

En appliquant H_2 à $(fix\ e)$ et à H_1, on obtient $(type\ (e\ (fix\ e))\ \ t)$.

D'où par hypothèse d'induction: $(type\ v\ t)$.

□

Morale.
La preuve n'a plus besoin du *lemme de substitution*, qui est souvent la partie la plus difficile d'une telle preuve.
Dans le cas *app*, la fonction H_3 représente un jugement hypothétique. L'application de cette fonction remplace l'application du lemme de substitution.
Pour permettre une telle preuve, la méta logique utilisée (ici: LF) doit admettre le *cut* (substitution) comme règle dérivée.

Typage HM (J. Hannan and D. Miller)

Typage sans schéma de type.
Ré-évaluation, mais pas de substitution.

$$types \;\; t \;\; ::= nat \mid \alpha \mid t \to t'$$

Règles:

$$lam \; \frac{\begin{array}{c}(x:t)\\ e:t'\end{array}}{lam\ x.e \;\; : \;\; t \to t'} \;\; (*)$$

$$app \; \frac{e : t' \to t \qquad e' : t'}{(e\ e') : t}$$

$$let \; \frac{e' : t' \qquad \begin{array}{c}(\frac{e' : u}{x : u})\\ e : t\end{array}}{let\ x = e'\ in\ e : t} \;\; (**)$$

'side condition':
$(*)$ x n'apparait libre dans aucune hypothèse vivante.

Typage HM en LF

$$\begin{array}{lcl}
ty & : & \text{type}\\
exp & : & \text{type}\\
nat & : & ty\\
\hookrightarrow & : & ty \to ty \to ty\\
lam & : & (exp \to exp) \to exp\\
app & : & exp \to exp \to exp\\
let & : & exp \to (exp \to exp) \to exp\\
type & : & exp \to ty \to \text{type}\\
lam & : & \forall f : exp \to exp.\forall t, t' : ty.\\
& & (\forall x : exp.(type\ x\ t) \to (type\ (f\ x)\ t')) \to\\
& & (type\ (lam\ f)\ (t \hookrightarrow t'))\\
app & : & \forall e, e' : exp.\forall t, t' : ty.\\
& & (type\ e\ (t' \hookrightarrow t)) \to (type\ e'\ t') \to (type\ (app\ e\ e')\ t)\\
let & : & \forall e' : exp.\forall e : exp \to exp.\forall t', t : ty.\ (type\ e'\ t') \to\\
& & (\forall x : exp.\ (\forall t_1 : ty.\ (type\ e'\ t_1) \to (type\ x\ t_1)) \to\\
& & \quad (type\ (e\ x)\ t)) \to\\
& & (type\ (let\ e'\ e)\ t)
\end{array}$$

Typage HM en LF - exemple

$$
\begin{array}{l}
(let\ \_\ \_\ \_\ \_ \\
\quad (lam\ \lambda x : exp.x\ \_\ \_\ \lambda x : exp.\lambda p : (type\ x\ a).p) \\
\quad \lambda f : exp. \\
\qquad \lambda H : (\forall a' : ty.\ (type\ (lam\ \lambda x : exp.x)\ a') \rightarrow (type\ f\ a')). \\
\qquad\quad (app\ \_\ \_\ \_\ \_ \\
\qquad\qquad (H\ ((b \hookrightarrow b) \hookrightarrow (b \hookrightarrow b))\ \_) \\
\qquad\qquad (H\ (b \hookrightarrow b)\ \_)) \\
: (type\ (let\ (lam\ \lambda x : exp.x)\ \lambda f : exp.(app\ f\ f))\ \ (b \hookrightarrow b))
\end{array}
$$

$$
\dfrac{\dfrac{\begin{array}{c}(x : a)\\ x : a\end{array}}{lam\ x.x : a \rightarrow a}
\qquad
\dfrac{\begin{array}{c}\left(\dfrac{\begin{array}{c}(x : b \rightarrow b)\\ x : b \rightarrow b\end{array}}{lam\ x.x : (b \rightarrow b) \rightarrow (b \rightarrow b)}\right)\\ f : (b \rightarrow b) \rightarrow (b \rightarrow b)\end{array}
\qquad
\begin{array}{c}\left(\dfrac{\begin{array}{c}(x : b)\\ x : b\end{array}}{lam\ x.x : (b \rightarrow b)}\right)\\ f : (b \rightarrow b)\end{array}}{(f\ f) : b \rightarrow b}}{let\ f = lam\ x.x\ in\ (f\ f) : b \rightarrow b}
$$

Bibliographie

References

[CDDK86] D. Clement, J. Despeyroux, Th. Despeyroux, and G. Kahn. A simple applicative language: Mini-ml. In *proceedings of the ACM Symposium on Lisp and Functional Programming, August 1986*, 1986. also available as a Research Report RR-529, Inria, France, May 1986.

[Har90] R. Harper. Systems of polymorphic type assignment in LF. Technical Report CMU-CS-90-144, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 1990.

[HHP87] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *The Second Annual Conference on Logic in Computer Science, LICS,Cornell*, 1987.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Journal of the ACM, 40(1):143-184*, January 1993.

[MP91] S. Michaylov and F. Pfenning. Natural semantics and some of its meta-theory in elf. In Lars Hallnäs, editor, *proceedings of the Second Workshop on Extentions of Logic Programing*, Springer-Verlag LNCS, 1991. also available as a Technical Report MPI–I–91–211, Max-Planck-Institute for Computer Science,Saarbrucken, Germany, August 1991.

[Pfe91] F. Pfenning. Logic programming in the LF logical framework. In G. Huet and G. Plotkin Cambridge University Press, editors, *proceedings of the Workshop on Logical Frameworks, Sophia-Antipolis, France, May 1990*, 1991.