Typage et programmation
Examen 2002-2003

François Pottier
Francois.Pottier@inria.fr

19 décembre 2002

Les questions 7 à 11 sont indépendantes de celles qui précèdent.

1 Présentation du système F

Le système F (également appelé *λ-calcul polymorphe*) est une extension du λ-calcul simplement typé, qui dispose d'une forme de polymorphisme plus puissante que celle de ML, dite *polymorphisme de première classe*. Les types, environnements de typage, expressions, valeurs et contextes d'évaluation du système F sont définis de la façon suivante :

$$\begin{array}{rcl}
\tau & ::= & \alpha \mid \mathrm{int} \mid \tau \rightarrow \tau \mid \forall\alpha.\tau \\
\Gamma & ::= & \bullet \mid \Gamma; x : \tau \\
a & ::= & x \mid k \mid \lambda x : \tau.a \mid a\,a \mid \Lambda\alpha.a \mid a\,\tau \\
w & ::= & k \mid \lambda x : \tau.a \mid \Lambda\alpha.w \\
A & ::= & [\,] \mid A\,a \mid w\,A \mid \Lambda\alpha.A \mid A\,\tau
\end{array}$$

(Nous avons choisi une stratégie d'appel par valeur avec évaluation de gauche à droite, mais d'autres choix sont bien sûr possibles.) La variable α est liée dans le *type universel* $\forall\alpha.\tau$. L'ensemble des variables de types libres dans un type τ ou dans un environnement τ, notés respectivement $\mathrm{ftv}(\tau)$ et $\mathrm{ftv}(\Gamma)$, sont définis en conséquence.

Question 1 *Donnez explicitement les définitions de* $\mathrm{ftv}(\tau)$ *et* $\mathrm{ftv}(\Gamma)$. ◇

La syntaxe des expressions contient des constantes entières $k \in \mathbb{N}$. La variable α est liée dans l'expression $\Lambda\alpha.a$, qui est une *abstraction vis-à-vis d'un type.* Comme toujours, nous considérerons types et expressions comme équivalents modulo α-conversion. L'expression $a\,\tau$ est une *application d'une expression à un type.* Ces deux nouvelles formes d'expressions se combinent de la même façon que la λ-abstraction et l'application habituelles, comme le montrent les règles de réduction :

$$\begin{array}{rcllr}
(\lambda x : \tau.a)\,w & \xrightarrow{\epsilon}_{\mathrm{F}} & a[w/x] & & (\beta\text{-}valeur) \\
(\Lambda\alpha.a)\,\tau & \xrightarrow{\epsilon}_{\mathrm{F}} & a[\tau/\alpha] & & (\beta\text{-}type) \\
A[a] & \rightarrow_{\mathrm{F}} & A[a'] & \text{si } a \xrightarrow{\epsilon}_{\mathrm{F}} a' & (contexte)
\end{array}$$

Nous avons choisi une variante du système F dans laquelle il est permis d'évaluer le corps d'une Λ-abstraction même si celle-ci n'est pas appliquée. Cela formalise l'idée que les types n'ont aucune influence sur le calcul : un calcul paramétré par un type peut être effectué aussi bien avant ou après qu'un type effectif est fourni. Ce détail aura son importance plus loin. Voici à présent les règles de typage du système F.

F-VAR
$$\Gamma \vdash_{\mathrm{F}} x : \Gamma(x)$$

F-CONSTANT
$$\Gamma \vdash_{\mathrm{F}} k : \mathrm{int}$$

F-ABS
$$\frac{\Gamma; x : \tau' \vdash_{\mathrm{F}} a : \tau}{\Gamma \vdash_{\mathrm{F}} \lambda x : \tau'.a : \tau' \to \tau}$$

F-APP
$$\frac{\Gamma \vdash_{\mathrm{F}} a : \tau' \to \tau \qquad \Gamma \vdash_{\mathrm{F}} a' : \tau'}{\Gamma \vdash_{\mathrm{F}} a\, a' : \tau}$$

F-TABS
$$\frac{\Gamma \vdash_{\mathrm{F}} a : \tau \qquad \alpha \notin \mathrm{ftv}(\Gamma)}{\Gamma \vdash_{\mathrm{F}} \Lambda\alpha.a : \forall\alpha.\tau}$$

F-TAPP
$$\frac{\Gamma \vdash_{\mathrm{F}} a : \forall\alpha.\tau}{\Gamma \vdash_{\mathrm{F}} a\, \tau' : \tau[\tau'/\alpha]}$$

Les quatre premières règles sont exactement les mêmes que dans le λ-calcul simplement typé. Dans le cas de la règle F-ABS, l'annotation de types τ' portée par la variable x dans l'expression $\lambda x : \tau'.a$ est utilisée pour imposer le type qui sera associé à x dans la prémisse. Nous présentons donc une variante du système F dans laquelle le type des paramètres de fonctions est déclaré explicitement.

La règle F-TABS indique que, si une expression est abstraite vis-à-vis d'une variable de types α, alors son type sera universellement quantifié vis-à-vis de α. Il s'agit donc d'une règle *d'introduction* du quantificateur universel ou règle de *généralisation*. La seconde prémisse impose que la variable que l'on souhaite généraliser n'apparaisse pas dans l'environnement Γ. Cette condition peut toujours être satisfaite par α-conversion de l'expression $\Lambda\alpha.a$.

La règle F-TAPP indique que, si une expression admet un type universel $\forall\alpha.\tau$, alors elle peut être appliquée à un type τ', ce qui provoque *l'instanciation* de α par τ' dans le type τ. Il s'agit donc d'une règle *d'élimination* du quantificateur universel.

Question 2 *Dans l'environnement vide, donnez une dérivation de typage pour les expressions* $\mathrm{id} = \Lambda\alpha.\lambda x : \alpha.x$, $\mathrm{double} = \Lambda\alpha.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f\,(f\,x)$, *et* $\mathrm{zero} = \mathrm{double}\,\mathrm{int}\,(\mathrm{id}\,\mathrm{int})\,0$. *Indiquez par quelles séries de réductions ces expressions se réduisent en une valeur.* ◇

Question 3 *Quel est le type de l'expression* $\Lambda\alpha.\lambda x : \alpha.y$ *dans l'environnement* $y : \alpha$ *?* ◇

2 Codage de ML dans le système F

L'objectif de cette partie est de montrer que ML peut être vu comme un sous-ensemble du système F. Pour ce faire, nous allons montrer que tout programme bien typé dans ML peut être *annoté* de façon à en faire un programme bien typé dans le système F. Le processus d'annotation consistera principalement à ajouter des abstractions et applications de types explicites.

On rappelle d'abord la définition du langage ML. Les types, schémas de types, environnements de typage, expressions, valeurs et contextes d'évaluation sont définis comme suit :

$$
\begin{array}{rcl}
\tau & ::= & \alpha \mid \text{int} \mid \tau \to \tau \\
\sigma & ::= & \forall \bar{\alpha}.\tau \\
\Gamma & ::= & \bullet \mid \Gamma; x : \sigma \\
e & ::= & x \mid k \mid \lambda x.e \mid e\, e \mid \text{let } x = e \text{ in } e \\
v & ::= & k\ mid \lambda x.e \\
E & ::= & [\,] \mid E\, e \mid v\, E \mid \text{let } x = E \text{ in } e
\end{array}
$$

Les règles de réduction sont les suivantes :

$$
\begin{array}{rcllr}
(\lambda x.e)\, v & \xrightarrow{\epsilon}_{\text{ML}} & e[v/x] & & (\beta) \\
\text{let } x = v \text{ in } e & \xrightarrow{\epsilon}_{\text{ML}} & e[v/x] & & (\mathit{let}) \\
E[e] & \to_{\text{ML}} & E[e] & \text{si } e \xrightarrow{\epsilon}_{\text{ML}} e' & (\mathit{contexte})
\end{array}
$$

Enfin, les règles de typage de ML sont les suivantes :

$$
\textsc{ml-Var}\ \frac{x : \forall \alpha_1 \ldots \alpha_n.\tau \in \Gamma}{\Gamma \vdash_{\text{ML}} x : \tau[\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]}
\qquad
\textsc{ml-Constant}\ \Gamma \vdash_{\text{ML}} k : \text{int}
\qquad
\textsc{ml-Abs}\ \frac{\Gamma; x : \tau' \vdash_{\text{ML}} e : \tau}{\Gamma \vdash_{\text{ML}} \lambda x.e : \tau' \to \tau}
$$

$$
\textsc{ml-App}\ \frac{\Gamma \vdash_{\text{ML}} e : \tau' \to \tau \qquad \Gamma \vdash_{\text{ML}} e' : \tau'}{\Gamma \vdash_{\text{ML}} e\, e' : \tau}
$$

$$
\textsc{ml-Let}\ \frac{\Gamma \vdash_{\text{ML}} e : \tau \qquad \bar{\alpha} = \text{ftv}(\tau) \setminus \text{ftv}(\Gamma) \qquad \Gamma; x : \forall \bar{\alpha}.\tau \vdash_{\text{ML}} e' : \tau'}{\Gamma \vdash_{\text{ML}} \text{let } x = e \text{ in } e' : \tau'}
$$

La règle ml-Var permet l'instanciation du schéma de types associé à x dans Γ. Les règles ml-Abs et ml-App sont celles du λ-calcul simplement typé. L'abstraction n'est pas annotée par un type en ML, car celui-ci est typiquement inféré. La règle ml-Let permet la généralisation du type associé à e.

Les types de ML forment un sous-ensemble des types du système F. Par ailleurs, nous considérerons l'ordre dans lequel apparaissent les quantificateurs dans un schéma de types comme significatif. Cela nous permet de considérer, sans ambiguïté, que les schémas de types de ML forment un sous-ensemble des types du système F ; on lira le schéma de types $\forall \alpha_1 \ldots \alpha_n.\tau$ comme le type $\forall \alpha_1 . \ldots \forall \alpha_n.\tau$. On supposera toujours, dans cette notation, les variables $\alpha_1, \ldots, \alpha_n$ distinctes deux à deux. (Du fait que l'ordre des quantificateurs est significatif, la règle ml-Let est non-déterministe : l'ensemble des variables libres dans τ et non dans Γ est ordonné de façon arbitraire en une séquence $\bar{\alpha}$ pour former le schéma $\forall \bar{\alpha}.\tau$.) Par conséquent, les environnements de typage de ML forment également un sous-ensemble des environnements de typage du système F.

Question 4 *Démontrez que, si $x : \forall \alpha_1 \ldots \alpha_n.\tau \in \Gamma$, alors le jugement $\Gamma \vdash_{\text{F}} x\, \tau_1\, \ldots\, \tau_n : \tau[\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$ est dérivable dans le système F. (Justifiez d'abord pourquoi il est suffisant de considérer le cas où les variables $\alpha_1, \ldots, \alpha_n$ n'apparaissent pas libres dans les types $\tau_1, \ldots, \tau_n$. Vous indiquerez ensuite en quoi cette hypothèse supplémentaire est utile.)* ◇

Question 5 *Démontrez que, si* $\Gamma \vdash_{\mathrm{F}} a : \tau$ *et* $\{\alpha_1, \dots, \alpha_n\} = \mathrm{ftv}(\tau) \setminus \mathrm{ftv}(\Gamma)$, *alors le jugement* $\Gamma \vdash_{\mathrm{F}} \Lambda\alpha_1 \dots . \Lambda\alpha_n . a : \forall\alpha_1 \dots . \forall\alpha_n . \tau$ *est dérivable dans le système F.* ◇

Question 6 *À l'aide de ces deux idées, définissez un prédicat (ou jugement) à quatre arguments* $\Gamma \vdash e \leadsto a : \tau$ *et démontrez qu'il jouit des propriétés suivantes :*

- $\Gamma \vdash e \leadsto a : \tau$ *implique* $\Gamma \vdash_{\mathrm{ML}} e : \tau$ *;*
- *réciproquement,* $\Gamma \vdash_{\mathrm{ML}} e : \tau$ *implique l'existence d'une expression a telle que* $\Gamma \vdash e \leadsto a : \tau$ *;*
- $\Gamma \vdash e \leadsto a : \tau$ *implique* $\Gamma \vdash_{\mathrm{F}} a : \tau$.

Ce nouveau jugement sera naturellement défini par induction sur la forme de e, à l'aide de règles qui enrichissent (en d'autres termes, qui contiennent) les règles de typage de ML. La démonstration des deux premières propriétés ci-dessus sera donc immédiate, et vous vous concentrerez sur la preuve de la troisième. ◇

Ainsi, tout programme bien typé en ML peut être transformé en un programme bien typé dans le système F. Mais ces deux programmes ont-ils la même sémantique ? C'est ce que nous allons à présent vérifier. Nous commençons par prouver que, dans le système F, les annotations de types présentes dans les termes n'ont aucune influence sur le calcul.

Question 7 *Définissez une fonction* $[\![\cdot]\!]$ *qui, à toute expression a du système système F, associe une expression du* λ*-calcul obtenue en supprimant les annotations de types. (On entend par* λ*-calcul le langage ML dépourvu de la construction* let*.) La fonction sera définie par induction sur la structure de l'expression a.* ◇

Question 8 *Vérifiez que, pour toute valeur w,* $[\![w]\!]$ *est une valeur du* λ*-calcul. Vérifiez que* $[\![a[w/x]]\!] = [\![a]\!][[\![w]\!]/x]$ *et* $[\![a[\tau/\alpha]]\!] = [\![a]\!]$*. (On prouve ces deux assertions par induction sur la structure de a ; donnez uniquement les cas où a est une variable ou une* λ*-abstraction.)* ◇

Question 9 *Démontrez que, si* $a \xrightarrow{\epsilon}_{\mathrm{F}} a'$*, alors, dans le* λ*-calcul,* $[\![a]\!]$ *se réduit, en zéro ou une étape, en* $[\![a']\!]$*. (On considère dans cette question, et dans les deux questions suivantes, que le* λ*-calcul est un fragment de ML ; en d'autres termes, la sémantique du* λ*-calcul est donnée par la sémantique de ML.)* ◇

Question 10 *Démontrez que, si A est un contexte d'évaluation du système F, alors* $[\![A]\!]$ *est un contexte d'évaluation du* λ*-calcul. Étendez ainsi le résultat précédent au cas où* $a \to_{\mathrm{F}} a'$. ◇

Question 11 *Déduisez de ces résultats que, si a se réduit en un entier k, alors* $[\![a]\!]$ *se réduit en ce même entier k.* ◇

Si a est bien typée dans le système F, elle doit se réduire en une valeur. En effet, le système F étant sûr (on peut le démontrer par la technique syntaxique standard, c'est-à-dire en établissant les propriétés d'auto-réduction et de progression), elle ne peut avoir un réduit bloqué ; le système F étant fortement normalisant (un résultat dû à Girard), elle ne peut diverger. Le résultat ci-dessus montre que, si une expression a est bien typée, alors a et son effacement $[\![a]\!]$ ont la même sémantique. Les annotations de types n'ont donc aucune influence sur le calcul.

Question 12 *Soit* $\approx$ *la congruence engendrée par l'axiome* $\text{let } x = e_1 \text{ in } e_2 \approx (\lambda x.e_2)\, e_1$. *(On rappelle qu'une congruence est une relation d'équivalence préservée par toutes les constructions du langage.) Démontrez que* $\Gamma \vdash e \rightsquigarrow a : \tau$ *implique* $e \approx [\![a]\!]$. $\diamond$

Ainsi, si a est le codage de e, alors l'effacement de a n'est autre (modulo $\approx$) que e lui-même. Puisque l'effacement préserve la sémantique, et en admettant que $\approx$ préserve la sémantique, nous avons donc vérifié que le codage de ML vers le système F préserve la sémantique.