Langages de programmation: sémantique et typage
Examen partiel 2004-2005

François Pottier

7 janvier 2005

Toutes les questions, à l'exception des questions 5 et 6, peuvent être traitées sans avoir nécessairement répondu à celles qui précèdent.

Présentation du système F

Le système F ou λ-*calcul polymorphe* est une extension du λ-calcul simplement typé. Les types, environnements de typage, expressions et contextes y sont définis de la façon suivante :

$$
\begin{array}{rcl}
\tau & ::= & \alpha \mid \tau \to \tau \mid \forall\alpha.\tau \\
\Gamma & ::= & \bullet \mid \Gamma; x : \tau \\
e & ::= & x \mid \lambda(x:\tau).e \mid e\ e \mid \Lambda\alpha.e \mid e\ \tau \\
C & ::= & [] \mid \lambda(x:\tau).C \mid C\ e \mid e\ C \mid \Lambda\alpha.C \mid C\ \tau
\end{array}
$$

La variable α est liée dans l'expression $\Lambda\alpha.e$, qui est une *abstraction vis-à-vis d'un type.* L'expression $e\ \tau$ est une *application d'une expression à un type.* Ces deux constructions se combinent de la même façon que la λ-abstraction et l'application habituelles, comme le montrent les règles de réduction :

$$
\begin{array}{rcllr}
(\lambda(x:\tau).e_1)\ e_2 & \longrightarrow & [e_2/x]e_1 & & (\beta) \\
(\Lambda\alpha.e)\ \tau & \longrightarrow & [\tau/\alpha]e & & (\beta\textit{-type}) \\
C[e] & \longrightarrow & C[e'] & \text{si } e \longrightarrow e' & (\textit{contexte})
\end{array}
$$

La méta-variable C dénote un contexte arbitraire : aucune stratégie d'évaluation n'est imposée.

Voici à présent les règles de typage du système F :

$$
\textsc{Var}\quad \Gamma \vdash x : \Gamma(x)
\qquad
\textsc{Abs}\quad \frac{\Gamma; x:\tau' \vdash e : \tau}{\Gamma \vdash \lambda(x:\tau').e : \tau' \to \tau}
\qquad
\textsc{App}\quad \frac{\Gamma \vdash e : \tau' \to \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash e\ e' : \tau}
$$

$$
\textsc{TAbs}\quad \frac{\Gamma \vdash e : \tau \qquad \alpha \notin \mathrm{ftv}(\Gamma)}{\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}
\qquad
\textsc{TApp}\quad \frac{\Gamma \vdash e : \forall\alpha.\tau}{\Gamma \vdash e\ \tau' : [\tau'/\alpha]\tau}
$$

Les trois premières règles sont exactement les mêmes que dans le λ-calcul simplement typé. Dans le cas de la règle Abs, l'annotation de types τ' portée par la variable x dans l'expression $\lambda(x:\tau').e$ 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 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 règle 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 le remplacement de α par τ' dans le type τ. Il s'agit donc d'une règle *d'élimination* du quantificateur universel ou règle *d'instanciation.*

Quelques questions élémentaires

Question 1 *Considérons le terme*

$$(\Lambda\alpha.\lambda(f:\beta\to\alpha).f\ x)\ (\alpha\to\beta)\ (\lambda(x:\beta).\lambda(y:\alpha).x)$$

Quelles en sont les variables de termes et variables de types libres ? Dans quel(s) environnement(s) est-il bien typé et quel est alors son type ? Donner une séquence de réductions issue de ce terme et conduisant à un terme irréductible. ◇

Solution. L'unique variable libre de ce terme est x, dont l'occurrence située la plus à gauche est libre. Ses variables de types libres sont β, dont toutes les occurrences sont libres, et α, dont les deux occurrences de droite sont libres.

La variable x étant libre, ce terme ne peut être bien typé que dans un environnement dont le domaine contient x. De plus, puisque le type de f est $\beta\to\alpha$, et puisque l'application $f\ x$ doit être bien typée, il est nécessaire que le type associé à x dans l'environnement soit β. Il faut donc que l'environnement Γ vérifie $\Gamma(x)=\beta$.

On peut à présent vérifier que cette condition est suffisante, en construisant une dérivation de typage pour ce terme et pour un tel environnement Γ. On ne donnera pas ici cette dérivation *in extenso*. En bref, on peut raisonner comme suit. D'après Var, App, Abs et TAbs, la Λ-abstraction de gauche admet le type $\forall\alpha.(\beta\to\alpha)\to\alpha$. Appliquons ensuite TApp. On remplace dans ce type α par $\alpha\to\beta$ et on obtient que l'application de types qui suit admet le type $(\beta\to\alpha\to\beta)\to\alpha\to\beta$. Enfin, par Var et Abs, l'abstraction $\lambda(x:\beta)\ldots.$ admet le type $\beta\to\alpha\to\beta$, donc, par App, le terme complet admet le type $\alpha\to\beta$.

Enfin, l'unique séquence de réductions issue de ce terme et conduisant à un terme irréductible est la suivante :

$$\begin{array}{lll} & (\Lambda\alpha.\lambda(f:\beta\to\alpha).f\ x)\ (\alpha\to\beta)\ (\lambda(x:\beta).\lambda(y:\alpha).x) & \\ \longrightarrow & (\lambda(f:\beta\to\alpha\to\beta).f\ x)\ (\lambda(x:\beta).\lambda(y:\alpha).x) & (\beta\text{-}\mathit{type}) \\ \longrightarrow & (\lambda(x:\beta).\lambda(y:\alpha).x)\ x & (\beta) \\ \longrightarrow & \lambda(y:\alpha).x & (\beta) \end{array}$$

Ajout de paires primitives

Question 2 *Indiquer comment la définition du système F (c'est-à-dire sa syntaxe, sa sémantique, et son système de types) doit être modifiée lorsque l'on souhaite introduire la notion de paire. On pourra considérer l'introduction et l'élimination des paires, au choix, soit comme de nouvelles constructions primitives, soit comme des applications de nouvelles constantes choisies de façon appropriée.* ◇

Solution. Supposons, pour commencer, que l'on considère l'introduction et l'élimination des paires comme de nouvelles constructions primitives. La grammaire des types, des expressions et des contextes est étendue comme suit :

$$\begin{array}{rcl} \tau & ::= & \ldots \mid \tau\times\tau \\ e & ::= & \ldots \mid (e,e) \mid \pi_i\ e \\ C & ::= & \ldots \mid (e,C) \mid (C,e) \mid \pi_i\ C \end{array}$$

L'indice i évolue dans $\{1,2\}$. La sémantique opérationnelle est étendue par l'ajout de la règle

$$\pi_i\ (e_1,e_2)\longrightarrow e_i$$

On ajoute au système de types les règles suivantes :

$$\textsc{Pair}\ \frac{\forall i\in\{1,2\}\quad \Gamma\vdash e_i:\tau_i}{\Gamma\vdash(e_1,e_2):\tau_1\times\tau_2} \qquad \textsc{Proj}\ \frac{\Gamma\vdash e:\tau_1\times\tau_2}{\Gamma\vdash\pi_i\ e:\tau_i}$$

Supposons à présent que l'on préfère introduire de nouvelles constantes. On étend alors la grammaire des types et des expressions comme suit :

$$\begin{array}{rcl} \tau & ::= & \ldots \mid \tau\times\tau \\ e & ::= & \ldots \mid (,) \mid \pi_i \end{array}$$

On ajoute au système de types les règles suivantes :

Pair
$\Gamma \vdash (,) : \forall\alpha_1.\forall\alpha_2.\alpha_1 \to \alpha_2 \to \alpha_1 \times \alpha_2$

Proj
$\Gamma \vdash \pi_i : \forall\alpha_1.\forall\alpha_2.\alpha_1 \times \alpha_2 \to \alpha_i$

Comme on le voit, ces deux constantes admettent des types polymorphes, et on devra par conséquent les appliquer explicitement à des types lorsqu'on souhaitera les utiliser. Cela s'observe, en particulier, dans la nouvelle règle de réduction :

$$\pi_i\ \tau_1\ \tau_2\ ((,)\ \tau_1'\ \tau_2'\ e_1\ e_2) \longrightarrow e_i$$

Notons qu'on n'exige pas l'égalité entre τ_i et τ_i', car on souhaite pouvoir argumenter que les annotations de types n'influent pas sur le calcul, et pourraient en principe être effacées. En réalité, cette égalité sera satisfaite si le rédex est bien typé. □

Ajout de paires par codage

Il est naturel d'ajouter au λ-calcul une notion primitive de paire. Néanmoins, on peut démontrer que cela n'augmente pas l'expressivité de ce calcul. En effet, dans un cadre non typé, on peut considérer l'introduction et l'élimination des paires comme du sucre syntaxique :

$$\begin{array}{rcl} (e_1, e_2) & \equiv & \lambda f.(f\ e_1\ e_2) \\ \pi_i\ e & \equiv & e\ (\lambda x_1.\lambda x_2.x_i) \end{array}$$

Ce codage est dit « à la Church. »

Question 3 *Préciser quelles hypothèses de fraîcheur sont nécessaires pour ce codage. Vérifier, dans ce cadre non typé, que l'expression $\pi_i\ (e_1, e_2)$ a le comportement attendu.* ◇

Solution. On doit choisir f fraîche vis-à-vis de e_1 et e_2. Dans un λ-calcul non typé, cette expression se réduit comme suit :

$$\begin{array}{rll} & \pi_i\ (e_1, e_2) & \\ \equiv & (\lambda f.f\ e_1\ e_2)\ (\lambda x_1.\lambda x_2.x_i) & \\ \longrightarrow & (\lambda x_1.\lambda x_2.x_i)\ e_1\ e_2 & \text{parce que } f \text{ n'apparaît pas dans } e_1 \text{ ou } e_2 \\ \longrightarrow\longrightarrow & e_i & \end{array}$$

ce qui est bien le comportement souhaité. □

Revenons à présent à un cadre typé. Supposons donc donné le système F, *non doté de paires*, et supposons que l'on veuille y coder les paires « à la Church. » On peut se demander si un tel codage est toujours possible et, si oui, quelle en est la définition exacte. Peut-être sera-t-il cette fois nécessaire d'employer des abstractions et applications de types explicites.

Question 4 *Dans le système F non doté de paires, proposer une définition pour les trois notations suivantes :*

$$\begin{array}{rcl} \tau_1 \times \tau_2 & \equiv & \ldots \textit{un type} \ldots \\ ((e_1, e_2) : \tau_1 \times \tau_2) & \equiv & \ldots \textit{une expression} \ldots \\ \pi_i\ (e : \tau_1 \times \tau_2) & \equiv & \ldots \textit{une expression} \ldots \end{array}$$

L'introduction de paire est notée $((e_1, e_2) : \tau_1 \times \tau_2)$, et non simplement (e_1, e_2), car les types τ_1 et τ_2 peuvent apparaître dans l'expansion de ce sucre syntaxique. On suppose donc ces types fournis explicitement. De même, l'élimination de paire est notée $\pi_i\ (e : \tau_1 \times \tau_2)$ et non simplement $\pi_i\ e$. ◇

Solution. L'idée est de conserver le codage initial, donc de traduire (e_1, e_2) par l'expression $\lambda f.(f\ e_1\ e_2)$. Toutefois, nous devons transformer ce terme non typé en un terme typé, donc, au minimum, spécifier le type de f. Il est clair que f doit être une fonction à deux arguments de types respectifs τ_1 et τ_2. Cependant, le type du résultat de f ne doit pas être contraint : ce pourra être tantôt τ_1, tantôt τ_2, lorsqu'on effectue une projection gauche ou droite. Par conséquent, ce type sera une variable α, que l'on quantifiera universellement. Ce raisonnement nous conduit au codage suivant :

$$((e_1, e_2) : \tau_1 \times \tau_2) \equiv \Lambda\alpha.\lambda(f : \tau_1 \to \tau_2 \to \alpha).(f\ e_1\ e_2)$$

On choisit bien sûr α fraîche vis-à-vis de e_1, e_2, τ_1, τ_2 et f fraîche vis-à-vis de e_1, e_2. Le type d'une paire ainsi codée est alors :

$$\tau_1 \times \tau_2 \equiv \forall\alpha.(\tau_1 \to \tau_2 \to \alpha) \to \alpha$$

où, encore une fois, α est choisie fraîche vis-à-vis de τ_1 et τ_2. La projection se fait alors, comme dans le cas non typé, par application à une fonction f appropriée. Cependant, puisqu'une paire est d'abord une abstraction de types, il est nécessaire de réaliser d'abord une application de types :

$$\pi_i\ (e : \tau_1 \times \tau_2) \equiv e\ \tau_i\ (\lambda(x_1 : \tau_1).\lambda(x_2 : \tau_2).x_i)$$

Question 5 *Vérifier que les règles habituelles de typage des paires (question 2) sont admissibles dans le système obtenu par codage (question 4). Une règle de typage*

$$\frac{P}{C}$$

est dite admissible *si et seulement si l'énoncé $P \Rightarrow C$ est vrai.* ◇

Solution. Soit Γ un environnement de typage. Rappelons la définition du codage d'une paire :

$$((e_1, e_2) : \tau_1 \times \tau_2) \equiv \Lambda\alpha.\lambda(f : \tau_1 \to \tau_2 \to \alpha).(f\ e_1\ e_2)$$

La variable de types α, étant choisie fraîche vis-à-vis de e_1, e_2, τ_1, τ_2, peut être librement renommée sans que le membre droit en soit affecté. On peut donc la choisir de surcroît fraîche vis-à-vis de Γ. Par ailleurs, parce que f n'apparaît pas dans e_1 ni e_2, les expressions e_1 et e_2 admettent les mêmes types dans l'environnement Γ et dans l'environnement $\Gamma; f : \tau_1 \to \tau_2 \to \alpha$. Une fois ces remarques formulées, il est facile de vérifier, à l'aide de Var, App, Abs et TAbs, que si pour tout i, l'expression e_i admet le type τ_i dans l'environnement Γ, alors l'expression $((e_1, e_2) : \tau_1 \times \tau_2)$ admet le type $\forall\alpha.(\tau_1 \to \tau_2 \to \alpha) \to \alpha$, c'est-à-dire $\tau_1 \times \tau_2$, dans l'environnement Γ. La règle de typage Pair (question 2) est donc admissible.

Rappelons à présent la définition du codage d'une projection :

$$\pi_i\ (e : \tau_1 \times \tau_2) \equiv e\ \tau_i\ (\lambda(x_1 : \tau_1).\lambda(x_2 : \tau_2).x_i)$$

Supposons que e admet le type $\tau_1 \times \tau_2$, c'est-à-dire $\forall\alpha.(\tau_1 \to \tau_2 \to \alpha) \to \alpha$, dans l'environnement Γ. Alors, parce que α est fraîche vis-à-vis de τ_1 et τ_2 et par TApp, l'expression $e\ \tau_i$ admet le type $(\tau_1 \to \tau_2 \to \tau_i) \to \tau_i$. On en déduit facilement, à l'aide de Var, Abs et App, que l'expression $\pi_i\ (e : \tau_1 \times \tau_2)$ admet le type τ_i. La règle de typage Proj (question 2) est donc admissible. □

Question 6 *Ce codage « à la Church » des paires serait-il possible si on se plaçait non pas dans le cadre du système F mais dans celui de ML ?* ◇

Solution. La notation $\tau_1 \times \tau_2$ doit représenter un type, et non pas un schéma de types. Par conséquent, dans le cadre de ML, la définition de cette notation ne peut plus exploiter la quantification universelle. On pourrait alors poser

$$\tau_1 \times \tau_2 \equiv (\tau_1 \to \tau_2 \to \tau) \to \tau$$

pour un certain τ. Mais le typage de la i-ème projection exigerait alors l'égalité entre τ_i et τ. Il serait donc nécessaire d'exiger l'égalité entre τ_1 et τ_2. On devrait poser alors :

$$\tau \times \tau \equiv (\tau \to \tau \to \tau) \to \tau$$

Ce codage restreint « fonctionne, » mais ne permet de représenter que des paires dont les deux composantes ont le même type, ce qui conduit à une importante perte d'expressivité. On peut donc dire que le codage « à la Church » des paires est impossible en ML. □

Question subsidiaire *Toujours dans le système F, définir un codage similaire pour les sommes.* ◇

Solution. Non fournie. □