Programmation fonctionnelle et systèmes de types (MPRI 2-4-2)

Seconde partie

13 février 2007

Nous nous intéressons à un système de types dédié à la SECD moderne (que l'on appellera, dans ce qui suit, la « machine »). Notre propos est de démontrer que ce système de types est sûr et que le schéma de compilation du λ-calcul simplement typé vers la machine que nous avons étudié en cours préserve le typage.

Chaque question est en principe indépendante de celles qui précèdent.

1 La machine

Cette partie du sujet rappelle plusieurs définitions étudiées en cours.

Voici la syntaxe des différentes composantes de la machine :

$$\begin{array}{lcl} v & ::= & N \mid c[e] \\ e & ::= & \epsilon \mid v.e \\ s & ::= & \epsilon \mid v.s \mid c.e.s \\ c & ::= & \mathsf{const}\, N; c \mid \mathsf{access}\, n; c \mid \mathsf{closure}\, c; c \mid \mathsf{apply}; c \mid \mathsf{return} \mid \mathsf{halt} \end{array}$$

En bref, une valeur v est un entier ou une clôture. Un environnement e est une liste de valeurs. Une pile s est une séquence de valeurs et de continuations formées d'un fragment de code et d'un environnement. Un fragment de code est une liste d'instructions terminée par l'une des instructions return ou halt. Nous avons omis les instructions add, let et endlet par souci de minimalisme. Une configuration $c \,/\, e \,/\, s$ est un triplet d'un fragment de code, un environnement, et une pile.

Voici la sémantique opérationnelle à petits pas de la machine :

$$\begin{array}{rcl} \mathsf{const}\, N; c \,/\, e \,/\, s & \longrightarrow & c \,/\, e \,/\, N.s \\ \mathsf{access}\, n; c \,/\, e \,/\, s & \longrightarrow & c \,/\, e \,/\, e(n).s \\ \mathsf{closure}\, c'; c \,/\, e \,/\, s & \longrightarrow & c \,/\, e \,/\, c'[e].s \\ \mathsf{apply}; c \,/\, e \,/\, v.c'[e'].s & \longrightarrow & c' \,/\, v.e' \,/\, c.e.s \\ \mathsf{return} \,/\, e \,/\, v.c'.e'.s & \longrightarrow & c' \,/\, e' \,/\, v.s \end{array}$$

La notation $e(n)$ dénote la n-ième composante de l'environnement e, pourvu que celle-ci existe.

2 Typage

Le système de types est basé sur les entités syntaxiques suivantes :

$$\begin{array}{lcl} \tau & ::= & \mathit{int} \mid \tau \to \tau \\ E & ::= & \epsilon \mid \tau.E \\ S & ::= & \epsilon \mid \tau.S \mid ?\tau \end{array}$$

En bref, un type de valeurs τ est soit un type de base, soit un type de fonctions. Un type d'environnements E est une liste de types de valeurs. Un type de piles S est une liste de types de valeurs, possiblement terminée par un *type de continuations* de la forme $?\tau$.

Le système de types est constitué de cinq jugements :

$\vdash v : \tau$	—	la valeur v admet le type τ
$\vdash e : E$	—	l'environnement e admet le type E
$\vdash s : S$	—	la pile s admet le type S
$E, S \vdash c$	—	sous un environnement de type E et une pile de type S,
	—	le fragment de code c est correct
$\vdash c \,/\, e \,/\, s$	—	la configuration $c \,/\, e \,/\, s$ est correcte

Voici les règles de typage des valeurs :

$$\textsc{v-Nat}\quad \vdash N : \mathit{int} \qquad\qquad \textsc{v-Closure}\quad \frac{\tau_1.E, ?\tau_2 \vdash c \qquad \vdash e : E}{\vdash c[e] : \tau_1 \to \tau_2}$$

Voici les règles de typage des environnements :

$$\textsc{e-Empty}\quad \vdash \epsilon : \epsilon \qquad\qquad \textsc{e-Value}\quad \frac{\vdash v : \tau \qquad \vdash e : E}{\vdash v.e : \tau.E}$$

Voici les règles de typage des piles :

$$\textsc{s-Empty}\quad \vdash \epsilon : \epsilon \qquad \textsc{s-Value}\quad \frac{\vdash v : \tau \qquad \vdash s : S}{\vdash v.s : \tau.S} \qquad \textsc{s-Cont}\quad \frac{E, \tau.S \vdash c \qquad \vdash e : E \qquad \vdash s : S}{\vdash c.e.s : ?\tau}$$

Voici les règles de typage du code :

$$\textsc{c-Const}\quad \frac{E, \mathit{int}.S \vdash c}{E, S \vdash \mathsf{const}\, N; c} \qquad \textsc{c-Access}\quad \frac{E, E(n).S \vdash c}{E, S \vdash \mathsf{access}\, n; c} \qquad \textsc{c-Closure}\quad \frac{\tau_1.E, ?\tau_2 \vdash c' \qquad E, \tau_1 \to \tau_2.S \vdash c}{E, S \vdash \mathsf{closure}\, c'; c}$$

$$\textsc{c-Apply}\quad \frac{E, \tau_2.S \vdash c}{E, \tau_1.\tau_1 \to \tau_2.S \vdash \mathsf{apply}; c} \qquad \textsc{c-Return}\quad E, \tau.?\tau \vdash \mathsf{return} \qquad \textsc{c-Halt}\quad E, \tau.\epsilon \vdash \mathsf{halt}$$

Voici enfin la règle de typage des configurations :

$$\textsc{Config}\quad \frac{E, S \vdash c \qquad \vdash e : E \qquad \vdash s : S}{\vdash c \,/\, e \,/\, s}$$

Nous souhaiterions à présent démontrer que ce système de types est sûr.

Question 1 (Auto-réduction) *Démontrez que* $\vdash c \,/\, e \,/\, s$ *et* $c \,/\, e \,/\, s \longrightarrow c' \,/\, e' \,/\, s'$ *impliquent* $\vdash c' \,/\, e' \,/\, s'$. ◇

Solution. La démonstration se fait par cas sur la règle de réduction utilisée. Dans chaque cas, nous employons les mêmes notations que lors de la définition de la relation $\longrightarrow$.

∘ *Cas* $\mathsf{const}\, N; c \,/\, e \,/\, s \longrightarrow c \,/\, e \,/\, N.s$. Une dérivation de typage pour la configuration $\mathsf{const}\, N; c \,/\, e \,/\, s$ (le *redex*) est nécessairement de la forme :

$$\textsc{Config}\ \dfrac{\textsc{c-Const}\ \dfrac{E, \mathit{int}.S \vdash c}{E, S \vdash \mathsf{const}\, N; c} \qquad \vdash e : E \qquad \vdash s : S}{\vdash \mathsf{const}\, N; c \,/\, e \,/\, s}$$

À partir des mêmes prémisses, on construit une dérivation de typage pour la configuration $c \,/\, e \,/\, N.s$ (le *réduit*) :

$$\dfrac{E, \mathit{int}.S \vdash c \qquad \vdash e : E \qquad \dfrac{\dfrac{}{\vdash N : \mathit{int}}\ \textsc{v-Nat} \qquad \vdash s : S}{\vdash N.s : \mathit{int}.S}\ \textsc{s-Value}}{\vdash c \,/\, e \,/\, N.s}\ \textsc{Config}$$

∘ *Cas* $\mathsf{access}\, n; c \,/\, e \,/\, s \longrightarrow c \,/\, e \,/\, e(n).s$. Une dérivation de typage pour le redex est nécessairement de la forme :

$$\dfrac{\textsc{c-Access}\ \dfrac{E, E(n).S \vdash c}{E, S \vdash \mathsf{access}\, n; c} \qquad \vdash e : E \qquad \vdash s : S}{\vdash \mathsf{access}\, n; c \,/\, e \,/\, s}\ \textsc{Config}$$

On démontre le lemme auxiliaire suivant : si $e(n)$ est défini et si $\vdash e : E$, alors $E(n)$ est défini et $\vdash e(n) : E(n)$. À partir de ce résultat et à partir des prémisses de la dérivation ci-dessus, on construit une dérivation de typage pour le réduit :

$$\dfrac{E, E(n).S \vdash c \qquad \vdash e : E \qquad \dfrac{\vdash e(n) : E(n) \qquad \vdash s : S}{\vdash e(n).s : E(n).s}\ \textsc{s-Value}}{\vdash c \,/\, e \,/\, e(n).s}\ \textsc{Config}$$

∘ *Cas* $\mathsf{closure}\, c'; c \,/\, e \,/\, s \longrightarrow c \,/\, e \,/\, c'[e].s$. Une dérivation de typage pour le redex est nécessairement de la forme :

$$\dfrac{\textsc{c-Closure}\ \dfrac{\tau_1.E, ?\tau_2 \vdash c' \qquad E, \tau_1 \to \tau_2.S \vdash c}{E, S \vdash \mathsf{closure}\, c'; c} \qquad \vdash e : E \qquad \vdash s : S}{\vdash \mathsf{closure}\, c'; c \,/\, e \,/\, s}\ \textsc{Config}$$

À partir des prémisses de cette dérivation, on construit une dérivation de typage pour le réduit :

$$\dfrac{E, \tau_1 \to \tau_2.S \vdash c \qquad \vdash e : E \qquad \dfrac{\dfrac{\tau_1.E, ?\tau_2 \vdash c' \qquad \vdash e : E}{\vdash c'[e] : \tau_1 \to \tau_2}\ \textsc{v-Closure} \qquad \vdash s : S}{\vdash c'[e].s : \tau_1 \to \tau_2.S}\ \textsc{s-Value}}{\vdash c \,/\, e \,/\, c'[e].s}\ \textsc{Config}$$

∘ *Cas* $\mathsf{apply}; c \,/\, e \,/\, v.c'[e'].s \longrightarrow c' \,/\, v.e' \,/\, c.e.s$. Une dérivation de typage pour le redex est nécessairement de la forme :

$$\dfrac{\textsc{c-Apply}\ \dfrac{E, \tau_2.S \vdash c}{E, \tau_1.\tau_1 \to \tau_2.S \vdash \mathsf{apply}; c} \qquad \vdash e : E \qquad \dfrac{\dfrac{\tau_1.E', ?\tau_2 \vdash c' \qquad \vdash e' : E'}{\vdash c'[e'] : \tau_1 \to \tau_2}\ \textsc{v-Closure} \qquad \dfrac{\vdash v : \tau_1 \qquad \vdash s : S}{\vdash v.c'[e'].s : \tau_1.\tau_1 \to \tau_2.S}\ \textsc{s-Value}^2}{\vdash \mathsf{apply}; c \,/\, e \,/\, v.c'[e'].s}\ \textsc{Config}$$

À partir des prémisses de cette dérivation, on construit une dérivation de typage pour le réduit :

$$\dfrac{\tau_1.E', ?\tau_2 \vdash c' \qquad \dfrac{\vdash v : \tau_1 \qquad \vdash e' : E'}{\vdash v.e' : \tau_1.E'}\ \textsc{e-Value} \qquad \dfrac{E, \tau_2.S \vdash c \qquad \vdash e : E \qquad \vdash s : S}{\vdash c.e.s : ?\tau_2}\ \textsc{s-Cont}}{\vdash c' \,/\, v.e' \,/\, c.e.s}\ \textsc{Config}$$

∘ *Cas* $\mathsf{return} \,/\, e \,/\, v.c'.e'.s \longrightarrow c' \,/\, e' \,/\, v.s$. Une dérivation de typage pour le redex est nécessairement de la forme :

$$\dfrac{\textsc{c-Return}\ \dfrac{}{E, \tau.?\tau \vdash \mathsf{return}} \qquad \vdash e : E \qquad \dfrac{\vdash v : \tau \qquad \dfrac{E', \tau.S \vdash c' \qquad \vdash e' : E' \qquad \vdash s : S}{\vdash c'.e'.s : ?\tau}\ \textsc{s-Cont}}{\vdash v.c'.e'.s : \tau.?\tau}\ \textsc{s-Value}}{\vdash \mathsf{return} \,/\, e \,/\, v.c'.e'.s}\ \textsc{Config}$$

À partir des prémisses de cette dérivation, on construit une dérivation de typage pour le réduit :

$$\dfrac{E', \tau.S \vdash c' \qquad \vdash e' : E' \qquad \dfrac{\vdash v : \tau \qquad \vdash s : S}{\vdash v.s : \tau.S}\ \textsc{s-Value}}{\vdash c' \,/\, e' \,/\, v.s}\ \textsc{Config}$$

Ceci conclut la démonstration. □

Question 2 (Progrès ; sûreté du typage) *Parmi les configurations irréductibles, lesquelles souhaite-t-on considérer comme bloquées, et lesquelles souhaite-t-on considérer comme des résultats ? Donnez un exemple de chaque type de configuration. Énoncez, sans démonstration, les propriétés de progrès et de sûreté du typage.* ◇

Solution. Les configurations bloquées sont les suivantes :

$\mathsf{access}\, n; c \,/\, e \,/\, s$	si $e(n)$ est indéfini
$\mathsf{apply}; c \,/\, e \,/\, s$	si s n'est pas de la forme $v.c'[e'].s'$
$\mathsf{return} \,/\, e \,/\, s$	si s n'est pas de la forme $v.c'.e'.s'$

Les résultats sont les configurations de la forme $\mathsf{halt} \,/\, e \,/\, s$. Ces configurations correspondent à un arrêt normal de la machine. Nous laissons le lecteur vérifier que toute configuration est soit réductible, soit bloquée, soit un résultat.
L'énoncé de progrès est standard : « toute configuration bien typée est soit réductible, soit un résultat ». L'énoncé de sûreté de typage, qui découle des propriétés d'auto-réduction et de progrès, est : « toute configuration bien typée soit se réduit, en zéro ou plusieurs étapes, en un résultat, soit diverge ». □

3 Compilation

Intéressons-nous maintenant à la traduction du λ-calcul simplement typé vers la machine.

Voici la syntaxe du λ-calcul, en notation de de Bruijn :

$$a \quad ::= \quad N \mid n \mid \lambda a \mid a\, a$$

Précisons que N désigne une constante entière et n un indice de de Bruijn.

Voici les règles de typage du λ-calcul simplement typé :

$$E \vdash N : \mathit{int} \qquad E \vdash n : E(n) \qquad \dfrac{\tau_1.E \vdash a : \tau_2}{E \vdash \lambda a : \tau_1 \to \tau_2} \qquad \dfrac{E \vdash a_1 : \tau_1 \to \tau_2 \qquad E \vdash a_2 : \tau_1}{E \vdash a_1\, a_2 : \tau_2}$$

Types τ et types d'environnements E sont comme définis plus haut.

Voici enfin le schéma de compilation du λ-calcul vers la machine. Celui-ci associe à un λ-terme a et à un fragment de code c un nouveau fragment de code, noté $[\![a]\!]; c$.

$$\begin{aligned} [\![N]\!]; c &= \mathsf{const}\, N; c \\ [\![n]\!]; c &= \mathsf{access}\, n; c \\ [\![\lambda a]\!]; c &= \mathsf{closure}\, ([\![a]\!]; \mathsf{return}); c \\ [\![a_1\, a_2]\!]; c &= [\![a_1]\!]; [\![a_2]\!]; \mathsf{apply}; c \end{aligned}$$

La traduction d'un programme clos a est $[\![a]\!]; \mathsf{halt}$.

Nous souhaiterions à présent démontrer que l'image d'un terme bien typé, au sens du λ-calcul simplement typé, est un fragment de code bien typé.

Question 3 (Préservation du typage) *Démontrez que $E \vdash a : \tau$ et $E, \tau.S \vdash c$ impliquent $E, S \vdash [\![a]\!]; c$. Expliquez en quelques phrases ce que signifie intuitivement cet énoncé.* ◇

Solution. La démonstration se fait par induction sur la structure de l'expression a.
∘ *Cas* N. Les hypothèses sont $E \vdash N : \mathit{int}$ et $E, \mathit{int}.S \vdash c$. Pour conclure, il suffit de construire la dérivation :

$$\frac{E, \mathit{int}.S \vdash c}{E, S \vdash \mathsf{const}\, N; c}\ \textsc{c-Const}$$

∘ *Cas* n. Les hypothèses sont $E \vdash n : E(n)$ et $E, E(n).S \vdash c$. Pour conclure, il suffit de construire la dérivation :

$$\frac{E, E(n).S \vdash c}{E, S \vdash \mathsf{access}\, n; c}\ \textsc{c-Access}$$

∘ *Cas* λa. Les hypothèses sont $E \vdash \lambda a : \tau_1 \to \tau_2$ et $E, \tau_1 \to \tau_2.S \vdash c$. Par inversion de la règle de typage de l'abstraction, la première hypothèse implique $\tau_1.E \vdash a : \tau_2$. Par ailleurs, d'après la règle return, nous avons $\tau_1.E, \tau_2.?\tau_2 \vdash \mathsf{return}$. L'hypothèse d'induction, appliquée à ces deux faits, donne $\tau_1.E, ?\tau_2 \vdash [\![a]\!]; \mathsf{return}$. Nous pouvons ainsi construire la dérivation :

$$\frac{\tau_1.E, ?\tau_2 \vdash [\![a]\!]; \mathsf{return} \qquad E, \tau_1 \to \tau_2.S \vdash c}{E, S \vdash \mathsf{closure}\,([\![a]\!]; \mathsf{return}); c}\ \textsc{c-Closure}$$

∘ *Cas* $a_1\, a_2$. Les hypothèses sont $E \vdash a_1\, a_2 : \tau_2$ et $E, \tau_2.S \vdash c$. Par inversion de la règle de typage de l'application, la première hypothèse implique $E \vdash a_1 : \tau_1 \to \tau_2$ et $E \vdash a_2 : \tau_1$. Par application de la règle c-Apply, la seconde hypothèse permet de dériver :

$$E, \tau_1.\tau_1 \to \tau_2.S \vdash \mathsf{apply}; c$$

Par application de l'hypothèse d'induction à a_2, il en découle :

$$E, \tau_1 \to \tau_2.S \vdash [\![a_2]\!]; \mathsf{apply}; c$$

Puis, par application de l'hypothèse d'induction à a_1, il vient :

$$E, S \vdash [\![a_1]\!]; [\![a_2]\!]; \mathsf{apply}; c$$

Ceci conclut la démonstration.
Que signifie intuitivement l'énoncé : « $E \vdash a : \tau$ et $E, \tau.S \vdash c$ impliquent $E, S \vdash [\![a]\!]; c$ » ? On note que le fragment de code c est utilisé en tant que continuation de $[\![a]\!]$. L'environnement de type E et la pile de type $\tau.S$ attendus par c sont donc ceux à la sortie de l'exécution de $[\![a]\!]$. Par ailleurs, l'énoncé est vrai pour tout S, ce qui montre que $[\![a]\!]$ ne fait aucune hypothèse sur le contenu de la pile. On peut donc lire informellement : « si a admet le type τ sous E, alors le fragment de code $[\![a]\!]$ attend pour environnement E, admet une pile arbitraire, et son exécution aura pour effet de pousser une valeur de type τ sur la pile ». □

Question 4 (Sûreté de la chaîne de compilation) *Soit a un programme clos bien typé. Démontrez que, à partir de la configuration $[\![a]\!]; \mathsf{halt} \,/\, \epsilon \,/\, \epsilon$, l'exécution de la machine se déroule sans erreur.* ◇

Solution. Si a est clos et bien typé, alors nous avons $\epsilon \vdash a : \tau$ pour un certain type τ. Par ailleurs, par application de la règle c-Halt, nous avons $\epsilon, \tau.\epsilon \vdash \mathsf{halt}$. D'après le résultat obtenu à la question 3, ceci implique $\epsilon, \epsilon \vdash [\![a]\!]; \mathsf{halt}$, d'où il résulte, par application de la règle Config, que la configuration $[\![a]\!]; \mathsf{halt} \,/\, \epsilon \,/\, \epsilon$ est bien typée. D'après le résultat obtenu à la question 2, l'exécution de la machine à partir de cette configuration se déroule sans erreur. □

4 Optimisation des appels terminaux

Pour optimiser la compilation des appels terminaux, étendons le jeu d'instructions :

$$c \quad ::= \quad \ldots \mid \mathsf{tailapply}$$

et ajoutons une règle de réduction :

$$\mathsf{tailapply} \,/\, e \,/\, v.c'[e'].s \quad \longrightarrow \quad c' \,/\, v.e' \,/\, s$$

Question 5 *Donnez la règle de typage associée à l'instruction* tailapply *et démontrez la sûreté du système ainsi étendu.* ◇

Solution. L'instruction tailapply n'a de sens que si on trouve sur la pile non seulement une valeur de type τ_1 puis une clôture de type $\tau_1 \to \tau_2$, comme dans le cas de apply, mais de plus une continuation, car il faut que le code de la clôture puisse exécuter l'instruction return avec succès. De plus, puisque la clôture produit un résultat de type τ_2, il faut que cette continuation admette le type $?\tau_2$. Une règle naturelle est donc :

$$\textsc{c-Tailapply}\quad E, \tau_1.\tau_1 \to \tau_2.?\tau_2 \vdash \mathsf{tailapply}$$

Le lecteur est invité à réfléchir à la façon dont cette règle correspond à une combinaison des règles c-Apply et c-Return.

La preuve d'auto-réduction est étendue par ajout du cas suivant.

∘ *Cas* $\mathsf{tailapply} \,/\, e \,/\, v.c'[e'].s \longrightarrow c' \,/\, v.e' \,/\, s$. Une dérivation de typage pour le redex est nécessairement de la forme :

$$\textsc{Config}\ \frac{\textsc{c-Tailapply}\ \dfrac{}{E, \tau_1.\tau_1 \to \tau_2.?\tau_2 \vdash \mathsf{tailapply}} \qquad \vdash e : E \qquad \textsc{s-Value}^2\ \dfrac{\textsc{v-Closure}\ \dfrac{\tau_1.E', ?\tau_2 \vdash c' \qquad \vdash e' : E'}{\vdash c'[e'] : \tau_1 \to \tau_2} \qquad \vdash v : \tau_1 \qquad \vdash s : ?\tau_2}{\vdash v.c'[e'].s : \tau_1.\tau_1 \to \tau_2.?\tau_2}}{\vdash \mathsf{tailapply} \,/\, e \,/\, v.c'[e'].s}$$

À partir des prémisses de cette dérivation, on construit une dérivation de typage pour le réduit :

$$\textsc{Config}\ \frac{\tau_1.E', ?\tau_2 \vdash c' \qquad \textsc{e-Value}\ \dfrac{\vdash v : \tau_1 \qquad \vdash e' : E'}{\vdash v.e' : \tau_1.E'} \qquad \vdash s : ?\tau_2}{\vdash c' \,/\, v.e' \,/\, s}$$

Ceci conclut la démonstration. □

Voici le schéma de compilation amélioré :

$$\begin{array}{rcll} (\!|a_1\, a_2|\!) & = & [\![a_1]\!]; [\![a_2]\!]; \mathsf{tailapply} & \\ (\!|a|\!) & = & [\![a]\!]; \mathsf{return} & \text{(dans les autres cas)} \end{array}$$

$$\begin{array}{rcl} [\![N]\!]; c & = & \mathsf{const}\, N; c \\ [\![n]\!]; c & = & \mathsf{access}\, n; c \\ [\![\lambda a]\!]; c & = & \mathsf{closure}\, (\!|a|\!); c \\ [\![a_1\, a_2]\!]; c & = & [\![a_1]\!]; [\![a_2]\!]; \mathsf{apply}; c \end{array}$$

Question 6 *Énoncez, sans démonstration, la propriété de préservation du typage associée à la fonction de compilation en position terminale* $(\!|\cdot|\!)$*. En d'autres termes, donnez un énoncé analogue à celui de la question 3 pour cette fonction.* ◇

Solution. La propriété souhaitée est : « $E \vdash a : \tau$ implique $E, ?\tau \vdash (\!|a|\!)$ ». En termes intuitifs, si l'expression a admet le type τ, alors le fragment de code $(\!|a|\!)$ attend une pile de type $?\tau$, ce qui revient informellement à dire que le code $(\!|a|\!)$ « pousse une valeur de type τ sur la pile puis exécute return ». L'énoncé se démontre simultanément avec celui de la question 3, par induction mutuelle. □