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* $\vdash c' \,/\, e' \,/\, s'$. ◇

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.*◇

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 :

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

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 $[\![a]\!]; c$.

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

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.* ◇

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.* ◇

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.* ◇

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.* ◇