

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

Examen 2007–2008

Xavier Leroy et François Pottier

12 février 2008

Ce problème étudie diverses propriétés de la transformation CPS, notamment la préservation de la sémantique dynamique et la préservation du typage. On signale que les questions sont souvent indépendantes les unes des autres, et que les questions les plus lointaines ne sont pas nécessairement les plus difficiles ou les plus coûteuses en temps.

Le langage considéré est le λ -calcul pur, dont on rappelle la syntaxe :

$$\begin{array}{l} \text{Termes : } a, b ::= x \mid \lambda x. a \mid a b \\ \text{Valeurs : } v ::= x \mid \lambda x. a \end{array}$$

Notons que les variables x sont considérées comme faisant partie des valeurs v .

La sémantique dynamique de ce langage est donnée sous deux formes équivalentes. La sémantique par réduction se présente comme la relation $a \rightarrow a'$, «le terme a se réduit en le terme a' », définie par :

$$(\lambda x. a) v \rightarrow a[x \leftarrow v] \qquad \frac{a \rightarrow a'}{a b \rightarrow a' b} \qquad \frac{b \rightarrow b'}{a b \rightarrow a b'}$$

La sémantique naturelle se présente comme la relation $a \Rightarrow v$, «le terme a s'évalue en la valeur v », définie par :

$$v \Rightarrow v \qquad \frac{a \Rightarrow \lambda x. a' \quad b \Rightarrow v' \quad a'[x \leftarrow v'] \Rightarrow v}{a b \Rightarrow v}$$

Partie I. Transformation CPS et préservation de la sémantique

On considère une transformation CPS, dont la définition est donnée par les équations ci-dessous. La transformation est constituée de deux fonctions définies de façon mutuellement récursive, à savoir la traduction des valeurs $\langle \cdot \rangle$ et la traduction des termes $\llbracket \cdot \rrbracket$.

$$\begin{array}{l} \langle x \rangle = x \\ \langle \lambda x. a \rangle = \lambda x. \llbracket a \rrbracket \\ \llbracket v \rrbracket = \lambda k. k \langle v \rangle \qquad (k \# v) \\ \llbracket a_1 a_2 \rrbracket = \lambda k. \llbracket a_1 \rrbracket (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 x_2 k)) \qquad (k \# a_1, a_2)(x_1 \# k, a_2)(x_2 \# k, x_1) \end{array}$$

On admettra sans démonstration le résultat de commutation ci-dessous entre substitution et transformation CPS :

$$\llbracket a[x \leftarrow v] \rrbracket = \llbracket a \rrbracket [x \leftarrow \langle v \rangle]$$

Le but de cette partie est de donner une preuve élémentaire (n'utilisant pas de monades) du fait que cette transformation CPS préserve la sémantique des programmes.

Question 1 Soit a un terme, v une valeur et $K = \lambda k \dots$ une valeur représentant une continuation. Supposons que $a \Rightarrow v$. Montrer que $\llbracket a \rrbracket K \xrightarrow{*} K \ (v)$. \diamond

Solution. La preuve procède par récurrence sur la dérivation de l'évaluation $a \Rightarrow v$ et par cas sur la forme de l'expression a .

◦ Cas a est une valeur. Nécessairement, $a = v$. On a donc $\llbracket a \rrbracket K = (\lambda k. k \ (v)) K$. On peut donc former la réduction suivante :

$$(\lambda k. k \ (v)) K \rightarrow K \ (v)$$

C'est le résultat attendu.

◦ Cas a est une application $a_1 \ a_2$. L'évaluation de a est donc de la forme

$$\frac{a_1 \Rightarrow \lambda x. a_3 \quad a_2 \Rightarrow v_2 \quad a_3[x \leftarrow v_2] \Rightarrow v}{a_1 \ a_2 \Rightarrow v}$$

On a la réduction suivante :

$$\llbracket a_1 \ a_2 \rrbracket K = (\lambda k. \llbracket a_1 \rrbracket (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 \ x_2 \ k))) K \rightarrow \llbracket a_1 \rrbracket (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 \ x_2 \ K)) = t_1$$

en renommant les variables x_1, x_2 si nécessaire pour qu'elles ne soient pas libres dans la continuation K .

Le terme t_1 est de la forme $\llbracket a_1 \rrbracket K_1$, avec $K_1 = (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 \ x_2 \ K))$. On peut donc appliquer l'hypothèse de récurrence au terme a_1 , à l'évaluation $a_1 \Rightarrow \lambda x. a_3$ et à la continuation K_1 : on obtient

$$t_1 \xrightarrow{*} (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 \ x_2 \ K)) (\lambda x. a_3) = (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 \ x_2 \ K)) (\lambda x. \llbracket a_3 \rrbracket) \rightarrow \llbracket a_2 \rrbracket (\lambda x_2. (\lambda x. \llbracket a_3 \rrbracket) x_2 \ K) = t_2$$

À nouveau, le terme t_2 est de la forme $\llbracket a_2 \rrbracket K_2$ avec $K_2 = (\lambda x_2. (\lambda x. \llbracket a_3 \rrbracket) x_2 \ K)$. On applique à nouveau l'hypothèse de récurrence à l'évaluation $a_2 \Rightarrow v_2$, obtenant

$$t_2 \xrightarrow{*} (\lambda x_2. (\lambda x. \llbracket a_3 \rrbracket) x_2 \ K) (v_2) \rightarrow (\lambda x. \llbracket a_3 \rrbracket) (v_2) K \rightarrow \llbracket a_3 \rrbracket [x \leftarrow (v_2)] K = t_3$$

(Ces réductions sont possibles car la traduction (v_2) est une valeur.) Comme la substitution commute avec la transformation CPS, on a $t_3 = \llbracket a_3[x \leftarrow v_2] \rrbracket K$. On peut donc appliquer une troisième fois l'hypothèse de récurrence à l'évaluation $a_3[x \leftarrow v_2] \Rightarrow v$, obtenant $t_3 \xrightarrow{*} K \ (v)$. Par transitivité de $\xrightarrow{*}$, on obtient le résultat désiré :

$$\llbracket a_1 \ a_2 \rrbracket K \xrightarrow{*} t_1 \xrightarrow{*} t_2 \xrightarrow{*} t_3 \xrightarrow{*} K \ (v)$$

Question 2 Soit a un programme qui s'évalue en la valeur v : $a \Rightarrow v$. Comment s'évalue le programme après transformation CPS, à savoir le terme $\llbracket a \rrbracket (\lambda x. x)$? \diamond

Solution. On applique la question précédente avec $K = \lambda x. x$.

$$\llbracket a \rrbracket (\lambda x. x) \xrightarrow{*} (\lambda x. x) \ (v) \rightarrow (v)$$

Donc, le programme transformé s'évalue en (v) , qui est une valeur. \square

Partie II. Transformation CPS et préservation des types

On suppose les langages source et cible de la transformation simplement typés. La grammaire des types est :

$$\tau ::= \alpha \mid \tau \rightarrow \tau$$

Question 3 On fixe une variable de types α pour représenter le type des réponses, c'est-à-dire le codomaine des continuations. Définir deux fonctions de traduction des types, que nous noterons également (\cdot) et $\llbracket \cdot \rrbracket$, de façon à ce que les énoncés suivants soient satisfaits :

1. si $\Gamma \vdash v : \tau$, alors $(\Gamma) \vdash (v) : (\tau)$;
2. si $\Gamma \vdash a : \tau$, alors $(\Gamma) \vdash \llbracket a \rrbracket : \llbracket \tau \rrbracket$.

Démontrer alors ces deux énoncés. ◇

Solution. La définition est :

$$\begin{aligned} \langle \alpha \rangle &= \alpha \\ \langle \tau_1 \rightarrow \tau_2 \rangle &= \langle \tau_1 \rangle \rightarrow \llbracket \tau_2 \rrbracket \\ \llbracket \tau \rrbracket &= (\langle \tau \rangle \rightarrow o) \rightarrow o \end{aligned}$$

Démontrons à présent les deux énoncés ci-dessus, par induction structurelle mutuelle. Nous avons en tout quatre cas :

◦ *Cas x.* La dérivation de typage originale est :

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \text{VAR}$$

Nous construisons alors :

$$\frac{\langle \Gamma \rangle(x) = \langle \tau \rangle}{\langle \Gamma \rangle \vdash \langle x \rangle : \langle \tau \rangle} \text{VAR}$$

Nous avons exploité les équations $\langle x \rangle = x$ et $\langle \Gamma \rangle(x) = \langle \Gamma(x) \rangle$.

◦ *Cas $\lambda x.a$.* La dérivation de typage originale est :

$$\frac{\Gamma; x : \tau_1 \vdash a : \tau_2}{\Gamma \vdash \lambda x.a : \tau_1 \rightarrow \tau_2} \text{ABS}$$

Nous appliquons l'hypothèse d'induction à la prémisse, puis construisons :

$$\frac{\langle \Gamma \rangle; x : \langle \tau_1 \rangle \vdash \llbracket a \rrbracket : \llbracket \tau_2 \rrbracket}{\langle \Gamma \rangle \vdash \langle \lambda x.a \rangle : \langle \tau_1 \rightarrow \tau_2 \rangle} \text{ABS}$$

Nous avons exploité les équations $\langle \lambda x.a \rangle = \lambda x.\llbracket a \rrbracket$ et $\langle \tau_1 \rightarrow \tau_2 \rangle = \langle \tau_1 \rangle \rightarrow \llbracket \tau_2 \rrbracket$.

◦ *Cas v.* Notre hypothèse est :

$$\Gamma \vdash v : \tau$$

L'hypothèse d'induction donne :

$$\langle \Gamma \rangle \vdash \langle v \rangle : \langle \tau \rangle$$

Soit k une variable fraîche pour v , donc pour $\langle v \rangle$. Par affaiblissement, nous avons :

$$\langle \Gamma \rangle; k : \langle \tau \rangle \rightarrow o \vdash \langle v \rangle : \langle \tau \rangle$$

Nous pouvons alors construire :

$$\frac{\text{VAR} \frac{\langle \Gamma \rangle; k : \langle \tau \rangle \rightarrow o \vdash k : \langle \tau \rangle \rightarrow o \quad \langle \Gamma \rangle; k : \langle \tau \rangle \rightarrow o \vdash \langle v \rangle : \langle \tau \rangle}{\langle \Gamma \rangle; k : \langle \tau \rangle \rightarrow o \vdash k \langle v \rangle : o} \text{APP}}{\langle \Gamma \rangle \vdash \lambda k.k \langle v \rangle : (\langle \tau \rangle \rightarrow o) \rightarrow o} \text{ABS}$$

D'après les équations $\llbracket v \rrbracket = \lambda k.k \langle v \rangle$ et $\llbracket \tau \rrbracket = (\langle \tau \rangle \rightarrow o) \rightarrow o$, nous avons obtenu le but, à savoir : $\langle \Gamma \rangle \vdash \llbracket v \rrbracket : \llbracket \tau \rrbracket$.

◦ *Cas $a_1 a_2$.* La dérivation de typage originale est :

$$\frac{\Gamma \vdash a_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash a_2 : \tau_1}{\Gamma \vdash a_1 a_2 : \tau_2} \text{APP}$$

L'application des hypothèses d'induction donne $(\Gamma) \vdash \llbracket a_1 \rrbracket : \llbracket \tau_1 \rightarrow \tau_2 \rrbracket$ et $(\Gamma) \vdash \llbracket a_2 \rrbracket : \llbracket \tau_1 \rrbracket$. Soient k, x_1, x_2 des variables satisfaisant $(k \# a_1, a_2), (x_1 \# k, a_2), (x_2 \# k, x_1)$. Nous pouvons alors construire :

$$\begin{array}{c}
\begin{array}{c}
\dots \vdash x_1 : (\tau_1 \rightarrow \tau_2) \\
\dots \vdash x_2 : (\tau_1) \\
\dots \vdash k : (\tau_2) \rightarrow o
\end{array} \\
\hline
(\Gamma); k; x_1; x_2 : (\tau_1) \vdash x_1 x_2 k : o \quad \text{APP}^2 \\
\hline
(\Gamma) \vdash \llbracket a_2 \rrbracket : \llbracket \tau_1 \rrbracket \quad (\Gamma); k; x_1 \vdash \lambda x_2. x_1 x_2 k : (\tau_1) \rightarrow o \quad \text{ABS} \\
\hline
(\Gamma); k; x_1 : (\tau_1 \rightarrow \tau_2) \vdash \llbracket a_2 \rrbracket (\lambda x_2. x_1 x_2 k) : o \quad \text{APP} \\
\hline
(\Gamma) \vdash \llbracket a_1 \rrbracket : \llbracket \tau_1 \rightarrow \tau_2 \rrbracket \quad (\Gamma); k \vdash \lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 x_2 k) : (\tau_1 \rightarrow \tau_2) \rightarrow o \quad \text{ABS} \\
\hline
(\Gamma); k : (\tau_2) \rightarrow o \vdash \llbracket a_1 \rrbracket (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 x_2 k)) : o \quad \text{APP} \\
\hline
(\Gamma) \vdash \llbracket a_1 a_2 \rrbracket : \llbracket \tau_2 \rrbracket \quad \text{ABS}
\end{array}$$

Nous avons exploité les équations :

$$\begin{aligned}
\llbracket a_1 a_2 \rrbracket &= \lambda k. \llbracket a_1 \rrbracket (\lambda x_1. \llbracket a_2 \rrbracket (\lambda x_2. x_1 x_2 k)) \\
\llbracket \tau_1 \rrbracket &= ((\tau_1) \rightarrow o) \rightarrow o \\
\llbracket \tau_2 \rrbracket &= ((\tau_2) \rightarrow o) \rightarrow o \\
\llbracket \tau_1 \rightarrow \tau_2 \rrbracket &= ((\tau_1 \rightarrow \tau_2) \rightarrow o) \rightarrow o \\
(\tau_1 \rightarrow \tau_2) &= (\tau_1) \rightarrow \llbracket \tau_2 \rrbracket
\end{aligned}$$

et utilisé à deux reprises de façon implicite le lemme d'affaiblissement. \square

Question 4 *Quel rôle a joué la définition de la traduction des variables de types dans la démonstration précédente ? Pourquoi ?* \diamond

Solution. Aucun : nous n'avons absolument pas utilisé l'équation $\llbracket \alpha \rrbracket = \alpha$. C'est normal : la validité d'une dérivation de typage est préservée lorsque les variables de types qu'elle contient sont instanciées. Choisir une définition de $\llbracket \alpha \rrbracket$ autre que l'identité reviendrait à instancier la dérivation de typage obtenue après traduction, ce qui serait sûr, quoique sans intérêt. \square

On se place maintenant dans Système F explicitement typé, avec restriction du polymorphisme aux valeurs. La grammaire des valeurs, termes et types est étendue comme suit :

$$\begin{aligned}
v &::= \dots \mid \Lambda \alpha. v \\
a &::= \dots \mid a \tau \\
\tau &::= \dots \mid \forall \alpha. \tau
\end{aligned}$$

Question 5 *Étendre les traductions des valeurs, des termes, et des types. Étendre la démonstration du fait que la traduction préserve le typage. Quelle propriété de la fonction de traduction des types exploite-t-on ? Quel rôle joue maintenant la définition de la traduction des variables de types ?* \diamond

Solution. L'extension est la suivante :

$$\begin{aligned}
\llbracket \forall \alpha. \tau \rrbracket &= \forall \alpha. \llbracket \tau \rrbracket \\
\llbracket \Lambda \alpha. v \rrbracket &= \Lambda \alpha. \llbracket v \rrbracket \\
\llbracket a \tau \rrbracket &= \lambda k. \llbracket a \rrbracket (\lambda x. k (x (\tau)))
\end{aligned}$$

On ajoute à la démonstration les deux cas suivants :

◦ *Cas* $\Lambda \alpha. v$. La dérivation de typage originale est :

$$\frac{\Gamma \vdash v : \tau \quad \alpha \# \Gamma}{\Gamma \vdash \Lambda \alpha. v : \forall \alpha. \tau} \text{TABS}$$

Nous construisons alors la dérivation :

$$\frac{(\Gamma) \vdash \llbracket v \rrbracket : \llbracket \tau \rrbracket \quad \alpha \# (\Gamma)}{(\Gamma) \vdash \llbracket \Lambda \alpha. v \rrbracket : \llbracket \forall \alpha. \tau \rrbracket} \text{TABS}$$

Pour établir la première prémisse, nous avons utilisé l'hypothèse d'induction ; pour la seconde, le fait que les variables de types libres dans (Γ) sont celles libres dans Γ . Enfin, nous avons réécrit la conclusion en exploitant les équations $(\forall\alpha.\tau) = \forall\alpha.(\tau)$ et $(\Lambda\alpha.v) = \Lambda\alpha.(v)$.

◦ *Cas a τ*. La dérivation de typage originale est :

$$\frac{\Gamma \vdash a : \forall\alpha.\tau_2}{\Gamma \vdash a \tau_1 : [\alpha \mapsto \tau_1]\tau_2} \text{TAPP}$$

En appliquant l'hypothèse d'induction à la prémisse, nous obtenons $(\Gamma) \vdash \llbracket a \rrbracket : \llbracket \forall\alpha.\tau_2 \rrbracket$, c'est-à-dire, par définition de la traduction des types, $(\Gamma) \vdash \llbracket a \rrbracket : ((\forall\alpha.(\tau_2)) \rightarrow o) \rightarrow o$. Par ailleurs, nous notons que la fonction de traduction des types satisfait les propriétés suivantes :

$$\begin{aligned} [\alpha \mapsto (\tau_1)](\tau_2) &= ([\alpha \mapsto \tau_1]\tau_2) \\ [\alpha \mapsto (\tau_1)]\llbracket \tau_2 \rrbracket &= \llbracket [\alpha \mapsto \tau_1]\tau_2 \rrbracket \end{aligned} \quad \square$$

La démonstration de ces propriétés se fait aisément par induction structurelle, et exploite l'équation $(\alpha) = \alpha$. C'est donc ici que la traduction des variables de types joue un rôle.

Nous pouvons à présent construire la dérivation suivante :

$$\frac{\frac{\frac{\dots \vdash k : (([\alpha \mapsto \tau_1]\tau_2) \rightarrow o) \quad \frac{\dots \vdash x : \forall\alpha.(\tau_2)}{\dots \vdash x (\tau_1) : ([\alpha \mapsto \tau_1]\tau_2)} \text{TAPP}}{\dots \vdash k (\tau_1) : ([\alpha \mapsto \tau_1]\tau_2) \rightarrow o} \text{APP}}{(\Gamma); k; x : \forall\alpha.(\tau_2) \vdash k (x (\tau_1)) : o} \text{ABS}}{(\Gamma) \vdash \llbracket a \rrbracket : ((\forall\alpha.(\tau_2)) \rightarrow o) \rightarrow o} \text{APP}}{(\Gamma); k \vdash \lambda x.k (x (\tau_1)) : (\forall\alpha.(\tau_2)) \rightarrow o} \text{ABS}}{(\Gamma); k : ([\alpha \mapsto \tau_1]\tau_2) \rightarrow o \vdash \llbracket a \rrbracket (\lambda x.k (x (\tau_1))) : o} \text{APP}}{(\Gamma) \vdash \lambda k.\llbracket a \rrbracket (\lambda x.k (x (\tau_1))) : \llbracket [\alpha \mapsto \tau_1]\tau_2 \rrbracket} \text{ABS}}$$

Question 6 *Peut-on tirer parti de l'expressivité de Système F pour éviter de fixer un type o des réponses ? Comment modifier alors la définition de la traduction des types ? (On ne demande pas d'expliquer comment étendre la traduction des termes ni la démonstration de la propriété de préservation du typage.)* ◊

Solution. Le choix du type des réponses étant arbitraire, on peut le quantifier universellement :

$$\begin{aligned} (\alpha) &= \alpha \\ (\tau_1 \rightarrow \tau_2) &= (\tau_1) \rightarrow \llbracket \tau_2 \rrbracket \\ (\forall\alpha.\tau) &= \forall\alpha.(\tau) \\ \llbracket \tau \rrbracket &= \forall o.((\tau) \rightarrow o) \rightarrow o \quad (o \# \tau) \end{aligned}$$

(Rappelons que o est une variable de types.) On pourrait revoir la démonstration de la propriété du typage, en insérant des abstractions et applications de types appropriées. Cela n'était pas demandé. ◻

Partie III. Exceptions et transformation CPS «à deux canons»

Dans cette partie, on étend le langage d'entrée de la transformation CPS avec les constructions $\text{raise}(a)$ pour lever une exception et $\text{try } a \text{ with } x \rightarrow b$ pour récupérer les exceptions levées pendant l'évaluation de a .

$$\begin{aligned} \text{Termes :} & & a, b & ::= & v \mid a \ b \mid \text{raise}(a) \mid \text{try } a \text{ with } x \rightarrow b \\ \text{Valeurs :} & & v & ::= & x \mid \lambda x.a \\ \text{Résultats d'évaluations} & & r & ::= & v \mid \text{raise}(v) \end{aligned}$$

La sémantique naturelle de ce langage est donnée par la relation $a \Rightarrow r$, où r est ou bien une valeur v («le terme a s'évalue en la valeur v sans lever d'exception non rattrapée») ou bien une levée d'exception $\text{raise}(v)$ («l'évaluation du terme a se termine abruptement en levant l'exception v »).

$$\begin{array}{c}
v \Rightarrow v \quad \frac{a \Rightarrow \lambda x.a' \quad b \Rightarrow v \quad a'[x \leftarrow v] \Rightarrow r}{a b \Rightarrow r} \quad \frac{a \Rightarrow \text{raise}(v)}{a b \Rightarrow \text{raise}(v)} \quad \frac{a \Rightarrow v_1 \quad b \Rightarrow \text{raise}(v_2)}{a b \Rightarrow \text{raise}(v_2)} \\
\frac{a \Rightarrow v}{\text{raise}(a) \Rightarrow \text{raise}(v)} \quad \frac{a \Rightarrow \text{raise}(v)}{\text{raise}(a) \Rightarrow \text{raise}(v)} \quad \frac{a \Rightarrow v}{(\text{try } a \text{ with } x \rightarrow b) \Rightarrow v} \quad \frac{a \Rightarrow \text{raise}(v) \quad b[x \leftarrow v] \Rightarrow r}{(\text{try } a \text{ with } x \rightarrow b) \Rightarrow r}
\end{array}$$

On définit une traduction CPS de ce langage étendu avec des exceptions vers le λ -calcul sans exceptions, en suivant l'approche connue sous le nom de «CPS à deux canons». La traduction CPS $\llbracket a \rrbracket$ d'un terme a est une fonction $\lambda k_1 k_2 \dots$ qui prend en arguments non pas une continuation k comme dans la partie I, mais deux continuations k_1 et k_2 .

- La première continuation k_1 est la continuation «normale». Si l'évaluation de a termine sur la valeur v sans lever d'exception, la traduction $\llbracket a \rrbracket k_1 k_2$ applique la continuation k_1 à la valeur $\langle v \rangle$ correspondant à v .
- La seconde continuation k_2 est la continuation d'exception. Si l'évaluation de a termine en levant une exception de valeur v , la traduction $\llbracket a \rrbracket k_1 k_2$ applique la continuation k_2 à la valeur $\langle v \rangle$.

Voici la définition de la traduction CPS «à deux canons» pour les valeurs et les applications :

$$\begin{aligned}
\langle x \rangle &= x \\
\langle \lambda x.a \rangle &= \lambda x.\llbracket a \rrbracket \\
\langle v \rangle &= \lambda k_1 k_2.k_1 \langle v \rangle \\
\langle a_1 a_2 \rangle &= \lambda k_1 k_2.\llbracket a_1 \rrbracket (\lambda x_1.\llbracket a_2 \rrbracket (\lambda x_2.x_1 x_2 k_1 k_2) k_2) k_2
\end{aligned}$$

Question 7 Définir la traduction CPS «à deux canons» pour les constructions `raise` et `try...with` :

$$\begin{aligned}
\llbracket \text{raise}(a) \rrbracket &= \lambda k_1 k_2. ??? \\
\llbracket \text{try } a \text{ with } x \rightarrow b \rrbracket &= \lambda k_1 k_2. ???
\end{aligned}$$

On pourra raisonner par cas selon que a termine normalement ou bien lève une exception. ◇

Solution. Pour la levée d'exception $\text{raise}(a)$, c'est dans tous les cas la continuation d'exception k_2 qui doit être appelée. Si a termine normalement sur une valeur v , cette valeur est la valeur de l'exception qu'il faut lever en la passant à k_2 . Si a termine abruptement sur une valeur d'exception v , il faut propager cette exception en la passant également à k_2 . D'où :

$$\llbracket \text{raise}(a) \rrbracket = \lambda k_1 k_2. \llbracket a \rrbracket k_2 k_2$$

Pour la capture d'exception $\llbracket \text{try } a \text{ with } x \rightarrow b \rrbracket$, si a termine normalement sur une valeur v , il faut transmettre cette valeur à la continuation normale k_1 . En revanche, si a termine abruptement sur une valeur d'exception v , il faut lier x à cette valeur, puis évaluer b avec les mêmes continuations normale k_1 et d'exception k_2 .

$$\llbracket \text{try } a \text{ with } x \rightarrow b \rrbracket = \lambda k_1 k_2. \llbracket a \rrbracket k_1 (\lambda x. \llbracket b \rrbracket k_1 k_2)$$

Question 8 Énoncer (sans démonstration pour l'instant) une propriété de préservation sémantique analogue à celle de la question 1. Plus précisément : si $a \Rightarrow r$ et si K_1 et K_2 sont des valeurs, en quel terme se réduit $\llbracket a \rrbracket K_1 K_2$? ◇

Solution. Intuitivement, si $r = v$ (l'évaluation de a ne lève pas d'exception non rattrapée), la continuation normale K_1 va finir par être appelée sur la valeur $\langle v \rangle$ correspondante. On aurait donc $\llbracket a \rrbracket K_1 K_2 \xrightarrow{*} K_1 \langle v \rangle$ dans ce cas.

De même, si $r = \text{raise}(v')$ (l'évaluation de a lève l'exception v), la continuation d'exception K_2 va être appelée sur la valeur $\langle v' \rangle$. On aurait donc $\llbracket a \rrbracket K_1 K_2 \xrightarrow{*} K_2 \langle v' \rangle$ dans ce cas.

Au final, l'énoncé de la préservation sémantique pour la transformation CPS à deux canons est donc :

$$\text{Si } a \Rightarrow r \text{ et si } K_1 \text{ et } K_2 \text{ sont des valeurs, alors } \llbracket a \rrbracket K_1 K_2 \xrightarrow{*} \begin{cases} K_1 \langle v \rangle & \text{si } r = v; \\ K_2 \langle v' \rangle & \text{si } r = \text{raise}(v') \end{cases} \quad \square$$

On admettra à nouveau sans démonstration le résultat de commutation ci-dessous entre substitution et transformation CPS :

$$\llbracket a[x \leftarrow v] \rrbracket = \llbracket a \rrbracket [x \leftarrow \langle v \rangle]$$

Question 9 Démontrer l'énoncé de préservation sémantique de la question précédente dans les cas suivants : (1) a est une valeur ; (2) $a = \text{try } a_1 \text{ with } x \rightarrow a_2$. \diamond

Solution. Comme pour la question 1, la démonstration est par récurrence sur la dérivation d'évaluation $a \Rightarrow r$.

◦ Cas a est une valeur v . Dans ce cas, $a = v$ et le résultat r de l'évaluation est également la valeur v . Donc,

$$\llbracket v \rrbracket K_1 K_2 = (\lambda k_1 k_2. k_1 \langle v \rangle) K_1 K_2 \rightarrow K_1 \langle v \rangle$$

C'est le résultat attendu.

◦ Cas $a = \text{try } a_1 \text{ with } x \rightarrow a_2$ et a_1 ne lève pas d'exception. Dans ce cas, l'évaluation de a est de la forme

$$\frac{a_1 \Rightarrow v}{(\text{try } a_1 \text{ with } x \rightarrow a_2) \Rightarrow v}$$

et $r = v$. On a la séquence de réductions suivante :

$$\begin{aligned} \llbracket a \rrbracket K_1 K_2 &= (\lambda k_1 k_2. \llbracket a_1 \rrbracket k_1 (\lambda x. \llbracket a_2 \rrbracket k_1 k_2)) K_1 K_2 \\ &\xrightarrow{*} \llbracket a_1 \rrbracket K_1 (\lambda x. \llbracket a_2 \rrbracket K_1 K_2) \\ &\xrightarrow{*} K_1 \langle v \rangle \text{ par hypothèse de récurrence appliquée à } a_1 \Rightarrow v \end{aligned}$$

C'est le résultat attendu.

◦ Cas $a = \text{try } a_1 \text{ with } x \rightarrow a_2$ et a_1 lève une exception. Dans ce cas, l'évaluation de a est de la forme

$$\frac{a_1 \Rightarrow \text{raise}(v) \quad a_2[x \leftarrow v] \Rightarrow r}{(\text{try } a_1 \text{ with } x \rightarrow a_2) \Rightarrow r}$$

On a la séquence de réductions suivante :

$$\begin{aligned} \llbracket a \rrbracket K_1 K_2 &= (\lambda k_1 k_2. \llbracket a_1 \rrbracket k_1 (\lambda x. \llbracket a_2 \rrbracket k_1 k_2)) K_1 K_2 \\ &\xrightarrow{*} \llbracket a_1 \rrbracket K_1 (\lambda x. \llbracket a_2 \rrbracket K_1 K_2) \\ &\xrightarrow{*} (\lambda x. \llbracket a_2 \rrbracket K_1 K_2) \langle v \rangle \text{ par hypothèse de récurrence appliquée à } a_1 \Rightarrow \text{raise}(v) \\ &\xrightarrow{*} \llbracket a_2 \rrbracket [x \leftarrow \langle v \rangle] K_1 K_2 \\ &= \llbracket a_2[x \leftarrow v] \rrbracket K_1 K_2 \text{ par le lemme de commutation} \\ &\xrightarrow{*} \begin{cases} K_1 \langle v \rangle & \text{si } r = v; \\ K_2 \langle v' \rangle & \text{si } r = \text{raise}(v') \end{cases} \text{ par hypothèse de récurrence appliquée à } a_2[x \leftarrow v] \Rightarrow r \end{aligned}$$

C'est le résultat attendu. \square

Partie IV. Transformation CPS «à deux canons» et préservation du typage

On définit un système de types simples pour le λ -calcul enrichi avec des exceptions de la partie III. Les types sont les suivants :

$$\begin{aligned} \tau &::= \alpha \mid \tau \rightarrow \phi \\ \phi &::= \tau + \tau \end{aligned}$$

Les jugements de typage sont de la forme suivante :

$$\begin{aligned} \Gamma \vdash v : \tau &\quad (\text{valeurs}) \\ \Gamma \vdash a : \phi &\quad (\text{termes}) \end{aligned}$$

Un type de résultat ϕ est de la forme $\tau_1 + \tau_2$, où τ_1 est le type de la valeur produite, si le terme considéré termine normalement, et τ_2 est le type de l'exception produite, si le terme considéré lance une exception. L'environnement de typage Γ associe aux variables des types ordinaires τ .

Question 10 Donner les six règles de typage correspondant aux deux formes de valeurs et aux quatre formes d'expressions. (On ne demande pas de démonstration.) \diamond

Solution. Les règles sont les suivantes :

$$\begin{array}{c}
\text{VAR} \\
\frac{}{\Gamma \vdash x : \Gamma(x)}
\end{array}
\qquad
\begin{array}{c}
\text{ABS} \\
\frac{\Gamma; x : \tau \vdash a : \phi}{\Gamma \vdash \lambda x.a : \tau \rightarrow \phi}
\end{array}
\qquad
\begin{array}{c}
\text{VAL} \\
\frac{\Gamma \vdash v : \tau_1}{\Gamma \vdash v : \tau_1 + \tau_2}
\end{array}
\qquad
\begin{array}{c}
\text{APP} \\
\frac{\Gamma \vdash a_1 : (\tau'_1 \rightarrow \tau_1 + \tau_2) + \tau_2 \quad \Gamma \vdash a_2 : \tau'_1 + \tau_2}{\Gamma \vdash a_1 a_2 : \tau_1 + \tau_2}
\end{array}$$

$$\begin{array}{c}
\text{RAISE} \\
\frac{\Gamma \vdash a : \tau_2 + \tau_2}{\Gamma \vdash \text{raise}(a) : \tau_1 + \tau_2}
\end{array}
\qquad
\begin{array}{c}
\text{TRY} \\
\frac{\Gamma \vdash a_1 : \tau_1 + \tau_2 \quad \Gamma; x : \tau_2 \vdash a_2 : \tau_1 + \tau'_2}{\Gamma \vdash \text{try } a_1 \text{ with } x \rightarrow a_2 : \tau_1 + \tau'_2}
\end{array}$$

La règle VAL reflète le fait qu'une valeur ne lance pas d'exception : le type τ_2 y est arbitraire. De façon duale, dans RAISE, le type τ_1 est arbitraire.

La règle APP reflète le fait qu'une exception peut être lancée par l'exécution du terme a_1 , par l'exécution du terme a_2 , ou bien lors de l'exécution de l'application de fonction ; dans les trois cas, on doit obtenir une exception de même type τ_2 .

Dans TRY, l'exception produite par a_1 , de type τ_2 , est liée à la variable x . Les deux branches a_1 et a_2 doivent produire une valeur de même type τ_1 . La branche a_2 peut produire une exception, de type τ'_2 , qui n'est pas rattrapée. \square

Question 11 Le langage source de la traduction CPS à deux canons est le λ -calcul simplement typé décrit ci-dessus, tandis que le langage cible reste le λ -calcul simplement typé standard. Définir les fonctions de traduction des types. (On ne demande pas de démonstration.) \diamond

Solution. La définition est la suivante :

$$\begin{aligned}
\llbracket \alpha \rrbracket &= \alpha \\
\llbracket \tau \rightarrow \phi \rrbracket &= \llbracket \tau \rrbracket \rightarrow \llbracket \phi \rrbracket \\
\llbracket \tau_1 + \tau_2 \rrbracket &= (\llbracket \tau_1 \rrbracket \rightarrow o) \rightarrow (\llbracket \tau_2 \rrbracket \rightarrow o) \rightarrow o
\end{aligned}
\qquad \square$$

Références

- [1] Josh Berdine, Peter W. O'Hearn, Uday S. Reddy, and Hayo Thielecke. [Linear continuation-passing](#). *Higher-Order and Symbolic Computation*, 15(2–3) :181–208, 2002.
- [2] Andrew Kennedy. [Compiling with continuations, continued](#). In *ACM International Conference on Functional Programming (ICFP)*, pages 177–190, September 2007.
- [3] Hayo Thielecke. [Comparing control constructs by double-barrelled CPS](#). *Higher-Order and Symbolic Computation*, 15(2–3) :141–160, 2002.
- [4] Hayo Thielecke. [From control effects to typed continuation passing](#). In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 139–149, January 2003.