Examen du cours “Typage et programmation”

D.E.A. “Programmation”

14 décembre 2000

Le but de ce problème est de concevoir une analyse statique de programmes ML avec exceptions qui nous indique si l'évaluation d'une expression peut ou non déclencher une exception non rattrapée.

Les parties 1, 2 et 3 sont essentiellement indépendantes (la partie 3 utilise les définitions des parties 1 et 2, mais pas les résultats des questions de ces parties). Les questions 8 et 9 sont techniques et peuvent être admises pour aborder la partie 4.

1 ML avec exceptions

Le langage que nous allons analyser, noté ML_e, est ML enrichi avec des exceptions. Il est semblable au langage de la section 5.7 des notes du cours, avec une présentation formelle légèrement différente. ML_e est formellement défini comme suit :

Expressions :	a	$::= x \mid c \mid op \mid \texttt{fun}\ x \to a \mid a_1\ a_2 \mid (a_1, a_2) \mid \texttt{let}\ x = a_1\ \texttt{in}\ a_2$ $\mid \texttt{raise}\ a \mid \texttt{try}\ a_1\ \texttt{with}\ x \to a_2$
Constantes :	c	$::= 0 \mid 1 \mid -1 \mid \ldots \mid \texttt{divzero}$
Opérateurs :	op	$::= + \mid / \mid \texttt{fst} \mid \texttt{snd}$
Valeurs :	v	$::= c \mid op \mid \texttt{fun}\ x \to a \mid (v_1, v_2)$
Contextes de réduction :	Γ	$::= [\,] \mid \Gamma\ a \mid v\ \Gamma \mid (\Gamma, a) \mid (v, \Gamma) \mid \texttt{let}\ x = \Gamma\ \texttt{in}\ a$ $\mid \texttt{raise}\ \Gamma \mid \texttt{try}\ \Gamma\ \texttt{with}\ x \to a$
Types :	τ	$::= \texttt{int} \mid \texttt{exn} \mid \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2$

Règles de réduction :

$$
\begin{array}{rcll}
(\texttt{fun}\ x \to a)\ v & \stackrel{\varepsilon}{\to} & a\{x \leftarrow v\} & (\beta_{fun}) \\
(\texttt{let}\ x = v\ \texttt{in}\ a) & \stackrel{\varepsilon}{\to} & a\{x \leftarrow v\} & (\beta_{let}) \\
+\ (n_1, n_2) & \stackrel{\varepsilon}{\to} & n \text{ si } n_1, n_2 \text{ entiers et } n = n_1 + n_2 & (\delta_+) \\
/\ (n_1, n_2) & \stackrel{\varepsilon}{\to} & n \text{ si } n_1, n_2 \text{ entiers, } n_2 \neq 0 \text{ et } n = n_1 / n_2 & (\delta_/) \\
/\ (n_1, 0) & \stackrel{\varepsilon}{\to} & \texttt{raise divzero} \text{ si } n_1 \text{ entier} & (\delta_{/0}) \\
\texttt{fst}\ (v_1, v_2) & \stackrel{\varepsilon}{\to} & v_1 & (\delta_{fst}) \\
\texttt{snd}\ (v_1, v_2) & \stackrel{\varepsilon}{\to} & v_2 & (\delta_{snd}) \\
(\texttt{try}\ v\ \texttt{with}\ x \to a) & \stackrel{\varepsilon}{\to} & v & (\text{try}_1) \\
(\texttt{try raise}\ v\ \texttt{with}\ x \to a) & \stackrel{\varepsilon}{\to} & a\{x \leftarrow v\} & (\text{try}_2) \\
(\texttt{raise}\ \ v)\ a & \stackrel{\varepsilon}{\to} & \texttt{raise}\ v & (\text{propagation}_1)
\end{array}
$$

$$
\begin{array}{rcll}
v\ (\texttt{raise}\ \ v') & \xrightarrow{\varepsilon} & \texttt{raise}\ v' & (\text{propagation}_2)\\
\texttt{let}\ x = \texttt{raise}\ \ v\ \texttt{in}\ a & \xrightarrow{\varepsilon} & \texttt{raise}\ v & (\text{propagation}_3)\\
(\texttt{raise}\ \ v, a) & \xrightarrow{\varepsilon} & \texttt{raise}\ v & (\text{propagation}_4)\\
(v, \texttt{raise}\ \ v') & \xrightarrow{\varepsilon} & \texttt{raise}\ v' & (\text{propagation}_5)
\end{array}
$$

$$
\frac{a \xrightarrow{\varepsilon} a'}{\Gamma(a) \xrightarrow{\varepsilon} \Gamma(a')}\ (\text{contexte})
$$

Règle de typage des constructions `raise` et `try` :

$$
\frac{E \vdash a : \texttt{exn}}{E \vdash \texttt{raise}\ a : \tau}\ (\text{raise}) \qquad\qquad \frac{E \vdash a_1 : \tau \qquad E + \{x : \texttt{exn}\} \vdash a_2 : \tau}{E \vdash (\texttt{try}\ a_1\ \texttt{with}\ x \to a_2) : \tau}\ (\text{try})
$$

Question 1 Quel est le résultat de l'évaluation de l'expression a ci-dessous ?

$$
a = \texttt{try}\ (1, /(100, n))\ \texttt{with}\ x \to (0, 0)
$$

Ici, n est une constante entière. On donnera chaque étape de réduction et on traitera les deux cas $n = 0$ et $n \neq 0$.

2 Types concrets

On considère le langage ML_r obtenu en ajoutant à mini-ML de base (sans exceptions) le type (τ_1, τ_2) `res` correspondant à la déclaration de type concret Caml suivante :

```
type (α, β) res = V of α | E of β
```

C'est-à-dire, ML_r comprend deux constructeurs V et E ainsi qu'une construction de filtrage `match...with` sur les valeurs de type (τ_1, τ_2) `res`.

Expressions :	$a\ ::= x \mid c \mid op \mid \texttt{fun x}\ \to a \mid a_1\ a_2 \mid (a_1, a_2) \mid \texttt{let}\ x = a_1\ \texttt{in}\ a_2$ $\mid \texttt{match}\ a_1\ \texttt{with V}\ x \to a_2 \mid \texttt{E}\ y \to a_3$
Constantes :	$c\ ::= 0 \mid 1 \mid -1 \mid \ldots \mid \texttt{divzero}$
Opérateurs :	$op\ ::= + \mid / \mid \texttt{fst} \mid \texttt{snd} \mid \texttt{V} \mid \texttt{E}$
Valeurs :	$v\ ::= c \mid op \mid \texttt{fun}\ x \to a \mid (v_1, v_2) \mid \texttt{V}\ v \mid \texttt{E}\ v$
Contextes de réduction :	$\Gamma\ ::= [\,] \mid \Gamma\ a \mid v\ \Gamma \mid \texttt{let}\ x = \Gamma\ \texttt{in}\ a \mid (\Gamma, a) \mid (v, \Gamma)$ $\mid \texttt{match}\ \Gamma\ \texttt{with V}\ x \to a_2 \mid \texttt{E}\ y \to a_3$
Types :	$\tau\ ::= \texttt{int} \mid \texttt{exn} \mid \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid (\tau_1, \tau_2)\ \texttt{res}$

Règles de réduction : les règles (β_{fun}), (β_{let}), (δ_+), $(\delta_/)$, (δ_{fst}), (δ_{snd}), (contexte), plus les deux règles suivantes pour le `match` :

$$
\begin{array}{rcll}
(\texttt{match V}\ v\ \texttt{with V}\ x \to a_2 \mid \texttt{E}\ y \to a_3) & \xrightarrow{\varepsilon} & a_2\{x \leftarrow v\} & (\text{match}_1)\\
(\texttt{match E}\ v\ \texttt{with V}\ x \to a_2 \mid \texttt{E}\ y \to a_3) & \xrightarrow{\varepsilon} & a_3\{y \leftarrow v\} & (\text{match}_2)
\end{array}
$$

et la règle suivante pour la division par zéro :

$$
/(n_1, 0) \xrightarrow{\varepsilon} /(n_1, 0)\ \text{si } n_1 \text{ constante entière} \qquad (\delta_{boucle})
$$

Règles de typage :

$$\begin{array}{rcl} TC(\mathtt{V}) & ::= & \forall\alpha,\beta.\ \alpha \to (\alpha,\beta)\ \mathtt{res} \\ TC(\mathtt{E}) & ::= & \forall\alpha,\beta.\ \beta \to (\alpha,\beta)\ \mathtt{res} \end{array}$$

$$\frac{E \vdash a_1 : (\tau_1,\tau_2)\ \mathtt{res} \qquad E + \{x : \tau_1\} \vdash a_2 : \tau \qquad E + \{y : \tau_2\} \vdash a_3 : \tau}{E \vdash (\mathtt{match}\ a_1\ \mathtt{with}\ \mathtt{V}\ x \to a_2 \mid \mathtt{E}\ y \to a_3) : \tau}\ \text{(match)}$$

Question 2 Montrer que les règles de réduction (match_1) et (match_2) préservent le typage, c'est-à-dire que l'hypothèse (H1) du cours est vérifiée pour ces deux règles.

Question 3 Montrer l'hypothèse (H2') du cours pour la construction `match` : si l'expression $a = (\mathtt{match}\ v\ \mathtt{with}\ \mathtt{V}\ x \to a_2 \mid \mathtt{E}\ y \to a_3)$ est bien typée dans l'environnement de typage vide, alors a se réduit.

Question 4 Le typage du langage ML_r est-il sûr ? Même question si l'on enlève la règle de réduction (δ_{boucle}).

3 Codage des exceptions

Nous allons maintenant définir une traduction de ML_e dans ML_r dans laquelle les exceptions, au lieu d'être considérées comme une construction primitive avec ses règles spécifiques, sont encodées dans les résultats des expressions. L'idée est de traduire toute expression a de ML_e en une expression $[\![a]\!]$ de ML_r telle que

- si a s'évalue en une valeur v sans lever d'exceptions, alors $[\![a]\!]$ s'évalue en $\mathtt{V}\ v$
- si l'évaluation de a déclenche une exception e, alors $[\![a]\!]$ s'évalue en $\mathtt{E}\ e$.

La traduction est définie formellement comme suit :

$$\begin{array}{rcl}
[\![x]\!] & = & \mathtt{V}\ x \\
[\![c]\!] & = & \mathtt{V}\ c \\
[\![\mathtt{fun}\ x \to a]\!] & = & \mathtt{V}(\mathtt{fun}\ x \to [\![a]\!]) \\
[\![a_1\ a_2]\!] & = & \begin{array}[t]{l} \mathtt{match}\ [\![a_1]\!] \\ \mathtt{with}\ \mathtt{V}\ x_1 \to \begin{array}[t]{l} \mathtt{match}\ [\![a_2]\!] \\ \mathtt{with}\ \mathtt{V}\ x_2 \to x_1\ x_2 \\ \quad\mid\ \mathtt{E}\ y_2 \to \mathtt{E}\ y_2 \end{array} \\ \quad\mid\ \mathtt{E}\ y_1 \to \mathtt{E}\ y_1 \end{array} \\
[\![(a_1,a_2)]\!] & = & \begin{array}[t]{l} \mathtt{match}\ [\![a_1]\!] \\ \mathtt{with}\ \mathtt{V}\ x_1 \to \begin{array}[t]{l} \mathtt{match}\ [\![a_2]\!] \\ \mathtt{with}\ \mathtt{V}\ x_2 \to \mathtt{V}(x_1,x_2) \\ \quad\mid\ \mathtt{E}\ y_2 \to \mathtt{E}\ y_2 \end{array} \\ \quad\mid\ \mathtt{E}\ y_1 \to \mathtt{E}\ y_1 \end{array} \\
[\![\mathtt{let}\ x = a_1\ \mathtt{in}\ a_2]\!] & = & \mathtt{match}\ [\![a_1]\!]\ \mathtt{with}\ \mathtt{V}\ x \to [\![a_2]\!] \mid \mathtt{E}\ y \to \mathtt{E}\ y
\end{array}$$

$$
\begin{array}{rcl}
[\![\texttt{raise}\ a]\!] & = & \texttt{match}\ [\![a]\!]\ \texttt{with V}\ x \to \texttt{E}\ x \mid \texttt{E}\ y \to \texttt{E}\ y \\
[\![\texttt{try}\ a_1\ \texttt{with}\ y \to a_2]\!] & = & \texttt{match}\ [\![a_1]\!]\ \texttt{with V}\ x \to \texttt{V}\ x \mid \texttt{E}\ y \to [\![a_2]\!] \\
[\![+]\!] & = & \texttt{V(fun}\ x \to \texttt{V}(+(\texttt{fst}\ x, \texttt{snd}\ x))) \\
[\![\texttt{fst}]\!] & = & \texttt{V(fun}\ x \to \texttt{V(fst}\ x)) \\
[\![\texttt{snd}]\!] & = & \texttt{V(fun}\ x \to \texttt{V(snd}\ x))
\end{array}
$$

Question 5 Donner les traductions des expressions suivantes :

$$
\begin{array}{rcl}
a & = & \texttt{fun}\ x \to 1 \\
b & = & \texttt{try raise}\ c\ \texttt{with}\ y \to 1
\end{array}
$$

(où c est une constante quelconque). En quoi se réduit la traduction $[\![b]\!]$?

Question 6 Justifier informellement les traductions de l'application $a_1\ a_2$ et de la récupération d'exceptions $\texttt{try}\ a_1\ \texttt{with}\ y \to a_2$. (On pourra discuter selon que a_1 ou a_2 lève ou non une exception, et on expliquera pourquoi dans chaque cas le comportement de la traduction "suit" bien le comportement de l'expression d'origine.)

Question 7 La traduction de l'opérateur `/` a été omise ci-dessus. Proposer (sans justification) une traduction adéquate, correspondant à lever l'exception `divzero` si le diviseur est nul. (On pourra utiliser des constructions de Caml non formalisées dans ML_r si nécessaire.)

Définition Si a est une expression de ML_r, on note $\mathcal{S}(a)$ la forme normale de a par-rapport à la réduction (match$_1$), appliquée dans n'importe quel contexte (et pas seulement un contexte d'évaluation). Autrement dit, $\mathcal{S}(a)$ est l'expression a dans laquelle toutes les sous-expressions

$$\texttt{match V}\ \ v\ \texttt{with V}\ x \to a_2 \mid \texttt{E}\ y \to a_3$$

sont remplacées par $a_2\{x \leftarrow v\}$. Nous utiliserons $\mathcal{S}$ pour simplifier les résultats de la traduction ci-dessus. Par exemple, si c_1 et c_2 sont des constantes, on a

$$
\begin{array}{rll}
[\![c_1\ c_2]\!] = & \texttt{match V}\ c_1 & \\
& \texttt{with V}\ x_1 \to & \texttt{match V}\ c_2 \\
& & \texttt{with V}\ x_2 \to x_1\ x_2 \\
& & \quad \mid\ \texttt{E}\ y_2 \to \texttt{E}\ y_2 \\
& \mid\ \texttt{E}\ y_1 \to \texttt{E}\ y_1 &
\end{array}
$$

mais $\mathcal{S}([\![c_1\ c_2]\!]) = c_1\ c_2$ tout simplement.

Question 8 Montrer que si v est une valeur de ML_e, alors sa traduction simplifiée $\mathcal{S}([\![v]\!])$ est une valeur de ML_r de la forme $\texttt{V}\ v'$ pour une certaine valeur v'.

Définition Dans la suite, on notera $\overline{v}$ cette valeur v'. Autrement dit, on a l'égalité $\mathcal{S}([\![v]\!]) = \texttt{V}\ \overline{v}$ pour toute valeur v. On admettra sans démonstration le lemme suivant de commutation entre traduction et substitution :

$$\mathcal{S}([\![a\{x \leftarrow v\}]\!]) = (\mathcal{S}([\![a]\!]))\{x \leftarrow \overline{v}\}$$

Question 9 Montrer que si $a \to a'$ dans ML_e, alors ou bien $\mathcal{S}(\llbracket a \rrbracket) = \mathcal{S}(\llbracket a' \rrbracket)$, ou bien $\mathcal{S}(\llbracket a \rrbracket) \to \mathcal{S}(\llbracket a' \rrbracket)$ dans ML_r. On montrera le résultat pour les réductions (β_{fun}), (try_1), (try_2) et (propagation_1) de ML_e, et on l'admettra pour les autres règles de réduction.

4 Typage dans ML_r et analyse d'exceptions

Question 10 Donner sans justification les types principaux dans ML_r des expressions suivantes :
- $\mathcal{S}(\llbracket 1 \rrbracket)$
- $\mathcal{S}(\llbracket \texttt{raise divzero} \rrbracket)$ (on supposera $TC(\texttt{divzero}) = \texttt{exn}$)
- $\mathcal{S}(\llbracket \texttt{try raise divzero with } y \to 1 \rrbracket)$

Résultat admis : si a est un terme clos bien typé de ML_e, alors $\mathcal{S}(\llbracket a \rrbracket)$ est bien typé dans ML_r et a un type principal de la forme (τ_1, τ_2) `res` où τ_1 est relié au type de a dans ML_e et τ_2 est soit `exn`, soit une variable de type α.

Question 11 Soit a un terme clos bien typé de ML_e. Montrer que si $\emptyset \vdash \mathcal{S}(\llbracket a \rrbracket) : (\tau, \alpha)$ `res` pour un type τ et une variable de type α, alors a ne peut pas se réduire en une exception non rattrapée `raise` v. (Indication : supposons que $a \xrightarrow{*} \texttt{raise } v$ dans ML_e. En quoi se réduit $\mathcal{S}(\llbracket a \rrbracket)$? Quel est le type du résultat de la réduction de $\mathcal{S}(\llbracket a \rrbracket)$? En déduire une contradiction.)

Question 12 En déduire un algorithme d'analyse d'exceptions non rattrapées. Cet algorithme prendra en entrée un terme clos a bien typé de ML_e et renverra `faux` si l'évaluation de a ne peut pas déboucher sur une exception non rattrapée (c'est-à-dire, $a \not\xrightarrow{*} \texttt{raise } v$), et `vrai` si l'évaluation de a peut ou non déboucher sur une exception non rattrapée. Prouver la correction de votre algorithme.