Typage

Examen du 19 décembre 2012 – durée : 2h

Docum nts autorisés

Programmation fonctionnelle

On considèr ici l langag MiniML suivant :

Expr ssions	e	$::= x \mid c \mid op \mid \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \mid \texttt{fun}\ x \to e \mid e_1\ e_2 \mid \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2$
Constant s	c	$::= \texttt{true} \mid \texttt{false} \mid \texttt{0} \mid \texttt{1} \mid \ldots$
Opérat urs	op	$::= \texttt{+} \mid - \mid * \mid \texttt{not} \mid \texttt{and} \mid \texttt{fix} \mid \texttt{iszero}$
Typ s	τ	$::= \texttt{int} \mid \texttt{bool} \mid \alpha \mid \tau_1 \to \tau_2$
Val urs	v	$::= c \mid op \mid \texttt{fun}\ x \to e$

On pourra utilis r l'abréviation $(\texttt{fun}\ x\ y \to \ldots)$ au li u d $(\texttt{fun}\ x \to \texttt{fun}\ y \to \ldots)$ pour l s fonctions à plusi urs argum nts. On pourra suppos r qu l'opérat ur `fix` st toujours appliqué à d s xpr ssions d la form $\texttt{fun}\ f\ x \to \ldots$, t l'abréviation $\texttt{let rec}\ f\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2$ désign ra $\texttt{let}\ f\ \texttt{=}\ \texttt{fix}(\texttt{fun}\ f\ x \to e_1)\ \texttt{in}\ e_2$.

Nous utilis rons ici la sémantiqu à p tit-pas av c substitutions vu n cours. Par x mpl , la règl d réduction conc rnant `fix` st la suivant :

$$\texttt{fix}(\texttt{fun}\ f \to e) \quad \to_\epsilon \quad e\{f \leftarrow \texttt{fix}(\texttt{fun}\ f \to e)\}$$

On rapp ll égal m nt l typag d l'opérat ur `fix` :

$$\vdash \texttt{fix} : \forall\alpha.(\alpha \to \alpha) \to \alpha$$

Exercice 1

Pour l s xpr ssions suivant s, donn r la suit d s réductions dans la sémantiqu à p tit-pas av c substitutions t dét rmin r si l'on aboutit à un val ur, un blocag ou à un suit infini d réductions.

1. $(\texttt{fun}\ x\ y \to x * y)\ \texttt{3}\ \texttt{4}$
2. $(\texttt{fun}\ x\ y \to x * y)\ \texttt{3}$
3. $(\texttt{fun}\ x\ y \to x * y)\ \texttt{3}\ \texttt{4}\ \texttt{5}$
4. $\texttt{if true then 1 else 2}$
5. $\texttt{if true then 1 else 2 3}$
6. $(\texttt{fun}\ x \to x\ x)\ (\texttt{fun}\ x \to x)$
7. $(\texttt{fun}\ x \to x\ x)\ (\texttt{fun}\ x \to x\ x)$
8. $\texttt{let}\ f\ \texttt{=}\ \texttt{fun}\ x \to x\ \texttt{in if}\ f\ \texttt{true then}\ f\ \texttt{1 else 2}$
9. $\texttt{let rec}\ f\ x\ \texttt{=}\ x\ \texttt{+}\ f(x - \texttt{1})\ \texttt{in}\ f\ \texttt{3}$
10. $\texttt{let rec}\ f\ x\ \texttt{=}\ \texttt{if iszero}(x)\ \texttt{then 0 else}\ x\ \texttt{+}\ f(x - \texttt{1})\ \texttt{in}\ f\ \texttt{3}$

Exercice 2

On utilise le typage polymorphe du MiniML vu en cours. Pour les expressions de la question précédente, construire une dérivation de typage dans l'environnement initial mentionné vide, ou justifier que l'expression est mal typée. Peut-on avoir des expressions mal typées mais se réduisant malgré tout vers une valeur ? Peut-on avoir des expressions bien typées ne se réduisant pas vers une valeur ?

Programmation Impérative

Comme vu en cours, on considère maintenant les opérateurs `ref`, `:=` et `!`, ainsi que la sémantique à petits pas avec stores, et les règles de typage MiniML adaptées à l'impératif. On rappelle que $e; e'$ est une abréviation pour `let _ =` e `in` e'.

Exercice 3

Pour les expressions suivantes, donner la suite des réductions dans la sémantique à petits-pas avec substitutions, à partir d'un store initial mentionné vide. Déterminer si l'on aboutit à une valeur, un blocage ou à une suite infinie de réductions.

1. `let` x `=` `ref 0 in` x `:= 1;` $!x$
2. `let` x `=` `ref true in` x `:= 1;` $!x$
3. `let` r `=` `ref (fun` $x \to x$`) in (!`r `1, !`r *true*`)`
4. `let` r `=` `ref (fun` $x \to x$`) in`
   `let` f x `=` `if iszero(`x`) then 0 else` x `+ !`r `(`$x - 1$`) in`
   r `:=` f`;` f `3`

Exercice 4

Pour les expressions de la question précédente, construire une dérivation de typage dans l'environnement initial mentionné vide, ou justifier que l'expression est mal typée.

Typage des exceptions

On rajoute maintenant une construction `try` e `with` $x \to e'$, un opérateur `raise`, et quelques constantes marquants des exceptions comme `Oups`, `DivByZero`. La sémantique de ces constructions est la suivante :

$$\texttt{try } v \texttt{ with } x \to e \quad \to_\epsilon \quad v$$

$$\texttt{try raise(}v\texttt{) with } x \to e \quad \to_\epsilon \quad e\{x \leftarrow v\}$$

Il faut aussi ajouter une règle exprimant la propagation des exceptions vers le haut des expressions :

$$\texttt{(raise(}v\texttt{))} \quad \to_\epsilon \quad \texttt{raise(}v\texttt{)}$$

lorsque est un contexte d'exception *non-vide*. Un contexte d'exception est un contexte de réduction ne contenant pas de `try ... with`.

Contextes de réduction :

$$::= \texttt{[]} \mid a \quad \mid \quad v \mid \texttt{let } x \texttt{ =} \quad \texttt{in } a \mid (\ , a) \mid (v, \ ) \mid \texttt{try} \quad \texttt{with } x \to a$$

Cont xt s d' xc ptions :

$$::= \texttt{[]} \mid a \quad \mid \quad v \mid \texttt{let}\ x\ \texttt{=}\quad \texttt{in}\ a \mid (\quad, a) \mid (v,\quad)$$

Exercice 5

Donn r la suit d s réductions pour l s xpr ssions suivant s :

1. `raise Oups`
2. `try 1 + raise Oups with` $x \to$ `0`

Exercice 6

Propos z d s règl s d typag pour `r ise` t `try ... with`. On pourra utilis r un typ particuli r `exn` pour marqu r l typ d s constant s d' xc ptions. Illustr r l comport m nt d vos règl s d typag sur l s xpr ssions précéd nt s t qu lqu s autr s x mpl s. Comm nt ét ndr c la pour qu l s xc ptions puiss nt avoir d s argum nts ?

Exercice 7

On souhait maint nant marqu r dans l typ s d s functions l s xc ptions qu' ll s p uv nt lanc r, dans l' sprit d java. Ainsi l typ d la division s rait :

$$(/) : \texttt{int} \to \texttt{int} \to \texttt{int throws DivByZero}$$

Propos r un t l systèm d typ pour un MiniPascal (c' st à dir MiniML sans polymorphism ni application parti ll ni fonctions n argum nts). Illustr z sur d s x mpl s. Qu s pass -t'il quand on ssai d'intégr r c s fonctionalités d MiniML n plus ?