Exceptions

Principles of Programming Languages

CSE 526
exception NotFound;

let rec get x l =
    match l with
    | [] -> raise NotFound
    | (y,z)::ys -> if (x=y) then z else (get x ys);

let rec bar x l =
    try
    (get x l) + 1
    with
    NotFound -> 0
    ;
Adding Exceptions to the $\lambda$-calculus: 1st Attempt

Syntax:

\[
t ::= \ldots \quad \text{terms} \\
| \quad \text{error}
\]

Additional Rules:

- Single-step evaluation rules:
  
  \[
  \text{error} \ t_2 \to \text{error} \quad \text{E-AppErr1}
  \]
  
  \[
  \nu_1 \text{error} \to \text{error} \quad \text{E-AppErr2}
  \]

- Typing rule:
  
  \[
  \Gamma \vdash \text{error} : T \quad \text{T-ERROR}
  \]
Adding Exceptions: 1st attempt (contd.)

- \((\lambda x. \lambda y. y) \text{error} \rightarrow \text{error}\)
- \((\text{fix } (\lambda x : A. x)) \text{error} \rightarrow^\omega \)
- The term \text{error} has non-unique types.
- Every normal form is a value or \text{error}. 
Exception Handling : 1st attempt

Syntax:

\[
\begin{align*}
t & : = \ldots \quad \text{terms} \\
& \mid \text{try } t \text{ with } t
\end{align*}
\]

Additional Rules:

- Single-step evaluation rules:

\[
\begin{align*}
\text{try } v_1 \text{ with } t_2 & \rightarrow v_1 \quad \text{E-TryV} \\
\text{try error with } t_2 & \rightarrow t_2 \quad \text{E-TryError}
\end{align*}
\]

\[
\begin{align*}
t_1 & \rightarrow t_1' \\
\frac{t_1 \rightarrow t_1'}{\text{try } t_1 \text{ with } t_2 \rightarrow \text{try } t_1' \text{ with } t_2} \quad \text{E-Try}
\end{align*}
\]

- Typing rule:

\[
\begin{align*}
\Gamma \vdash t_1 : T, \Gamma \vdash t_2 : T \\
\frac{\Gamma \vdash \text{try } t_1 \text{ with } t_2 : T}{}
\end{align*}
\]

T-Try
Exceptions Carrying Values

A generalization of the 1st attempt:
Syntax:

\[
\begin{align*}
t & ::= \ldots \quad \text{terms} \\
& | \quad \text{raise } t \\
& | \quad \text{try } t \text{ with } t
\end{align*}
\]

- In "\text{raise } t\)”, \(t\) is the extra information that we want to pass to the handler.
- Note that the extra information is a term and not just a value.
- We will write evaluation rules to ensure that the extra information is fully evaluated before it is passed to the handler.
Exceptions Carrying Values: Evaluation Rules

Additional Rules:

\[
(\text{raise } v_1) t_2 \rightarrow \text{raise } v_1 \quad \text{E-APPRAISE1}
\]

\[
v_1 (\text{raise } v_2) \rightarrow \text{raise } v_2 \quad \text{E-APPRAISE2}
\]

\[
\frac{t_1 \rightarrow t_1'}{\text{raise } t_1 \rightarrow \text{raise } t_2}
\]

\[
\text{E-RAISE}
\]

\[
\text{raise } (\text{raise } v_1) \rightarrow \text{raise } v_1 
\]

\[
\text{E-RAISERAISE}
\]

\[
\text{try } v_1 \text{ with } t_2 \rightarrow v_1 
\]

\[
\text{E-TRYV}
\]

\[
\text{try } (\text{raise } v_1) \text{ with } t_2 \rightarrow t_2 \; v_1
\]

\[
\text{E-TRYRAISE}
\]

\[
\frac{t_1 \rightarrow t_1'}{\text{try } t_1 \text{ with } t_2 \rightarrow \text{try } t_1' \text{ with } t_2}
\]

\[
\text{E-TRY}
\]
Additional Rules:

\[
\frac{\Gamma \vdash t_1 : T_{\text{exn}}} {\Gamma \vdash \text{raise } t_1 : T} \quad \text{T-RAISE}
\]

\[
\frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T_{\text{exn}} \to T} {\Gamma \vdash \text{try } t_1 \text{ with } t_2 : T} \quad \text{T-TRY}
\]
\( T_{\text{exn}} \)

- \( T_{\text{exn}} \) is the type for terms that can be passed as values by exceptions.
- \( T_{\text{exn}} = \text{Nat} \): This is similar to the “error number” convention used in Unix.
- \( T_{\text{exn}} \) is a variant type, e.g.:

\[
T_{\text{exn}} = < \text{divideByZero}: \text{Unit}, \newline \text{overflow}: \text{Unit}, \newline \text{fileNotFound}: \text{String}, \newline ... >
\]

Variant types, by themselves, are insufficient since we will have to declare, \textit{a priori}, all possible exceptions.

- ML uses “extensible variant types” for exceptions. These are similar to variant types, but a programmer can introduce new tags into this type (e.g. using \texttt{exception} declaration).