COMP3610/6361
Principles of Programming Languages

Peter Höfner

Aug 22, 2023

Section 9

Exceptions

Motivation

Trapped errors

Cause execution to halt immediately.
Examples: jumping to an illegal address, raising a top-level exception.
Innocuous?

Untrapped errors

May go unnoticed for a while and later cause arbitrary behaviour.
Examples: accessing data past the end of an array, security loopholes in Java abstract machines.
Insidious!

program should signal error

- devision by zero
- index out of bound (e.g. record type)
- . . .
- lookup key missing
- file not found

Choice 1: Raising Exceptions

Idea: introduce term $\mathbf{error}$ that completely aborts an evaluation of a term.

$$E ::= \ldots \mid \mathbf{error}$$

(no change of values nor types)

(err) $\quad \Gamma \vdash \mathbf{error} : T$

Errors – Semantics

SOS rules

(apperr1) $\langle \mathbf{error}\ E\ ,\ s \rangle \longrightarrow \langle \mathbf{error}\ ,\ s \rangle$

(apperr2) $\langle v\ \mathbf{error}\ ,\ s \rangle \longrightarrow \langle \mathbf{error}\ ,\ s \rangle$

Errors

- $(\textbf{fn}\ x : \text{int} \Rightarrow x)\ \mathbf{error} \rightarrow ?$
- $\textbf{let val rec}\ x : \text{int} \rightarrow \text{int} = (\textbf{fn}\ y : \text{int} \Rightarrow y)\ \textbf{in}\ x\ \mathbf{error}\ \textbf{end} \rightarrow ?$

- $\mathbf{error}$ can have arbitrary type, which violates type uniqueness (can be fixed by subtyping)
- type preservation is maintained
- progress property needs adaptation (homework 2)

Choice 2: Handling Exceptions

Idea: install exception handlers (e.g. ML or Java)

$$E ::= \ldots \mid \textbf{try } E \textbf{ with } E$$

(no change of values nor types)

Handling Exceptions – Typing and Semantics

try E_1 **with** E_2 means 'return result of evaluating E_1, unless it aborts, in which case the handler E_2 is evaluated'

Typing

$$\text{(try)}\quad \frac{\Gamma \vdash E_1 : T \qquad \Gamma \vdash E_2 : T}{\Gamma \vdash \textbf{try}\ E_1\ \textbf{with}\ E_2 : T}$$

SOS rules

$$\text{(try1)}\quad \langle \textbf{try}\ v\ \textbf{with}\ E\ ,\ s\rangle \longrightarrow \langle v\ ,\ s\rangle$$

$$\text{(try2)}\quad \langle \textbf{try}\ \mathbf{error}\ \textbf{with}\ E\ ,\ s\rangle \longrightarrow \langle E\ ,\ s\rangle$$

$$\text{(try3)}\quad \frac{\langle E_1\ ,\ s\rangle \longrightarrow \langle E_1'\ ,\ s'\rangle}{\langle \textbf{try}\ E_1\ \textbf{with}\ E_2\ ,\ s\rangle \longrightarrow \langle \textbf{try}\ E_1'\ \textbf{with}\ E_2\ ,\ s'\rangle}$$

Choice 3: Exceptions with Values

Idea: inform user about type of error

$$E ::= \ldots \mid \sout{\textbf{error}} \mid \textbf{raise}\ E \mid \textbf{try}\ E\ \textbf{with}\ E$$

(no change of values)

Exceptions with Values – Typing

Typing

(try_ex) $$\frac{\Gamma \vdash E : T_{ex}}{\Gamma \vdash \textbf{raise}\ E : T}$$

(try_v) $$\frac{\Gamma \vdash E_1 : T \qquad \Gamma \vdash E_2 : T_{ex} \to T}{\Gamma \vdash \textbf{try}\ E_1\ \textbf{with}\ E_2 : T}$$

Exceptions with Values – Semantics

SOS rules

(apprai1) $\langle (\textbf{raise}\ v)\ E\ ,\ s\rangle \longrightarrow \langle \textbf{raise}\ v\ ,\ s\rangle$

(apprai2) $\langle v_1\ (\textbf{raise}\ v_2)\ ,\ s\rangle \longrightarrow \langle \textbf{raise}\ v_2\ ,\ s\rangle$

(rai) $$\frac{\langle E\ ,\ s\rangle \longrightarrow \langle E_1'\ ,\ s'\rangle}{\langle \textbf{raise}\ E\ ,\ s\rangle \longrightarrow \langle \textbf{raise}\ E'\ ,\ s'\rangle}$$

(rai2) $\langle \textbf{raise}\ (\textbf{raise}\ v)\ ,\ s\rangle \longrightarrow \langle \textbf{raise}\ v\ ,\ s\rangle$

(try1) $\langle \textbf{try}\ v\ \textbf{with}\ E\ ,\ s\rangle \longrightarrow \langle v\ ,\ s\rangle$

(try2) $\langle \textbf{try raise}\ v\ \textbf{with}\ E\ ,\ s\rangle \longrightarrow \langle E\ v\ ,\ s\rangle$

(try3) $$\frac{\langle E_1\ ,\ s\rangle \longrightarrow \langle E_1'\ ,\ s'\rangle}{\langle \textbf{try}\ E_1\ \textbf{with}\ E_2\ ,\ s\rangle \longrightarrow \langle \textbf{try}\ E_1'\ \textbf{with}\ E_2\ ,\ s'\rangle}$$

The Type T_{ex} (I)

- $T_{ex} =$ nat: corresponds to `errno` in Unix OSs;
  0 indicates success; other values report various exceptional conditions.
  (similar in `C++`).
- $T_{ex} =$ string: avoids looking up error codes; more descriptive; error handling may now require parsing a string
- T_{ex} could be of type record

$$
\begin{aligned}
T_{ex} ::= \{&\texttt{dividedByZero} : \text{unit},\\
&\texttt{overflow} : \text{unit},\\
&\texttt{fileNotFound} : \text{string},\\
&\texttt{fileNotReadable} : \text{string},\\
&\ldots\}
\end{aligned}
$$

The Type T_{ex} (II)

- 'T_{ex} in ML': make records more flexible to allow fields to be added, sometimes called *extensible records* or *extensible variant type*
- 'T_{ex} in Java': use of classes, uses keyword `throwable`, which allows the declaration of new errors. (We do not yet know what an object is)
- ...