Today we reviewed the syntax and semantics of (call-by-value) exceptions. We discussed why exceptions can’t be values, how exceptions can “contain values”, and how exceptions should be typed.

1 Introducing Exceptions

The simple lambda calculus:

\[
t ::= b | t \to t
e ::= i | x | \lambda x.e | e \cdot e | \text{error}
v ::= i | \lambda x.e
\]

Rules of small step semantics (call by value):

\[
\begin{align*}
e_1 & \rightarrow e_1' \\
e_1 e_2 & \rightarrow e_1' e_2 \\
e_2 & \rightarrow e_2' \\
v_1 e_2 & \rightarrow v_1 e_2'
\end{align*}
\]

\[
(\lambda x.e_1)v_2 \rightarrow e_1[x := v_2]
\]

How do we insert exceptions? Can’t make an error a value because in the 3rd rule, \(x\) might not be in \(e_1\) and \(e_1\) might continue normally!

We need to reconsider the last two rules’ semantics and add in errors.

\[
\begin{align*}
e_2 & \rightarrow e_2' \\
\text{error } e_2 & \rightarrow \text{error}
\end{align*}
\]

\[
(\lambda x.e_1)\text{error} \rightarrow \text{error}
\]
2 Exception Typing

The problem is we want to be able to do something like:

\[
\text{if } e \text{ then } 17 \text{ else error}
\]

The problem 17 and error must have the same type. So what type does error have? Under other circumstances it must have the same type as a bool.

Can we say:

\[
\Gamma \vdash \text{error} : t
\]

This typing would cause a loss of unicity of typing. But, we lost unicity of typing with lambda expressions but fixed it with a type annotation.

So, we can do the same here (subscript t):

\[
\Gamma \vdash \text{error}_t : t
\]

Notice that this production has no requirements (top) but can have any type. This should only happen when the computation will not return a value (like exceptions).

3 Error Handling

Extend the language with:

\[
e ::= \ldots | \text{try } e \text{ with } e \\
v ::= \ldots | ?
\]
How do we pass errors around like a value?

First, change \texttt{error} to be \texttt{raise}. The \texttt{raise} construct is the error, but the \texttt{e} is the “message”.

Now, change \texttt{try error with e2 \rightarrow e2} to \texttt{try (raise v) with e2 \rightarrow e2 \hat{v}}.

Also, change \texttt{error e2 \rightarrow error} to \texttt{(raise v) e2 \rightarrow raise v}.

And finally, change \texttt{(\lambda x.e1)error \rightarrow error} to \texttt{(\lambda x.e1)(raise v) \rightarrow raise v}.

We also need to define how to evaluate \texttt{raise e}.

\[
\begin{align*}
e & \rightarrow e1 \\
\texttt{raise e} & \rightarrow \texttt{raise e'} \\
\texttt{raise (raise v)} & \rightarrow \texttt{raise v}
\end{align*}
\]

What type does \texttt{raise} have? The Type annotation becomes tedious, so what else can we do?

\[
\begin{align*}
\Gamma \vdash e1 : t & \quad \Gamma \vdash e2 : \texttt{Exn} \rightarrow t \\
\Gamma \vdash \texttt{try e1 with e2} : t
\end{align*}
\]