

Comp 411 Notes - Fri, Feb 11 2005

EXCEPTIONS

Scribe: Seth James Nielson

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:

$$\begin{aligned} t &::= b \mid t \rightarrow t \\ e &::= i \mid x \mid \lambda x. e \mid e \mid \mathbf{error} \\ v &::= i \mid \lambda x. e \end{aligned}$$

Rules of small step semantics (call by value):

$$\frac{e_1 \rightarrow e'_1}{e_1 e_2 \rightarrow e'_1 e_2} \quad \frac{e_2 \rightarrow e'_2}{v_1 e_2 \rightarrow v_1 e'_2} \\ \frac{}{(\lambda x. e_1) v_2 \rightarrow e_1[x := v_2]}$$

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.

$$\frac{e_2 \rightarrow e'_2}{\mathbf{error} e_2 \rightarrow \mathbf{error}} \\ \frac{}{(\lambda x. e_1) \mathbf{error} \rightarrow \mathbf{error}}$$

2 Exception Typing

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

if e then 17 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:

$$\overline{\Gamma \vdash \mathbf{error} : t}$$

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):

$$\overline{\Gamma \vdash \mathbf{error}_t : t}$$

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:

$$\begin{aligned} e &::= \dots | \mathbf{try } e \mathbf{ with } e \\ v &::= \dots | ? \end{aligned}$$

$$\frac{e_1 \rightarrow e'_1}{\text{try } e_1 \text{ with } e_2 \rightarrow \text{try } e'_1 \text{ with } e_2}$$

$$\frac{}{\text{try } v \text{ with } e_2 \rightarrow v}$$

$$\frac{}{\text{try error with } e_2 \rightarrow e_2}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash \text{try } e_1 \text{ with } e_2 : t}$$

$$\frac{\Gamma \vdash e : \text{Exn}}{\Gamma \vdash \text{raise } e : \text{Exn}}$$

How do we pass errors around like a value?

First, change `error` to be `raise e`. The `raise` construct is the error, but the `e` is the “message”.

Now, change $\frac{}{\text{try error with } e_2 \rightarrow e_2}$ to $\frac{}{\text{try } (\text{raise } v) \text{ with } e_2 \rightarrow e_2 v}$.

Also, change $\frac{}{\text{error } e_2 \rightarrow \text{error}}$ to $\frac{}{(\text{raise } v) e_2 \rightarrow \text{raise } v}$

And finally, change $\frac{}{(\lambda x.e_1)\text{error} \rightarrow \text{error}}$ to $\frac{}{(\lambda x.e_1)(\text{raise } v) \rightarrow \text{raise } v}$.

We also need to define how to evaluate `raise e`.

$$\frac{e \rightarrow e_1}{\text{raise } e \rightarrow \text{raise } e'}$$

$$\frac{}{\text{raise } (\text{raise } v) \rightarrow \text{raise } v}$$

What type does `raise v` have? The Type annotation becomes tedious, so what else can we do?

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : \text{Exn} \rightarrow t}{\Gamma \vdash \text{try } e_1 \text{ with } e_2 : t}$$