Exceptions

Advanced Compiler Construction and Program Analysis

Lecture 6 ½

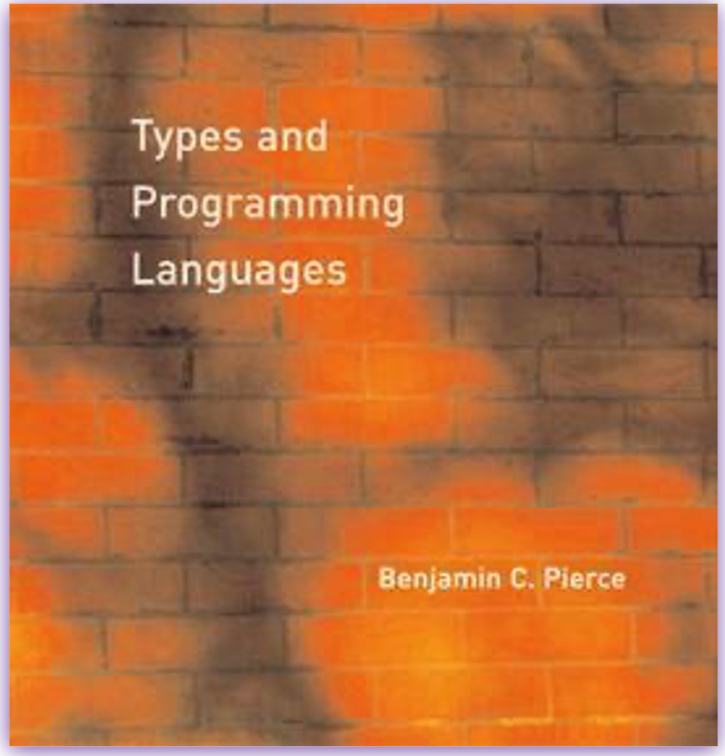
Innopolis University, Spring 2022

The topics of this lecture are covered in detail in…

Benjamin C. Pierce.

Types and Programming Languages

MIT Press 2002



Exceptions: syntax

`t ::= …`	***terms***
`error`	*runtime error*

Exceptions: evaluation

`t ::= …`	***terms***
`error`	*runtime error*

$$\texttt{error}\ t_2 \longrightarrow \texttt{error}$$

Exceptions: evaluation

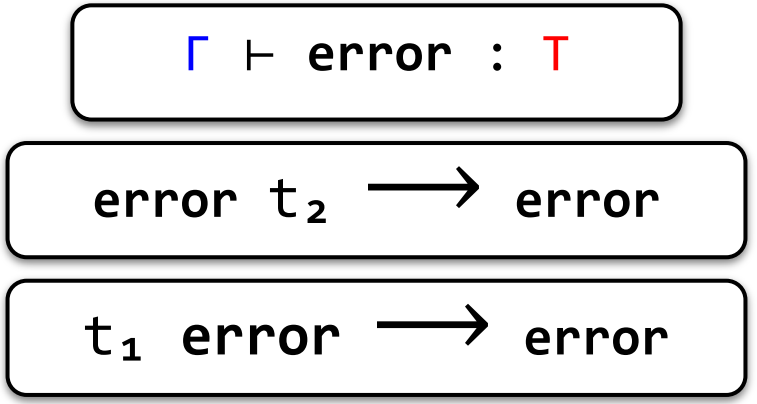
`t ::= …`	***terms***
`error`	*runtime error*

$$\texttt{error}\ t_2 \longrightarrow \texttt{error}$$

$$t_1\ \texttt{error} \longrightarrow \texttt{error}$$

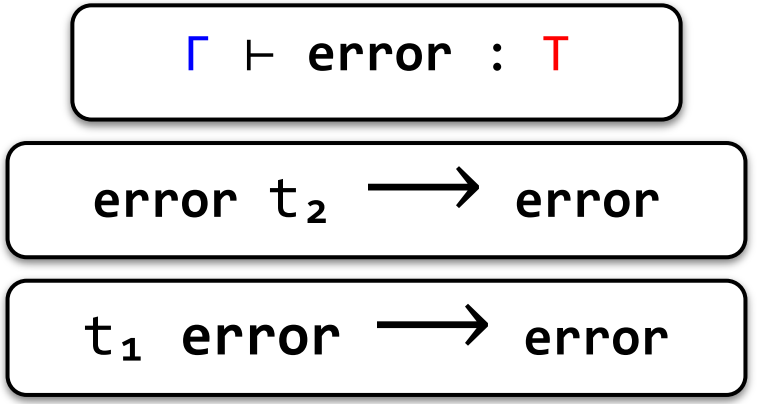
Exceptions: typing

t ::= …	***terms***
error	*runtime error*



Exceptions: evaluation exercise

t ::= …	***terms***
error	*runtime error*

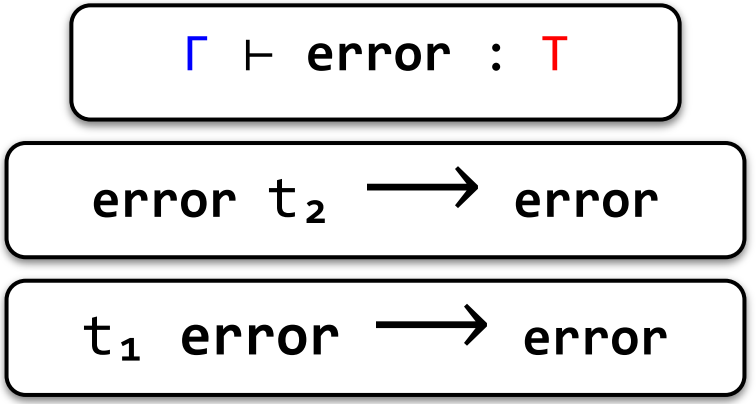


Exercise 6½.1. Explain, how the following terms evaluate:

1. `(λx:Nat.0)` **`error`**
1. `(fix (λx:Nat.x))` **`error`**

Exceptions: typing exercise

t ::= …	***terms***
error	*runtime error*



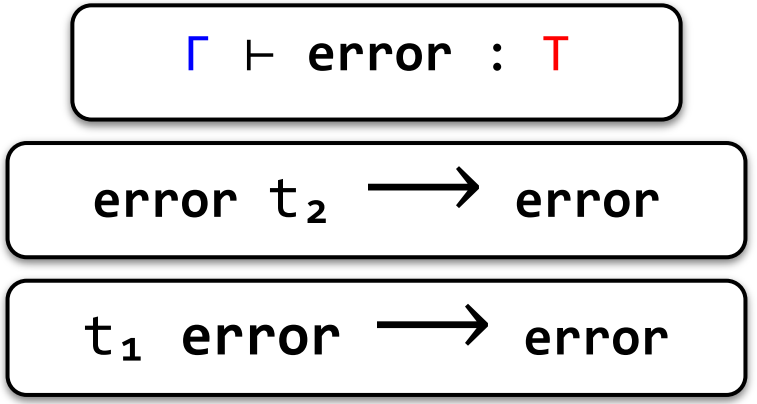
Exercise 6½.2. What is the type of **error** in these terms?

1. (λx:Bool.x) **error**

1. (λx:Bool.x) (**error** true)

Exceptions: type safety

t ::= …	***terms***
error	*runtime error*



Theorem 6½.3 [Progress].
Suppose **t** is a closed, well-typed term. Then either

1. It can evaluate to another term **t'**, or
2. It is a value, or
3. It is **error**

Exceptions: `try-with`

```
t ::= …                          terms
    try t with t                 trap errors
```

$$\texttt{try } v_1 \texttt{ with } t_2 \longrightarrow v_1$$

$$\texttt{try error with } t_2 \longrightarrow t_2$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{try } t_1 \texttt{ with } t_2 \longrightarrow \texttt{try } t_1 \texttt{ with } t_2}$$

$$\frac{\Gamma \vdash t_1 : T \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash \texttt{try } t_1 \texttt{ with } t_2 : T}$$

Exceptions carrying values: syntax

`t ::= …`	***terms***
`raise t`	*raise exception*
`try t catch x⇒t`	*handle exception*

Exceptions carrying values: evaluation

t ::= …	***terms***
raise t	*raise exception*
try t **catch** x⇒t	*handle exception*

$$(\textbf{raise}\ v_1)\ t_2 \longrightarrow \textbf{raise}\ v_1$$

$$t_1\ (\textbf{raise}\ v_2) \longrightarrow \textbf{raise}\ v_2$$

$$\textbf{try}\ v_1\ \textbf{catch}\ x \Rightarrow t_2 \longrightarrow v_1$$

$$\textbf{try}\ (\textbf{raise}\ v_1)\ \textbf{catch}\ x \Rightarrow t_2 \longrightarrow [x \mapsto v_1]t_2$$

Exceptions carrying values: typing

t ::= …	***terms***
raise t	*raise exception*
try t **catch** x⇒t	*handle exception*

$$\frac{\Gamma \vdash t : T_e}{\Gamma \vdash \textbf{raise}\ t : T}$$

$$\frac{\Gamma \vdash t_1 : T \qquad \Gamma, x{:}T_e \vdash t_2 : T}{\Gamma \vdash \textbf{try}\ t_1\ \textbf{catch}\ x \Rightarrow t_2 : T}$$

Exceptions carrying values: typing

```
t ::= …                          terms
      raise t                    raise exception
    try t catch x⇒t              handle exception
```

Type of exception values

$$\frac{\Gamma \vdash t : T_e}{\Gamma \vdash \texttt{raise}\ t : T}$$

$$\frac{\Gamma \vdash t_1 : T \qquad \Gamma, x:T_e \vdash t_2 : T}{\Gamma \vdash \texttt{try}\ t_1\ \texttt{catch}\ x \Rightarrow t_2 : T}$$

Types of exception values (1–3 of 5)

There are several options to choose from:

1. Use **`Nat`** — this would be the code of the error (e.g. like errno convention in Unix)

2. Use **`String`** — flexible, but uncomfortable to handle

3. Use a variant type:

```
Te =  < divideByZero: Unit,
        overflow: Unit,
        fileNotFound: String,
        fileNotReadable: String, … >
```

Types of exception values (4–5 of 5)

There are several options to choose from:

4. Use *extensible* variant type (open sum type).
   E.g. declare distinct exception labels, and then use any in **`raise`** or **`try-catch`**.

5. Rely on subtyping. E.g. in Java all exceptions are subclasses of **`Throwable`** class. Though, this is quite similar to extensible variant types. Except, subtyping also imposes a partial order on the types of exceptions.

Exceptions: more exercises

Exercise 6½.4. Formalize type system with exception types as extensible variants.

Exercise 6½.5. Formalize type system where each function type also carries information about possible exceptions it may throw. Prove that this system is type safe.

Summary

- ❑ Exceptions

See you next time!