

Exceptions (Chapter 14)

CIS 500
 Software Foundations
 Fall 2004
 8 November

Most programming languages provide some mechanism for interrupting the normal flow of control in a program to signal some exceptional condition.

Examples?

Note that it is always **possible** to program without exceptions — instead of raising an exception, we return **None**; instead of returning result **x** normally, we return **Some(x)**. But now we need to wrap every function application in a **case** to find out whether it returned a result or an exception.

→ much more convenient to build this mechanism into the language.

Motivation

- ◆ Midterm II is one week from Wednesday.
- ◆ Give Midterm I regrades (problem 8(b)) to Cheryl by November 15.
- ◆ Homework 7 due today.
- ◆ Homework 8 out today, due November 15.

Announcements

An “abort” primitive in STLC

First step: raising exceptions (but not catching them).

$t ::= \dots$
terms
 error
run-time error

Evaluation

$\text{error } t_2 \rightarrow \text{error}$ (E-APPERR1)
 $v_1 \text{ error} \rightarrow \text{error}$ (E-APPERR2)

◆ What if we had booleans and numbers in the language?

Typing

Typing

$\Gamma \vdash \text{error} : \perp$ (T-ERROR)

Varieties of non-local control

There are **many** ways of adding “non-local control flow”

- ◆ `exit(1)`
- ◆ `goto`
- ◆ `setjmp/longjmp`
- ◆ `raise/try` (or `catch/throw`) in many variations
- ◆ `callcc` / continuations
- ◆ more esoteric variants (cf. many Scheme papers)

Varieties of non-local control

There are **many** ways of adding “non-local control flow”

- ◆ `exit(1)`
 - ◆ `goto`
 - ◆ `setjmp/longjmp`
 - ◆ `raise/try` (or `catch/throw`) in many variations
 - ◆ `callcc` / continuations
 - ◆ more esoteric variants (cf. many Scheme papers)
- Let's begin with the simplest of these.

Typing errors

Note that the typing rule for `error` allows us to give it any type `T`.

$\vdash \text{error} : T$ (T-ERROR)

This means that both

`if x>0 then 5 else error`

and

`if x>0 then true else error`

will typecheck.

Aside: Syntax-directedness

Note that this rule

$\vdash \text{error} : T$ (T-ERROR)

has a problem from the point of view of implementation: it is not syntax-directed!

This will cause the Uniqueness of Types theorem to fail.

For purposes of defining the language and proving its type safety, this is not a problem — Uniqueness of Types is not critical.

Let's think a little, though, about how the rule might be fixed...

An alternative

Can't we just decorate the `error` keyword with its intended type, as we have done to fix related problems with other constructs?

$\vdash \text{error as } T : T$ (T-ERROR)

No, this doesn't work!

E.g. (assuming our language also has numbers and booleans):

`succ (if (error as Bool) then 5 else 7)`
 \rightarrow `succ (error as Bool)`

Exercise: Come up with a similar example using just functions and `error`.

and live with the resulting nondeterminism of the typing relation.

$\Gamma \vdash \text{error} : \text{T}$ (T-ERROR)

Let's stick with the original rule

For now...

In effect, we are replacing the **uniqueness of typing** property by a weaker (but still useful) property called **most general typing**.
 I.e., although a term may have many types, we always have a compact way of **representing** the set of all of its possible types.

$\Gamma \vdash \text{error} : 'a$ (T-ERROR)

In a system with universal polymorphism (like OCaml), the variability of typing for **error** can be dealt with by assigning it a variable type!

Another alternative

The **preservation** theorem requires no changes when we add **error**: if a term of type **T** reduces to **error**, that's fine, since **error** has every type **T**.

Type safety

(Of course, what we've really done is just pushed the complexity of the old **error** rule onto the **Bot** type! We'll return to this point later.)

$\Gamma \vdash \text{error} : \text{Bot}$ (T-ERROR)

Alternatively, in a system with subtyping (which we'll discuss in the next lecture) and a minimal **Bot** type, we **can** give **error** a unique type:

Yet another alternative

Instead, we keep **error** as a non-value normal form, and refine the statement of progress to explicitly mention the possibility that terms may evaluate to **error** instead of to a value.

THEOREM [PROGRESS]: Suppose t is a closed, well-typed normal form. Then either t is a value or $t = \mathbf{error}$.

Progress

The **preservation** theorem requires no changes when we add **error**: if a term of type T reduces to **error**, that's fine, since **error** has every type T .

Progress, though, requires a little more care.

Type safety

$t ::= \dots$
terms
trap errors

Catching exceptions

Evaluation

- (E-TRYV) $\text{try } v_1 \text{ with } t_2 \longrightarrow v_1$
- (E-TRYERRROR) $\text{try error with } t_2 \longrightarrow t_2$
- (E-TRY) $\frac{t_1 \longrightarrow t'_1 \quad \text{try } t_1 \text{ with } t_2 \longrightarrow \text{try } t'_1 \text{ with } t_2}{\text{try } t_1 \text{ with } t_2 \longrightarrow \text{try } t'_1 \text{ with } t_2}$
- (T-TRY) $\frac{\Gamma \vdash \text{try } t_1 \text{ with } t_2 : T}{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T}$

Typing

e.g., the term $(\lambda x:\text{Nat}.0) \text{ error}$ could evaluate to either 0 (which would be wrong) or **error** (which is what we intend).

(E-APPABS) $(\lambda x:T_1. t_12) v_2 \longrightarrow [x \mapsto v_2]t_12$

overlap with our existing computation rule for applications:

(E-APPERRR2) $v_1 \text{ error} \longrightarrow \text{error}$

First, note that we do **not** want to extend the set of values to include **error**, since this would make our new rule for propagating errors through applications.

Progress

Exceptions carrying values

$t ::= \dots$
 $\text{raise } t$
raise exception

Evaluation

(E-APPRAISE1) $(\text{raise } v_{11}) \ t_2 \rightarrow \text{raise } v_{11}$

(E-APPRAISE2) $v_1 \ (\text{raise } v_{21}) \rightarrow \text{raise } v_{21}$

(E-RAISE) $\frac{t_1 \rightarrow t'_1}{\text{raise } t_1 \rightarrow \text{raise } t'_1}$

(E-RAISERAISE) $\text{raise} \ (\text{raise } v_{11}) \rightarrow \text{raise } v_{11}$

Typing

(T-EXN) $\frac{\Gamma \vdash t_1 : \text{Exn}}{\Gamma \vdash \text{raise } t_1 : \text{T}}$

(T-TRY) $\frac{\Gamma \vdash t_1 : \text{T} \quad \Gamma \vdash t_2 : \text{Exn} \rightarrow \text{T}}{\Gamma \vdash \text{try } t_1 \text{ with } t_2 : \text{T}}$

(E-TRYV) $\text{try } v_1 \text{ with } t_2 \rightarrow v_1$

(E-TRYRAISE) $\text{try } \text{raise } v_{11} \text{ with } t_2 \rightarrow t_2 \ v_{11}$

(E-TRY) $\frac{t_1 \rightarrow t'_1}{\text{try } t_1 \text{ with } t_2 \rightarrow \text{try } t'_1 \text{ with } t_2}$