

9 November

Fall 2005

Software Foundations

CIS 500

Exceptions (Chapter 14)

Motivation

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 $\exists(x)$. 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.

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.

An “abort” primitive in STLC

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

$t ::= \dots$

error

run-time error

terms

Evaluation

$\text{error } t_2 \longrightarrow \text{error}$

(E-APPERR1)

$v_1 \text{ error} \longrightarrow \text{error}$

(E-APPERR2)

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

Typing

Typing

$\Gamma \vdash \text{error} : \Gamma$

(T-ERROR)

Typing errors

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

$\Gamma \vdash \text{error} : T$

(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

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

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?

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

(T-ERROR)

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?

$\Gamma \vdash (\text{error as } T) : T$ (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)
  ← succ (error as Bool)
```

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

Let's stick with the original rule

$\Gamma \vdash \text{error} : T$

(T-ERROR)

and live with the resulting nondeterminism of the typing relation.

For now...

Type safety

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

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.

Progress

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.

(E-APPERR2) $v_1 \text{ error} \longrightarrow \text{error}$

overlap with our existing computation rule for applications:

(E-APPABS) $(\lambda x:T_{11}.t_{12}) v_2 \longrightarrow [x \mapsto v_2]t_{12}$

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).

Progress

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 $t = \mathbf{error}$.

Catching exceptions

terms

trap errors

Evaluation

$\text{try } t \text{ with } t$

$t ::= \dots$

$\text{try } v_1 \text{ with } t_2 \rightarrow v_1$

(E-TRYV)

$\text{try error with } t_2 \rightarrow t_2$

(E-TRYERROR)

$t_1 \rightarrow t'_1$

$\text{try } t_1 \text{ with } t_2 \rightarrow \text{try } t'_1 \text{ with } t_2$

(E-TRY)

Typing

$$\frac{\Gamma \vdash \text{try } t_1 \text{ with } t_2 : \tau}{\Gamma \vdash t_1 : \tau \quad \Gamma \vdash t_2 : \tau}$$

(T-TRY)

Exceptions carrying values

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

raise exception

terms

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)

$$\frac{\text{raise } t_1 \rightarrow \text{raise } t'_1}{t_1 \rightarrow t'_1}$$

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

(E-TRYV)

try v₁ with t₂ → v₁

(E-TRYRAISE)

try raise v₁₁ with t₂ → t₂ v₁₁

(E-TRY)

$$\frac{t_1 \rightarrow t'_1}{\text{try } t_1 \text{ with } t_2 \rightarrow \text{try } t'_1 \text{ with } t_2}$$

Typing

$$\frac{\Gamma \vdash t_1 : T_{\text{exn}}}{\Gamma \vdash \text{raise } t_1 : T}$$

(T-EXN)

$$\frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T_{\text{exn}} \rightarrow T}{\Gamma \vdash \text{try } t_1 \text{ with } t_2 : T}$$

(T-TRY)