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 Some(x). But now we need to wrap every function application in a case to find out whether it returned a result or raised an exception in a case to find out whether it returned a result. — much more convenient to build this mechanism into the language.

Exceptions (Chapter 14)
Varieties of non-local control

There are many ways of adding “non-local control” to a language. We’ve seen:

- exit(1)
- goto
- setjmp/longjmp
- raise/try (or catch/throw) in many variations
- callcc / continuations

There are many more esoteric variants (cf. many Scheme papers)

Let’s begin with the simplest of these.

An “abort” primitive in STLC

Let’s begin with the simplest of these:

- callcc / continuations
- raise/try (or catch/throw) in many variations
- exit(1)
- goto

There are many ways of adding “non-local control” to a language.
Aside: Syntax-directedness

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

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

But will this cause the uniqueness of types theorem to fail? 

This will cause the uniqueness of types theorem to fail. 

And this means that both

Note that this rule for error allows us to give it any type.
Another alternative

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

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

(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.)

Yet another alternative

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:

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

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.

For now...

Let's stick with the original rule

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

and live with the resulting nondeterminism of the typing relation.

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`.

CIS 500, 8 November
Typesafety

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.

Progress

Instead, we keep error as a non-value normal form, and rename the statement:

Theorem [Progress]: Suppose \( t \) is a closed, well-typed normal form. If \( t \) is not a value or \( t = \text{error} \), then \( t \) is either a value or \( t = \text{error} \).
Exception carrying values

```
\begin{align*}
&\text{try \texttt{t1} with \texttt{t2} : T} & & \text{try \texttt{t1} with \texttt{t2} : T} \\
&\Downarrow \text{t1} & & \Downarrow \text{t2} : \text{T-exn} \\
&\Downarrow \text{try \texttt{v11} with \texttt{t2} : \text{T-exn}} & & \Downarrow \text{try \texttt{v11} with \texttt{t2} : \text{T-exn}} \\
&\Downarrow \text{t1} & & \Downarrow \text{t2} : \text{T-exn} \\
\end{align*}
```

```
\text{try \texttt{t1} with \texttt{t2} : T} & \text{try \texttt{t1} with \texttt{t2} : T} \\
\Downarrow \text{t1} & \Downarrow \text{t2} : \text{T-exn} \\
\Downarrow \text{try \texttt{v11} with \texttt{t2} : \text{T-exn}} & \Downarrow \text{try \texttt{v11} with \texttt{t2} : \text{T-exn}} \\
\Downarrow \text{t1} & \Downarrow \text{t2} : \text{T-exn} \\
```

```
\text{try \texttt{t1} with \texttt{t2} : T} & \text{try \texttt{t1} with \texttt{t2} : T} \\
\Downarrow \text{t1} & \Downarrow \text{t2} : \text{T-exn} \\
\Downarrow \text{try \texttt{v11} with \texttt{t2} : \text{T-exn}} & \Downarrow \text{try \texttt{v11} with \texttt{t2} : \text{T-exn}} \\
\Downarrow \text{t1} & \Downarrow \text{t2} : \text{T-exn} \\
```

Typing