First a type system

(In this example, exceptions raise constant strings $s$)

\[
\begin{align*}
\tau & ::= \text{bool} \mid \tau \to \tau \mid \tau \times \tau \\
\epsilon & ::= x \mid \text{true} \mid \text{false} \mid \lambda x. \epsilon \mid e_1 \mid (e_1, e_2) \mid e_1 \mid e_2
\end{align*}
\]

\[
\begin{array}{c}
\Gamma \vdash e : \tau \\
\Gamma, x : \tau_1 \vdash e : \tau_2 \\
\Gamma, \lambda x. \epsilon : \tau_1 \to \tau_2 \hline
\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \\
\Gamma \vdash e_1 : \text{bool} \\
\Gamma \vdash e_2 : \tau \\
\Gamma \vdash e_3 : \tau
\end{array}
\]

Add effects

\[
\begin{array}{c}
\epsilon ::= \ldots \text{sets of strings...} \\
\tau ::= \text{bool} \mid \tau \to \tau \mid \tau \times \tau \\
e ::= x \mid \text{true} \mid \text{false} \mid \lambda x. e \mid e_1 \mid e_2 \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash e : \tau_1 \epsilon \\
\Gamma \vdash x : \Gamma(x) ; \emptyset \\
\Gamma \vdash x : \text{true} ; \emptyset \\
\Gamma \vdash x : \text{false} ; \emptyset
\end{array}
\]

Key facts

Soundness: If $\vdash e : \tau$, $\epsilon$ and $e$ raises uncaught exception $s$, then $s \in \epsilon$.

- Corollary to Preservation and Progress (once you define the operational semantics for exceptions)

All effect systems work this way:

- Values effectless
- Functions have latent effects
- Conservative due to control-flow (if and try/handle)
- Often some way to mask effects (here, catch an exception)

Only a couple rules special to this effect system:

- Also, not always sets and $\cup$

Type-and-effect systems

New topic: An elegant framework to extend type systems to track "things that may happen" (effects) during evaluation

Plain-old type systems have judgments like $\Gamma \vdash e : \tau$ to mean:

- $e$ won’t get stuck
- If $e$ produces a value, that value has type $\tau$

Adding effects reuses the "plumbing" of typing rules to compute something about "how $e$ executes"

- There are many things we may want to conservatively approximate
  - Example: What exceptions might get thrown
- All effect systems are very similar, especially treatment of functions
  - Example: All values have no effect since their "computation" does nothing

More general rules

Every effect system also substantially more expressive via appropriate subsumption:

- Typing rule for subeffecting (also useful for Preservation)
- Subtyping of function types is covariant in latent effects

\[
\begin{align*}
\frac{\Gamma \vdash e : \tau ; \epsilon \subseteq \epsilon'}{\Gamma \vdash e : \tau ; \epsilon'}
\end{align*}
\]

Not shown: Also want effect polymorphism (type variables ranging over effects) for higher-order functions like map
Other examples

- Definitely terminates (true) or possibly diverges (false)
  - Give `fix e` effect `false`
  - Give values effect `true`
  - Treat `U as and`
  - No change to rules for functions, pairs, conditionals, etc.
- What type casts might occur
- Are certain variables always accessed in critical sections
- Does code obey a locking protocol
- Does code only access memory regions that haven’t been deallocated
- ...

Really a general way to lift static analysis to higher-order functions

- Key is recognizing “from a mile away” when an effect system is the right tool