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
First a type system

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

$$\tau ::= \text{bool} \mid \tau \rightarrow \tau \mid \tau \ast \tau$$

$$e ::= x \mid \text{true} \mid \text{false} \mid \lambda x.\ e \mid e\ e \mid (e, e) \mid e.1 \mid e.2$$

$$\Gamma \vdash e : \tau$$

$$\begin{align*}
\Gamma &\vdash x : \Gamma(x) \\
\Gamma &\vdash \text{true} : \text{bool} \\
\Gamma &\vdash \text{false} : \text{bool}
\end{align*}$$

$$\begin{align*}
\Gamma, x : \tau_1 &\vdash e : \tau_2 \\
\Gamma &\vdash \lambda x.\ e : \tau_1 \rightarrow \tau_2 \\
\Gamma &\vdash e_1 : \tau_1 \\
\Gamma &\vdash e_2 : \tau_2 \\
\Gamma &\vdash (e_1, e_2) : \tau_1 \ast \tau_2 \\
\Gamma &\vdash e.1 : \tau_1 \\
\Gamma &\vdash e.2 : \tau_2 \\
\end{align*}$$

$$\begin{align*}
\Gamma &\vdash e_1 : \text{bool} \\
\Gamma &\vdash e_2 : \tau \\
\Gamma &\vdash e_3 : \tau \\
\Gamma &\vdash \text{if } e_1\ e_2\ e_3 : \tau \\
\Gamma &\vdash \text{raise } s : \tau
\end{align*}$$

$$\begin{align*}
\Gamma &\vdash e_1 : \tau \\
\Gamma &\vdash e_2 : \tau \\
\Gamma &\vdash \text{try } e_1\ \text{handle } s\ e_2 : \tau
\end{align*}$$
Add effects

\[ \epsilon ::= \ldots \text{sets of strings} \ldots \]

\[ \tau ::= \text{bool} \mid \tau \to \tau \mid \tau \ast \tau \]

\[ e ::= x \mid \text{true} \mid \text{false} \mid \lambda x. \ e \mid e \ e \mid (e, e) \mid e.1 \mid e.2 \]

\[
\begin{align*}
\Gamma \vdash e : \tau ; \epsilon \\
\Gamma \vdash x : \Gamma(x); \emptyset \\
\Gamma \vdash \text{true} : \text{bool}; \emptyset \\
\Gamma \vdash \text{false} : \text{bool}; \emptyset \\
\Gamma, x : \tau_1 \vdash e : \tau_2 ; \epsilon \\
\Gamma \vdash \lambda x. \ e : \tau_1 \to \tau_2 ; \emptyset \\
\Gamma \vdash e_1 : \tau_2 ; \epsilon_1 \\
\Gamma \vdash e_2 : \tau_2 ; \epsilon_2 \\
\Gamma \vdash (e_1, e_2) : \tau_1 \ast \tau_2 ; \epsilon_1 \cup \epsilon_2 \\
\Gamma \vdash e : \tau_1 \ast \tau_2 ; \epsilon \\
\Gamma \vdash e.1 : \tau_1 ; \epsilon \\
\Gamma \vdash e.2 : \tau_2 ; \epsilon \\
\Gamma \vdash e_1 : \text{bool} ; \epsilon_1 \\
\Gamma \vdash e_2 : \tau ; \epsilon_2 \\
\Gamma \vdash e_3 : \tau ; \epsilon_3 \\
\Gamma \vdash \text{if} \ e_1 \ e_2 \ e_3 : \tau ; \epsilon_1 \cup \epsilon_2 \cup \epsilon_3 \\
\Gamma \vdash \text{raise} \ s : \tau ; \{s\} \\
\Gamma \vdash \text{try} \ e_1 \ \text{handle} \ s \ e_2 : \tau ; (\epsilon_1 - \{s\}) \cup \epsilon_2
\end{align*}
\]
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 $\cup$
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

\[ \Gamma \vdash \tau : e; \epsilon \quad \epsilon \subseteq \epsilon' \]
\[ \Gamma \vdash \tau : e; \epsilon' \]
\[ \tau_3 \leq \tau_1 \quad \tau_2 \leq \tau_4 \quad \epsilon \subseteq \epsilon' \]

\[ \tau_1 \epsilon \Rightarrow \tau_2 \leq \tau_3 \epsilon' \Rightarrow \tau_4 \]

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 `∪` 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