CSE-505: Programming Languages

Lecture 19 — Type-and-Effect Systems

Zach Tatlock
2016

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 τ

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)

$$\begin{array}{rcl}
\tau & ::= & \textbf{bool} \mid \tau \to \tau \mid \tau * \tau \\
e & ::= & x \mid \textbf{true} \mid \textbf{false} \mid \lambda x.\ e \mid e\ e \mid (e, e) \mid e.1 \mid e.2 \\
& \mid & \textbf{if}\ e\ e\ e \mid \textbf{raise}\ s \mid \textbf{try}\ e\ \textbf{handle}\ s\ e
\end{array}$$

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{}{\Gamma \vdash \textbf{true} : \textbf{bool}} \qquad \frac{}{\Gamma \vdash \textbf{false} : \textbf{bool}}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.\ e : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau_1}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2} \qquad \frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash e.1 : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash e.2 : \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \textbf{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \textbf{if}\ e_1\ e_2\ e_3 : \tau}$$

$$\frac{}{\Gamma \vdash \textbf{raise}\ s : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \textbf{try}\ e_1\ \textbf{handle}\ s\ e_2 : \tau}$$

Add effects

$$\begin{array}{lcl} \epsilon & ::= & \text{...sets of strings...} \\ \tau & ::= & \textbf{bool} \mid \tau \xrightarrow{\epsilon} \tau \mid \tau * \tau \\ e & ::= & x \mid \textbf{true} \mid \textbf{false} \mid \lambda x.\ e \mid e\ e \mid (e, e) \mid e.1 \mid e.2 \\ & \mid & \textbf{if}\ e\ e\ e \mid \textbf{raise}\ s \mid \textbf{try}\ e\ \textbf{handle}\ s\ e \end{array}$$

$$\boxed{\Gamma \vdash e : \tau; \epsilon} \qquad \frac{}{\Gamma \vdash x : \Gamma(x); \emptyset} \qquad \frac{}{\Gamma \vdash \textbf{true} : \textbf{bool}; \emptyset} \qquad \frac{}{\Gamma \vdash \textbf{false} : \textbf{bool}; \emptyset}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2; \epsilon}{\Gamma \vdash \lambda x.\ e : \tau_1 \xrightarrow{\epsilon} \tau_2; \emptyset} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \xrightarrow{\epsilon_3} \tau_1; \epsilon_1 \qquad \Gamma \vdash e_2 : \tau_2; \epsilon_2}{\Gamma \vdash e_1\ e_2 : \tau_1; \epsilon_1 \cup \epsilon_2 \cup \epsilon_3}$$

$$\frac{\Gamma \vdash e_1 : \tau_1; \epsilon_1 \qquad \Gamma \vdash e_2 : \tau_2; \epsilon_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2; \epsilon_1 \cup \epsilon_2} \qquad \frac{\Gamma \vdash e : \tau_1 * \tau_2; \epsilon}{\Gamma \vdash e.1 : \tau_1; \epsilon} \qquad \frac{\Gamma \vdash e : \tau_1 * \tau_2; \epsilon}{\Gamma \vdash e.2 : \tau_2; \epsilon}$$

$$\frac{\Gamma \vdash e_1 : \textbf{bool}; \epsilon_1 \qquad \Gamma \vdash e_2 : \tau; \epsilon_2 \qquad \Gamma \vdash e_3 : \tau; \epsilon_3}{\Gamma \vdash \textbf{if}\ e_1\ e_2\ e_3 : \tau; \epsilon_1 \cup \epsilon_2 \cup \epsilon_3}$$

$$\frac{}{\Gamma \vdash \textbf{raise}\ s : \tau; \{s\}} \qquad \frac{\Gamma \vdash e_1 : \tau; \epsilon_1 \qquad \Gamma \vdash e_2 : \tau; \epsilon_2}{\Gamma \vdash \textbf{try}\ e_1\ \textbf{handle}\ s\ e_2 : \tau; (\epsilon_1 - \{s\}) \cup \epsilon_2}$$

Key facts

Soundness: If $\cdot \vdash e : \tau; \epsilon$ 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

$$\frac{\Gamma \vdash \tau : e; \epsilon \qquad \epsilon \subseteq \epsilon'}{\Gamma \vdash \tau : e; \epsilon'} \qquad \frac{\tau_3 \leq \tau_1 \qquad \tau_2 \leq \tau_4 \qquad \epsilon \subseteq \epsilon'}{\tau_1 \xrightarrow{\epsilon} \tau_2 \leq \tau_3 \xrightarrow{\epsilon'} \tau_4}$$

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 $\cup$ 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