

Effect Systems as Abstract Interpretations

Colin S. Gordon
NSAD 2026



DREXEL UNIVERSITY

School of

Computer and
Information Sciences

Howley College of Engineering and Computing

Types and Abstract Interpretations

- Many approaches exist to drawing static conclusions about programs
- Understanding their relationships *formally* is an important part of understanding trade-offs
- Types vs Abstract Interpretations are a recurring theme
 - Cousot (1997) captures several typed lambda calculi as abstract interpretations
 - Many people have argued that all type systems are abstract interpretations
 - (Seems wrong for “full-spectrum” dependent type theories, especially Homotopy Type Theory, but that’s a longer discussion for another time.)
 - This is a fruitful perspective, even for type theorists
 - Type inference
 - Alternative machinery for soundness

An Incomplete History of Effect Systems as Abstract Interpretations

- Vouillon and Jouvelot (1995): [Unpublished report](#) develops equivalent abstract interpretation and effect system for time complexity analysis
- Skalka and colleagues (2008, 2020) build effect systems that essentially perform trace-based abstract interpretation
- Tang and Jouvelot (1994), Holdermans and Hage (2010), cover control flow analysis via effect systems
- Hofmann and Chen (2014) build an abstract interpretation from Büchi automata, use that as an effect system
- Nicola et al. (2025) explicitly construct an effect system for policy checking as an abstract interpretation

Many hints of a deeper connection, but no systematic relationship

Cousot's (1997) Types as Abstract Interpretations

- Take simply-typed lambda calculus
- Notice domain-theoretic semantics are already a domain
- Denote terms as (meta-)functions from environment to values
- Types form an abstract domain
- Static type environments form an abstract domain
- Can give a *typing* — pair of type env. and type — as abstract domain
- Build the Galois connection
- Tada! Types (for STLC) are abstract interpretations!
- Then he revised the constructions to model polymorphism

We will follow this lead, but stop short of polymorphism

Type-and-Effect Systems

Broadly, classify the *side effects of interest* for an expression

May track accessed locks, memory regions, *checked exceptions* thrown, message ordering, atomicity.... and much more

Augmented type judgement: $\Gamma \vdash e : \tau \mid X$

- “Under assumptions Γ , e has type τ and side effects X ”

$$\begin{array}{c} \text{T-UNIT} \\ \hline \Gamma \vdash \text{tt} : \text{unit} \mid I \end{array} \qquad \begin{array}{c} \text{T-VAR} \\ \Gamma(x) = \tau \\ \hline \Gamma \vdash x : \tau \mid I \end{array}$$

$$\begin{array}{c} \text{T-LAMBDA} \\ \Gamma, x : \tau \vdash e : \tau' \mid \chi \\ \hline \Gamma \vdash (\lambda x : \tau. e) : \tau \xrightarrow{\chi} \tau' \mid I \end{array}$$

$$\text{T-APP} \frac{\Gamma \vdash e_1 : \tau \xrightarrow{\chi} \tau' \mid \chi_1 \quad \Gamma \vdash e_2 : \tau \mid \chi_2}{\Gamma \vdash e_1 e_2 : \tau' \mid \chi_1 \triangleright \chi_2 \triangleright \chi}$$

$$\text{T-EV} \frac{}{\Gamma \vdash \text{ev}(\chi) : \text{unit} \mid \chi}$$

$$\text{T-BRANCH} \frac{\Gamma \vdash e_1 : \tau \mid \chi_1 \quad \Gamma \vdash e_2 : \tau \mid \chi_2}{\Gamma \vdash \text{if}^* e_1 e_2 : \tau \mid \chi_1 \sqcup \chi_2}$$

$$\text{T-REC} \frac{\Gamma, f : \tau \xrightarrow{\chi} \tau', x : \tau \vdash e : \tau' \mid \chi}{\Gamma \vdash (\text{rec}_f x : \tau. e) : \tau \xrightarrow{\chi} \tau' \mid I}$$

$$\text{T-SUB} \frac{\Gamma \vdash e : \tau \mid \chi \quad \chi \sqsubseteq \chi'}{\Gamma \vdash e : \tau \mid \chi'}$$

Effect Quantales (Gordon 2017, 2021)

An algebraic model of flow-sensitive effects

- A structure $Q = \langle E, \triangleright, \sqcup, I \rangle$ where
 - $\langle E, \triangleright, I \rangle$ is a *partial* monoid (i.e., program order)
 - $\langle E, \sqcup \rangle$ is a *partial* join semilattice (i.e., non-determinism)
 - $\triangleright$ distributes over $\sqcup$
- Extends to *iterable effect quantales*, with a (partial) finite iteration
- Captures most (non-dependent) flow-sensitive effects in the literature
- Subsumes flow-insensitive bounded-join-semilattice model
- General treatments for control operators (2020) and error reporting (2023)

Soundness for Effect Quantales (Original)

Gordon (2017, 2021) gives a soundness guarantee of the form:

When for a closed program, $\vdash e : \tau \mid X$

If $e \rightarrow^{a_1} e' \rightarrow^{a_2} \dots \rightarrow^{a_n} v$,

Then $a_1 \triangleright a_2 \triangleright \dots \triangleright a_n \sqsubseteq X$

Or if $e \rightarrow^{a_1} e' \rightarrow^{a_2} \dots \rightarrow^{a_n} e_n$,

Then $\exists X', a_1 \triangleright a_2 \triangleright \dots \triangleright a_n \triangleright X' \sqsubseteq X$ and $\vdash e_n : \tau \mid X'$

i.e., an effect is a property of execution event sequences

Some Denotational Semantics

$W = \{\omega\}$	error
$U = \text{tt}$	unit
$f \in V \simeq W_{\perp} \oplus U_{\perp} \oplus [V \rightarrow P(V \times Q^*)]_{\perp}$	values
$R \in \mathbb{R} = X \rightarrow V$	environments
$\phi \in \mathbb{S} = \mathbb{R} \rightarrow P(V \times Q^*)$	sem. domains

$S[-] : \text{Expression} \rightarrow \mathbb{S}$
$S[\text{tt}] = \Lambda R. \{ \langle \uparrow \text{tt}, I \rangle \}$
$S[x] = \Lambda R. \{ \langle R(x), I \rangle \}$
$S[\lambda x : \tau. e] = \Lambda R. \{ \langle \uparrow (\Lambda \hat{x}. \hat{x} \gg (S[e](R[x \leftarrow \hat{x}]))) , I \rangle \}$
$S[e_1 e_2] = \Lambda R. S[e_1](R) \gg (\Lambda \langle f, T_1 \rangle.$
$(f = \perp \vee f = \Omega) ? \langle f, T_1 \rangle :$
$S[e_2](R) \gg (\Lambda \langle a, T_2 \rangle.$
$(a = \perp \vee a = \Omega) ? \langle a, T_1 \cdot T_2 \rangle :$
$(\text{nonfunc}(f) ? \langle \Omega, T_1 \cdot T_2 \rangle :$
$\text{let } \langle u, T_3 \rangle = f \ a \text{ in } \langle u, T_1 \cdot T_2 \cdot T_3 \rangle)))$
$S[\text{ev}(\chi)] = \Lambda R. \{ \langle \uparrow \text{tt}, \chi \rangle \}$
$S[\text{if}^* e_t e_f] = \Lambda R. S[e_t](R) \cup S[e_f](R)$
$S[\text{rec}_f x : \tau. e] = \Lambda R. \text{lf}_{\uparrow(\Lambda \hat{x}. \emptyset)} (\Lambda \phi. S[\lambda x : \tau. e](R[f \leftarrow \phi]))$

Effects as Abstract Domains (Setup)

Effect quantales are already orders, but..

$$\gamma_0(-) : Q \rightarrow (\mathbb{P}(Q^*))$$
$$\gamma_0(\chi) = \{T \mid \text{fold} \triangleright I T \sqsubseteq \chi\}$$

- The natural abstract operations (post-composition) are partial
- The ordering is partial

Could use partial Galois connections!

But then we'd have to adapt any other abstract interpretation results later.

Instead, complete for a normal Galois connection:

1. Complete the monoid
2. Complete the order

Completion, in Two Steps

1. Monoid completion

- a. Add an extra element $\star$, roughly “did stuff in a bad order”
- b. Initially unordered w.r.t. other elements

2. Order completion

- a. Standard / Dedekind-MacNeille completion
- b. Elements are now set S such that $(S^u)^l = S$
- c. Inject $x \in Q$ as $\downarrow x$
- d. Joins of unordered elements (e.g., of original element and $\star$) are now Q

We call the result $\mathbb{C}(Q)$

Effects as Abstract Domains

$$\gamma(-) : \mathbb{C}(Q) \rightarrow \mathbb{P}(Q^*)$$
$$\gamma(S) = \bigcup_{\chi \in S} \{T \mid \text{fold} \triangleright I \ T \sqsubseteq \chi\}$$

Same general idea:

$\chi \in Q$ embeds as set of sequences that $\triangleright$ -compose to $\sqsubseteq \chi$

But now we embed $\mathbb{C}(Q)$, which is a set of elements (incl. $\star$)

So we union the naive approach over everything in the set

Abstraction is simply the corresponding most precise operator

Typings, including effects

- Types and environments are concretized as in Cousot (1997)
 - Except latent effects
- Typings pick up effects
- Types and effects are independent

$$\Gamma \vdash e : \tau \mid \chi \equiv \langle \Gamma, \tau, \downarrow \chi \rangle \in \mathbb{T}[[e]] \wedge \chi \in Q$$

$$\begin{aligned} \gamma(-) &: \text{Type} \rightarrow \mathbb{P}(\mathbb{V}) \\ \gamma(\text{unit}) &= \{\text{tt}\} \\ \gamma(\tau \xrightarrow{\chi} \tau') &= \{\langle f, T \rangle \mid \forall v \in \gamma(\tau). f \ v \in (\gamma(\tau') \times \gamma(\chi))\} \\ \gamma(-) &: \text{TypeEnvironment} \rightarrow \mathbb{P}(\mathbb{R}) \\ \gamma(\epsilon) &= \{\emptyset\} \text{ (i.e., the singleton set with empty function)} \\ \gamma(\Gamma, x : \tau) &= \{R \in \mathbb{R} \mid \exists R' \in \gamma(\Gamma). \exists v \in \gamma(\tau). R = R[x \mapsto v]\} \\ \gamma(-) &: \text{Typing} \rightarrow \mathbb{P}(\mathbb{V} \times Q^*) \\ \gamma(\langle \Gamma, \tau, \chi \rangle) &= \{\phi \in \mathbb{S} \mid \forall R \in \gamma(\Gamma). \phi(R) \in \gamma(\tau) \times \gamma(\chi)\} \\ \gamma(-) &: \text{ProgramType} \rightarrow \mathbb{P}(\mathbb{S}) \\ \gamma(\bar{T}) &= \bigcap \{\gamma(\langle \Gamma, \tau, \chi \rangle) \mid \langle \Gamma, \tau, \chi \rangle \in \bar{T}\} \end{aligned}$$

Figure 3. Concretizing parts of a type-and-effect system.

Lemma 5.1 (Type Safety). *If $\Gamma \vdash e : \tau \mid \chi$, then*

$$S[[e]] \subseteq \gamma(\langle \Gamma, \tau, \downarrow \chi \rangle)$$

or stated equivalently,

$$\forall \phi \in S[[e]]. \forall R \in \gamma(\Gamma). \phi(R) \in \gamma(\tau) \times \gamma(\downarrow \chi)$$

So Where Does This Get Us?

- A *systematic* relationship:
 - all effect quantale systems can be abstract domains
- An alternative formulation of soundness for sequential effects
 - Paper has safety proof of type rules
 - Similar to *semantic/logical type soundness*
- A foothold on bridging rich theory of abstract interpretation *systematically* for uses in type-and-effect systems
 - Paper compares standard abstract interpretation derivation of iterative approximation to effect quantale iteration