OREGON
PROGRAMMING
LANGUAGES
SUMMER
SCHOOL

Algebraic Effects and Handlers — Ningning Xie

Lecture 1 - *June 30, 2025*

Acknowledgments

- *Lectures on Algebraic Effects*, Gordon Plotkin, NII Shonan meeting No. 146, 2019
- *Effect-Handler Oriented Programming*, Sam Lindley, OPLSS'22
- Collaborators: Daan Leijen, Jonathan Brachthäuser, Daniel Hillerström, Philipp Schuster, Youyou Cong, Kazuki Ikemori, Daniel D. Johnson, Dougal Maclaurin, Adam Paszke, Gordon Plotkin

Motivation

Effects include input/output, exceptions, concurrency, mutable state, etc. They are often implemented in an ad-hoc manner.

Algebraic Effects and Effect Handlers

Effect handlers are composable and structured control-flow abstractions introduced by Plotkin and Pretnar (ESOP 2009).

Algebraic Effects

An algebraic effect consists of:

- Operations (effect constructors) with signatures
- A set of axioms

Example: Boolean Location

- Signatures: $\mathrm{put}_t : 1, \mathrm{put}_f : 1, get : 2$
- Axioms:
  - $get(m, m) = m$
  - $get(get(m, m'), get(n, n')) = get(m, n')$
  - $\mathrm{put}_b(\mathrm{put}_{b'}(m)) = \mathrm{put}_{b'}(m)$
  - $get(\mathrm{put}_t(m), \mathrm{put}_f(n)) = get(m, n)$
  - $\mathrm{put}_b(get(m_t, m_f)) = \mathrm{put}_b(m_b)$

Interpretations

- $T_b(X) = B \to (X \times B)$: This interpretation threads a boolean state through the computation. It is sound and complete because all axioms hold and only the axioms hold: for any two terms m and n, $m = n$ iff their interpretations are equal.

  Example:
  $$[\![\mathrm{put}_t(get(m_t, m_f))]\!] = \lambda s.\, [\![m_t]\!]t = [\![\mathrm{put}_t(m_t)]\!]$$

- $T_{log}(X) = B \to (X \times \text{List } B)$: This logs the entire state trace. It is complete (all equal terms have equal interpretations), but not sound, since terms with different logging behaviors may be semantically distinguished even if they are provably equal in the theory.

  Counterexample:
  $$[\![\mathrm{put}_b(\mathrm{put}_{b'}(m))]\!] \neq [\![\mathrm{put}_{b'}(m)]\!]$$

  The left logs two entries, the right logs one.

- $T_{discard}(X) = B \to X$: This discards the state entirely. It is sound (it preserves all equalities), but not complete, because terms that are not provably equal may still map to the same function, losing information about the effect.

  Example:
  $$[\![\mathrm{put}_b(m)]\!] = [\![m]\!] \quad \text{but} \quad \mathrm{put}_b(m) \neq m$$

Other Examples

- Exception: signature $raise_e : 0$, interpretation $X + E$
- Non-determinism: $or : 2$, axioms of associativity, commutativity, absorption; interpretation: finite nonempty subsets of X

What is Algebraic about Algebraic Effects? The term "algebraic" stems from the correspondence with algebraic theories in universal algebra:

- Each effect is described by operations (like functions in algebra) and equational laws (axioms).
- These operations are required to distribute over evaluation contexts, respecting their structure.
- Categorically, algebraic effects correspond to free models (initial algebras) of these theories.

Algebraicity in Programming Terms Algebraicity means that effectful operations behave uniformly under evaluation contexts:

$$\textbf{Evaluation context } E ::= \square \mid E\ n \mid (\lambda x.\ m)\ E$$

$$E[op(m_1, \ldots, m_n)] = op(E[m_1], \ldots, E[m_n])$$

This ensures that:

- The operational behavior of effects is predictable and modular.
- Effects are context-independent, enabling compositional semantics.

Categorical Viewpoint In category theory, an algebraic theory corresponds to a monad T generated freely by operations and equations.

Fix a finitary equational axiomatic theory Ax. Then for any set X and operation symbol $\mathrm{op} : n$ we have the function:

$$T_{\mathrm{Ax}}(X)^n \xrightarrow{\mathrm{op}_{F_{\mathrm{Ax}}(X)}} T_{\mathrm{Ax}}(X)$$

Further for any function $f : X \to T_{\mathrm{Ax}}(Y)$, $f^\dagger$ is a homomorphism:

$$\begin{tikzcd}
T_{\mathrm{Ax}}(X)^n \arrow[r, "\mathrm{op}_{F_{\mathrm{Ax}}(X)}"] \arrow[d, "(f^\dagger)^n"'] \arrow[dr, phantom, "="] & T_{\mathrm{Ax}}(X) \arrow[d, "f^\dagger"] \\
T_{\mathrm{Ax}}(Y)^n \arrow[r, "\mathrm{op}_{F_{\mathrm{Ax}}(Y)}"'] & T_{\mathrm{Ax}}(Y)
\end{tikzcd}$$

We call such a polymorphic family of functions $T_{\mathrm{Ax}}(X)^n \xrightarrow{\varphi_X} T_{\mathrm{Ax}}(X)$ algebraic.

Example: Algebraicity of Exception

- $raise_e()\ n = raise_e()$ (operation ignores context)
- $(\lambda x.\ m)\ raise_e() = raise_e()$ (context propagation)

Example: Algebraicity of Nondeterminism

- $or(m, n)\ p = or(m\ p, n\ p)$
- $(\lambda x.\ p)\ or(m, n) = or((\lambda x.\ p)\ m, (\lambda x.\ p)\ n)$

Is a set of axioms the right set of axioms?

- Equationally inconsistent: proves $x = y$
- Hilbert-Post complete: adding any unprovable equation makes theory inconsistent

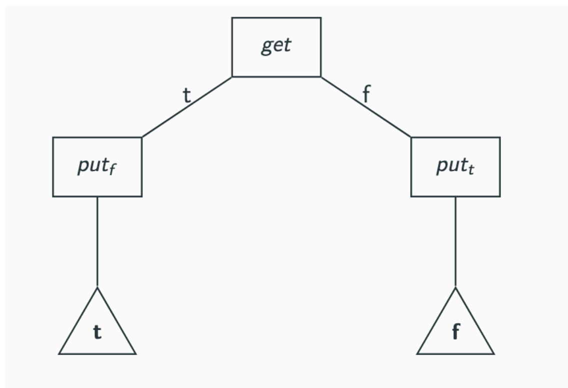
Computational Trees and Free Monads

A computational tree is a tree representing the structure of a computation:

- **Leaves** represent return values (i.e., pure computations).
- **Internal nodes** represent effectful operations (e.g., `get`, `put`, `raise`).

This makes the control structure of effectful computations explicit and manipulable.

Example Term toggle $= get(\text{put}_f(t), \text{put}_t(f))$ builds a tree with **get** at the root and two branches for **putf** and **putt** subcomputations.



Tree Semantics This representation lets us:

- Inspect or transform effectful programs (e.g., optimization, reasoning).
- Compose computations structurally via monadic bind.
- Reify effect structure for later interpretation.

Free Monad Representation In Haskell-style pseudocode:

```
data Free f a = Pure a | Free (f (Free f a))
```

- `Pure a` represents a pure return value.
- `Free f` wraps an operation whose continuation is another `Free` computation.

This structure corresponds to the free monad over a signature functor `f` encoding the effect operations.

Bind Operation Bind recursively substitutes subtrees:

```
return c >>= r = r c
op(m1,...,mn) >>= r = op(m1 >>= r, ..., mn >>= r)
```

This operation:

- Implements substitution of computations into leaves.
- Respects the algebraic structure imposed by operations.

Category-Theoretic Insight Free monads correspond to initial algebras for a given effect signature:

- The free monad `Free f a` generates all computations involving `f`-effects.
- Any interpretation of effects (e.g., into state machines, evaluators) is a monad morphism from `Free f`.

This gives a modular and uniform way to define operational semantics, denotational semantics, and interpreters.