# Local operations can't change the world

Leo White

Jane Street

## Effects as local operations

Locality captures the general idea of a value depending on some notion of "location". We can represent it in a type system using a comonadic *global modality* $\Box$ that tracks which values do not depend on their location. A particularly useful notion of "location" is the ambient monad of a language – the monad describing which effects ordinary functions can perform.

This allows a language, such as OxCaml[2], whose runtime supports algebraic effects and whose type system can track locality, to provide an isomorphism:

$$M(\Box\, X) \qquad \cong \qquad Y \to \Box\, X$$

where $M$ is any algebraic monad, $Y$ is a specific type related to that monad, and $\Box$ is a comonadic global modality.

The ability to use an ordinary arrow type in place of a monad allows for conveniently writing effectful code in direct style, just as algebraic effects do. In OxCaml, we use this to replace our futures monad `'a Deferred.t` with functions of type `Async.t @ local -> 'a`. You can picture the `Async.t` as a collection of effectful operations. The `@ local` mode annotation constrains where those operations can be used, and thus contains the extent of their effect: local operations can't change the world.

## Presheaves for algebraic effects

Our language falls out naturally from its categorical semantics, so we focus on that. We intend to index the meaning of types by an ambient algebraic monad of effects: rather than have the semantics of each type be a set of elements, the semantics of each type is a mapping from the ambient monad to a set of elements.

For a category $\mathbb{C}$, $\widehat{\mathbb{C}} = \mathrm{Set}^{\mathbb{C}^{\mathrm{op}}}$ is the category of presheaves over $\mathbb{C}$. If $\mathbb{C}$ has a terminal object $\top$, we can define the global modality $\Box : [\widehat{\mathbb{C}}, \widehat{\mathbb{C}}]$ such that:

$$\Box(P)(L) = P(\top)$$

We use the ambient monad as our notion of location, so we take $\mathbb{C}$ to be $\mathrm{Law}^{\mathtt{op}}$, where Law is the category of countable Lawvere theories. Lawvere theories are a categorical representation of algebraic theories. Morphisms in Law are maps from the operations of one theory to the operations of another that preserve the equations of the first theory according to the equations of the second. The initial theory is the algebraic theory with no operations. The coproduct of two theories is the theory made by combining the operations and equations of both theories.

Given a Lawvere theory $L$, the underlying set of its free algebra forms a monad $T_L$ on Set. $T_{\_}$ is functorial and we can use it to construct a monad $\mathcal{T}$ on $\widehat{\mathrm{Law}^{\mathtt{op}}}$ in the obvious way, such that:

$$\mathcal{T}(P)(L) = T_L(P(L))$$

We will use $\mathcal{T}$ as our computation monad.

$\mathrm{Law}^{\mathtt{op}}$ embeds into $\widehat{\mathrm{Law}^{\mathtt{op}}}$ via the Yoneda embedding. For each Lawvere theory $L$ there is a presheaf $y(L)$ in $\widehat{\mathrm{Law}^{\mathtt{op}}}$ such that:

$$y(L)(L') = \mathrm{Hom}_{\mathrm{Law}}(L, L')$$

We will use $y(L)$ to represent maps from $L$ into the ambient monad. Note that $y$ preserves finite products: $y(L+L') \cong y(L) \times y(L')$ where $+$ is the coproduct in Law, which is the product in $\mathrm{Law}^{\mathtt{op}}$.

Categories of presheaves are cartesian closed, and the exponential $P \Rightarrow Q$ in $\widehat{\mathrm{Law}^{\mathtt{op}}}$ maps $L$ to $\mathrm{Hom}_{\widehat{\mathrm{Law}^{\mathtt{op}}}}(y(L) \times P, Q)$. So, via the Yoneda lemma, we get the following:

$$(y(L) \Rightarrow \mathcal{T}(\Box\, A))(L') \quad \cong \quad T_{(L'+L)}(A(\top))$$

which will form the basis of our language's isomorphisms between arrows $Y \to \Box\, A$ and algebraic monads $M(\Box\, A)$.

## A language with reflection for algebraic monads

We build a simple language to (mostly) achieve our aim. Rather than working with arbitrary algebraic monads as promised, we follow the finest traditions of the algebraic effects literature and restrict our attention to algebraic theories without equations. Our language is simply typed, featuring all the usual type formers. It uses a variation of fine-grained call-by-value: it has separate syntax and typing judgements for values and computations.

We define algebraic signatures $\Sigma$ as finite sets of operations

$$\{op_1 : \tau_{p_1} \rightarrowtail \tau_{a_1}; \ldots; op_n : \tau_{p_n} \rightarrowtail \tau_{a_n}\}$$

An operation type $\tau_p \rightarrowtail \tau_a$ indicates an operation parameterised by $\tau_p$ and with arity $\tau_a$. Such signatures correspond to Lawvere theories without equations. We restrict parameters and arities to base types to ensure that our theories only have a countable number of operations of countable arities. We write $[\![\Sigma]\!]$ for the corresponding Lawvere theory, and $F_\Sigma$ for the endofunctor on Set for which $T_{[\![\Sigma]\!]}$ is the free monad.

The language provides inductive types $W_\Sigma$ to directly represent terms of an algebraic signature $\Sigma$. $W_\Sigma(\tau)$ has a constructor $op$ with arguments of type $\tau_p$ and $\tau_a \to W_\Sigma(\tau)$ for each operation $op : \tau_p \rightarrowtail \tau_a$ in $\Sigma$, as well as a constructor $ret$ with an argument of type $\tau$.

The language provides types $y(\Sigma)$ to represent morphisms from $[\![\Sigma]\!]$ into the ambient monad. Using these, our desired isomorphism appears in the langauge as two operations, reflection $\Downarrow_\Sigma (c)$ and reification $\Uparrow_\Sigma (v)$.

$$\frac{\Gamma \vdash_C c : W_\Sigma(\Box\, \tau)}{\Gamma \vdash_V \Downarrow_\Sigma (c) : y(\Sigma) \to \Box\, \tau} \qquad \frac{\Gamma \vdash_V v : y(\Sigma) \to \Box\, \tau}{\Gamma \vdash_C \Uparrow_\Sigma (v) : W_\Sigma(\Box\, \tau)}$$

Our denotational semantics in terms of $\widehat{\mathrm{Law}^{\mathtt{op}}}$ is straightforward. For types, the translation $[\![\tau]\!] : \widehat{\mathrm{Law}^{\mathtt{op}}}$ is as follows:

$$[\![y(\Sigma)]\!] = y([\![\Sigma]\!]) \qquad [\![\Box\, \tau]\!] = \Box [\![\tau]\!] \qquad [\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \Rightarrow \mathcal{T}([\![\tau_2]\!])$$

$$[\![W_\Sigma(\tau)]\!] = \mu X.([\![\tau]\!] + F_\Sigma(\mathcal{T}(X)))$$

where $\mu X.F(X).$ represents the initial algebra of a suitable endofunctor. For any Lawvere theory $L$, $\mu X.T_L([\![\tau]\!] + F_\Sigma(X))$ is isomorphic to to $T_{(L+[\![\Sigma]\!])}$ [1]. That means that $\mathcal{T}([\![W_\Sigma(\Box\, \tau)]\!])$ is isomorphic to the presheaf that maps $L$ to $T_{(L+[\![\Sigma]\!])}([\![\tau]\!](\top))$, which in turn is isomorphic to $[\![y(\Sigma) \to \Box\, \tau]\!]$ by the result in the previous section. This isomorphism directly provides the semantics of reflection and reification.

# References

[1] Martin Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical computer science*, 357(1-3):70–99, 2006.

[2] Anton Lorenzen, Leo White, Stephen Dolan, Richard A Eisenberg, and Sam Lindley. Oxidizing OCaml with modal memory management. *Proceedings of the ACM on Programming Languages*, 8(ICFP):485–514, 2024.