Building Extensible Program Logics through Effect Handlers

Simon Oddershede Gregersen
joint work with Zichen Zhang and Joseph Tassarotti

Suppose we wanted to verify a program…

Programs logics are powerful! But what if we don't have one?

"Logic developer"

1. Model the new feature with an operational or denotational **semantics**.
2. Define a program logic by a collection of **reasoning rules**.
3. Prove that the rules from (2) are **sound** with respect to the semantics from (1).

… and then we **verify** the program!

"Program verifier"

Distinct skills!

… and difficult to reuse & combine.

Our approach

Effect handlers: a language feature to *program* custom effects in a modular way.

We use effect handlers to “bootstrap” program logics for different effects, e.g.,

- mutable state,
- shared-memory concurrency,
- distributed execution with unreliable networks, and
- crash-recovery with durable state.

and obtain rules analogous to **or stronger** than rules from specialized logics.

Our approach

1. Define a **core calculus** with effect handlers.
2. Develop a **program logic** for the core calculus.
3. Implement and verify effects/interpreters using effect handlers and the logic.
4. Use the derived “sub-logics” to verify effectful programs.

Effect handlers

try

$\dots$

raise $\mathrm{EFF}(v)$

$\dots$

with

$\mid \mathrm{EFF}(v), k \Rightarrow \dots;\ k\ w; \dots$

$\mid$ ret $v \quad \Rightarrow \dots$

end

Example: global memory cell

"Logic developer"

$$\mathsf{state} \triangleq \mathsf{rec}\ \mathsf{go}\ k\ r\ \sigma.$$
$$\mathsf{try}\ k\ r\ \mathsf{with}$$
$$\mid \mathtt{READ}(),\quad k \Rightarrow \mathsf{go}\ k\ \sigma\ \sigma$$
$$\mid \mathtt{WRITE}(v),\ k \Rightarrow \mathsf{go}\ k\ ()\ v$$
$$\mid x,\quad \_ \Rightarrow \mathsf{go}\ k\ (\mathsf{raise}\ x)\ \sigma$$
$$\mid \mathsf{ret}\ v \quad \Rightarrow v$$

$$\mathsf{run}_{\mathsf{state}} \triangleq \lambda \mathit{main}, \mathit{init}.\ \mathsf{state}\ \mathit{main}\ ()\ \mathit{init}$$

"Program verifier"

$$\mathit{main} \triangleq \lambda\ \_.$$
$$\mathsf{raise}\ \mathtt{WRITE}(41);$$
$$\mathsf{let}\ x = \mathsf{raise}\ \mathtt{READ}()\ \mathsf{in}$$
$$x + 1$$

Goal: verify programs as if effects were primitive

Core calculus

A **pure**, **sequential** lambda calculus with effect handlers:

The only primitive effect! (→ pick)

$$v ::= \cdots \mid \mathsf{cont}\ N$$

$$e ::= \cdots \mid \mathsf{raise}\ e \mid (\mathsf{try}\ e\ \mathsf{with}\ v\ k \Rightarrow e \mid \mathsf{ret}\ v \Rightarrow e) \mid \mathsf{pick}$$

$$K ::= \cdots \mid \mathsf{raise}\ K \mid (\mathsf{try}\ K\ \mathsf{with}\ v\ k \Rightarrow e \mid \mathsf{ret}\ v \Rightarrow e)$$

$$N ::= \cdots \mid \mathsf{raise}\ N$$

Operational semantics accordingly, e.g.,

$$\mathsf{try}\ (N[\mathsf{raise}\ w])\ \mathsf{with}\ v\ k \Rightarrow e_1 \mid \mathsf{ret}\ v \Rightarrow e_2 \quad \to \quad e_1[w/v][\mathsf{cont}\ N/k]$$

$$\mathsf{pick} \quad \to \quad n \in \mathbb{N}$$

Ficus

A sequential separation logic for the core effect-handler calculus, building on Hazel (de Vilhena & Pottier, 2021)—and Iris, of course.

protocol

$$\text{ewp } e \ \langle\Psi\rangle \ \{\Phi\}$$

postcondition

A “standard” partial correctness program logic, satisfying the rules you’d expect.

Protocols

value raised by the effect

"the continuation"

A protocol is a predicate $\Psi : \mathit{Val} \to (\mathit{Val} \to \mathit{iProp}) \to \mathit{iProp}$

Ficus satisfies

$$\frac{\Psi(v, \Phi)}{\text{ewp } \mathsf{raise}\ v\ \langle\Psi\rangle\ \{\Phi\}}$$

$$\frac{\text{ewp } e\ \langle\Psi\rangle\ \{\Phi\} \qquad \begin{array}{l}\big(\forall v_2.\ \Phi(v_2) \twoheadrightarrow \text{ewp } e_2\ \langle\Psi'\rangle\ \{\Phi'\}\big) \land \\ \big(\forall v_1, k_1.\ \Psi(v_1, \lambda w.\ \text{ewp } k_1\ w\ \langle\Psi\rangle\ \{\Phi\}) \twoheadrightarrow \text{ewp } e_1\ \langle\Psi'\rangle\ \{\Phi'\}\big)\end{array}}{\text{ewp } (\mathsf{try}\ e\ \mathsf{with}\ v_1\ k_1 \Rightarrow e_1 \mid \mathsf{ret}\ v_2 \Rightarrow e_2)\ \langle\Psi'\rangle\ \{\Phi'\}}$$

Example: global memory cell

$$\mathsf{READ}(v, \Phi) \triangleq \exists x.\, v = \mathtt{READ}() * S(x) * (S(x) \mathrel{-\!\!*} \Phi(x))$$

$$\mathsf{WRITE}(v, \Phi) \triangleq \exists x, y.\, v = \mathtt{WRITE}(y) * S(x) * (S(y) \mathrel{-\!\!*} \Phi())$$

$$\mathsf{STATE}(v, \Phi) \triangleq \mathsf{READ}(v, \Phi) \vee \mathsf{WRITE}(v, \Phi)$$

$$\frac{S(x)}{\text{ewp } \mathsf{raise}\ \mathtt{READ}()\ \langle \mathsf{STATE} \rangle\ \{v.\, v = x * S(x)\}}$$

$$\frac{S(x)}{\text{ewp } \mathsf{raise}\ \mathtt{WRITE}(y)\ \langle \mathsf{STATE} \rangle\ \{v.\, v = () * S(y)\}}$$

$$\frac{S(\mathit{init}) \mathrel{-\!\!*} \text{ewp } \mathit{main}\ ()\langle \mathsf{STATE} \rangle\{\Phi\}}{\text{ewp } \mathit{run}_{\mathsf{state}}\ \mathit{main}\ \mathit{init}\langle \bot \rangle\{\Phi\}}$$

Protocols cont'd

In practice, we want to nest and compose handlers. To this end, define

$$(\Psi_1 \oplus \Psi_2)(v, \Phi) \triangleq \Psi_1(v, \Phi) \vee \Psi_2(v, \Phi)$$

and thus

$$\Psi_1 \sqsubseteq \Psi_1 \oplus \Psi_2$$

We generalize all rules accordingly, e.g.,

$$\frac{S(x) \qquad \text{STATE} \sqsubseteq \Psi}{\text{ewp } \text{raise } \texttt{READ}()\ \langle \Psi \rangle \{v.\, v = x * S(x)\}}$$

Example: heap

$$
\begin{aligned}
\text{heap} \triangleq \ & \textsf{rec}\ \text{go}\ k. \\
& \quad \textsf{try}\ k\ ()\ \textsf{with} \\
& \quad \mid \texttt{ALLOC}(), \quad k \Rightarrow \ldots \\
& \quad \mid \texttt{LOAD}(\ell, v), \ k \Rightarrow \ldots\ \textsf{raise}\ \texttt{READ}()\ \ldots \\
& \quad \mid \texttt{STORE}(\ell, v), k \Rightarrow \ldots\ \textsf{raise}\ \texttt{WRITE}(h[\ell \mapsto v])\ \ldots \\
& \quad \mid x, \quad \_ \Rightarrow \ldots\ \textsf{raise}\ x\ \ldots \\
& \quad \mid \textsf{ret}\ v \quad \Rightarrow v
\end{aligned}
$$

Concurrency

nondeterministic choice

$$
\begin{aligned}
\text{conc} \triangleq\ & \textbf{rec}\ \text{go pool}. \\
& \quad \textbf{let}\ ((k, r, t), \text{pool}) = \text{choose pool}\ \textbf{in} \\
& \quad \textbf{try}\ k\ r\ \textbf{with} \\
& \quad \mid \mathtt{FORK}(e), k \Rightarrow \text{go}\ (\text{pool} \uplus (e, (), \mathcal{C}) \uplus (k, (), t)) \\
& \quad \mid x, \quad \_ \Rightarrow \text{go}\ (\text{pool} \uplus (k, \textbf{raise}\ x, t)) \\
& \quad \mid \textbf{ret}\ v \quad \Rightarrow \textbf{if}\ t = \mathcal{M}^{\times}\ \textbf{then}\ v \\
& \quad \qquad \textbf{else}\ \text{go}\ (\text{pool} \uplus (\lambda\_.\ v, (), t^{\times})) \\
\text{run}_{\text{conc}} \triangleq\ & \lambda \mathit{main}.\ \text{go}\ \{(\mathit{main}, (), \mathcal{M})\}
\end{aligned}
$$

Challenges

Iris/CSL invariants rely on **atomicity**. Our handler is not atomic...

$$\frac{\boxed{P}^{\mathcal{N}} \qquad \mathcal{N} \subseteq \mathcal{E} \qquad \triangleright P \twoheadrightarrow \mathsf{wp}_{\mathcal{E} \setminus \mathcal{N}}\ e\ \{v.\, \triangleright P * \Phi(v)\} \qquad \boxed{\mathsf{atomic}(e)}}{\mathsf{wp}_{\mathcal{E}}\ e\ \{\Phi\}}$$

While expressive, e.g., Perennial and Nola also consider alternative forms.

Instead, we want to allow handler implementers to define **custom notions** of invariant suitable for the kind of effect they are modeling.

Iris invariants

$$\frac{\boxed{P}^{\mathcal{N}} \qquad \mathcal{N} \subseteq \mathcal{E}}{{}_{\mathcal{E}}{\Rrightarrow}_{\mathcal{E}\setminus\mathcal{N}} \triangleright P * (\triangleright P \twoheadrightarrow {}_{\mathcal{E}\setminus\mathcal{N}}{\Rrightarrow}_{\mathcal{E}} \mathsf{True})}$$

$$\frac{{}_{\mathcal{E}_1}{\Rrightarrow}_{\mathcal{E}_2} \mathsf{wp}_{\mathcal{E}_2}\ e\ \{v.\ {}_{\mathcal{E}_2}{\Rrightarrow}_{\mathcal{E}_1} \Phi(v)\} \qquad \mathsf{atomic}(e)}{\mathsf{wp}_{\mathcal{E}_1}\ e\ \{\Phi\}}$$

The fancy update uses a mechanism called "**world satisfaction**" and tracks the set of all enabled/disabled invariants

$${}_{\mathcal{E}_1}{\Rrightarrow}_{\mathcal{E}_2} P \triangleq \mathsf{wsat} * \mathsf{Tok}(\mathcal{E}_1) \twoheadrightarrow {\Rrightarrow} (\mathsf{wsat} * \mathsf{Tok}(\mathcal{E}_2) * P)$$

Extensible worlds

To achieve the kind of extensibility we're looking for, Ficus does not bake in a single world.

$$\mathsf{ewp}_{\mathsf{W}_1,\mathsf{W}_2}\ e\ \{\Phi\} \qquad\qquad {}_{\mathsf{W}_1}\!\models\!\!\!\Rightarrow_{\mathsf{W}_2} P$$

Worlds are just Iris assertions, but helpful to think of them more abstractly:

$$\mathsf{W}_1 \oplus \mathsf{W}_2 \triangleq \mathsf{W}_1 * \mathsf{W}_2$$
$$\mathsf{W}_1 \sqsubseteq \mathsf{W}_2 \triangleq \exists \mathsf{W}'.\, \mathsf{W}_2 \dashv\vdash \mathsf{W}_1 \oplus \mathsf{W}'$$

Extensible worlds cont'd

$$\frac{\mathsf{W}_1 \mathrel{-\!\!*} \Rrightarrow (P * \mathsf{W}_2)}{{}_{\mathsf{W}_1}\!\Rrightarrow_{\mathsf{W}_2} P}
\qquad
\frac{Q \vdash {}_{\mathsf{W}_2}\!\Rrightarrow_{\mathsf{W}_3} P}{({}_{\mathsf{W}_1}\!\Rrightarrow_{\mathsf{W}_2} Q) \vdash {}_{\mathsf{W}_1}\!\Rrightarrow_{\mathsf{W}_3} P}
\qquad
\frac{{}_{\mathsf{W}_1}\!\Rrightarrow_{\mathsf{W}_2} P}{{}_{\mathsf{W}_1 \oplus \mathsf{W}}\!\Rrightarrow_{\mathsf{W}_2 \oplus \mathsf{W}} P}$$

$$\frac{{}_{\mathsf{W}_1}\!\Rrightarrow_{\mathsf{W}_2} \mathsf{ewp}_{\mathsf{W}_2,\mathsf{W}_3}\ e\ \langle \Psi \rangle\ \{\Phi\}}{\mathsf{ewp}_{\mathsf{W}_1,\mathsf{W}_3}\ e\ \langle \Psi \rangle\ \{\Phi\}}
\qquad
\frac{{}_{\mathsf{W}_1}\!\Rrightarrow_{\bot} \Psi(v, (\lambda r.\ {}_{\bot}\!\Rrightarrow_{\mathsf{W}_2} \Phi(r)))}{\mathsf{ewp}_{\mathsf{W}_1,\mathsf{W}_2}\ \mathsf{raise}\ v\ \langle \Psi \rangle\ \{\Phi\}}$$

No atomicity requirement!

Recovering Iris invariants

$$\frac{\boxed{P}^{\mathcal{N}} \qquad \mathcal{N} \subseteq \mathcal{E}}{{}_{\mathsf{Tok}(\mathcal{E})}{\Rrightarrow}_{\mathsf{Tok}(\mathcal{E}\setminus\mathcal{N})} \triangleright P * (\triangleright P \mathrel{-\!\!*} {}_{\mathsf{Tok}(\mathcal{E}\setminus\mathcal{N})}{\Rrightarrow}_{\mathsf{Tok}(\mathcal{E})} \mathsf{True})}$$

… but by framing we can also include other components in the world.

Concurrent protocols

The concurrency handler is context generic—it simply re-raises effects and potentially control to outer handlers (shared memory, message-passing, ...).

We introduce a **protocol transformer** that lifts a protocol for outer effects into a concurrent protocol.

$$\mathsf{ATOM}_{\mathsf{W}}(\Psi)(v, \Phi) \triangleq \Psi(v, (\lambda r.\ {}_{\bot}{\Rrightarrow}_{\mathsf{W}}\ {}_{\mathsf{W}}{\Rrightarrow}_{\bot}\ \Phi(r)))$$

If we pick $\mathsf{W} \triangleq \mathsf{Tok}(\top)$ we recover Iris invariants.

Concurrency cont'd

$$\mathsf{CONC}_{\mathsf{W}}(\Psi) \triangleq \mathsf{ATOM}_{\mathsf{W}}(\Psi \oplus \mathsf{FORK}_{\mathsf{W}}(\Phi))$$

$$\mathsf{FORK}_{\mathsf{W}}(\Psi)(v, \Phi) \triangleq \exists e.\, v = \mathtt{FORK}(e) * \triangleright \mathsf{ewp}_{\mathsf{W}}\ e\ \langle \mathsf{CONC}_{\mathsf{W}}(\Psi) \rangle\ \{\_.\ \mathsf{True}\} * \Phi\ ()$$

$$\frac{\mathsf{ewp}_{\mathsf{W}}\ e\ \langle \mathsf{CONC}_{\mathsf{W}}(\Psi) \rangle\ \{\Phi\}}{\mathsf{ewp}_{\mathsf{W}}\ \mathsf{raise}\ \mathtt{FORK}(e)\ \langle \mathsf{CONC}_{\mathsf{W}}(\Psi) \rangle\ \{v.\, v = ()\}}$$

$$\frac{\mathtt{fork} \notin \mathsf{tags}(\Psi) \qquad \mathsf{ewp}_{\mathsf{W}}\ \mathit{main}\ ()\ \langle \mathsf{CONC}_{\mathsf{W}}(\Psi) \rangle\ \{\Phi\}}{\mathsf{ewp}_{\mathsf{W}}\ \mathsf{run}_{\mathsf{conc}}\ \mathit{main}\ \langle \Psi \rangle\ \{\Phi\}}$$

Is this sound?

You may rightfully object that a standard operational semantics generates more interleavings of thread operations than our handler-based semantics.

Intuitively, **thread-local pure steps are not observable to other threads**.

t1: $e_1 \rightarrow e_2 \rightarrow \texttt{raise}\ \texttt{EFF}(v) \qquad e_3 \rightarrow \cdots$

t2: $e'_1 \rightarrow e'_2 \rightarrow \texttt{raise}\ \texttt{EFF'}(v')$

More precisely...

$$e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow \cdots$$

$$\wr\wr$$

$$e_1 \rightarrow \text{yield} \rightarrow e_2 \rightarrow \text{yield} \rightarrow e_3 \rightarrow \cdots$$

Which boils down to showing $\text{yield} \simeq_{\text{ctx}} ()$

Banyan: A relational logic

Following the CaReSL/Iris approach, we build a relational logic on top of our unary logic using an **effect specification resource**

$$\text{espec}_{\text{W}}\ e\ \langle \Psi \rangle$$

… which can be updated and progressed similarly to the weakest precondition.

$$\text{espec}_{\text{W}}\ K[e]\ \langle \Psi \rangle * e \to^* e' \vdash \Rrightarrow_{\text{W}} \text{espec}_{\text{W}}\ K[e']\ \langle \Psi \rangle$$

$$\text{espec}_{\text{W}}\ N[\text{raise}\ v]\ \langle \Psi \rangle * \Psi(v, \Phi) \vdash \Rrightarrow_{\text{W}} \exists w.\, \text{espec}_{\text{W}}\ N[w]\ \langle \Psi \rangle * \Phi(w)$$

What about concurrency?

Like in the unary case, we need to make use of extensible worlds.

However, we also want to **reason about threads independently**. In Iris, this is achieved by having a per-thread specification resource and context

$i \Rrightarrow e$ specCtx

In our setting, threads may both share effects and have local effects!

Effect specification resource

Given **any** (abstract) predicate spec : Expr → iProp such that

$$\mathsf{spec}(e) \vdash \Rrightarrow_{\mathsf{W}} \mathsf{spec}(e') \qquad \text{if } e \to^* e'$$

Define

$$\mathsf{genspec}_{\mathsf{W}}\ e\ \langle \Psi \rangle \triangleq \exists K.\, \mathsf{spec}(K[e]) * \mathsf{handler}_{\mathsf{W}}(\Psi)(K)$$

capturing that e runs in a handler context satisfying Ψ.

Pick, for example, $\mathsf{spec}(e) \triangleq e_0 \to^* e$.

Effect specification resource

$$
\begin{aligned}
\mathsf{handler}_{\mathsf{W}}(\Psi) \triangleq \mathsf{gfp}\ F, K.\ & \\
& \forall v.\ \mathsf{spec}(K[v]) \twoheadrightarrow\ \Rrightarrow_{\mathsf{W}} \mathsf{spec}(v) \\
\wedge\ & \forall v, N, \Phi.\ \mathsf{spec}(K[N[\mathsf{raise}\ v]]) * \Psi(v, \Phi) \twoheadrightarrow\ \Rrightarrow_{\mathsf{W}} \\
& \qquad \exists K', w.\ \mathsf{spec}(K'[N[w]]) * \Phi(w) * F(K')
\end{aligned}
$$

"termination case"

"effect case"

Per-thread effect spec resource

per-thread spec resource **a "base" spec resource**

$$\text{spec}_t(e) \triangleq \exists k, r.\ \boxed{\circ\{(k, r, t)\}} * (\forall K.\ \text{spec}(K[k\ r]) \twoheadrightarrow {\Rrightarrow}\text{spec}(K[e]))$$

$$\text{CTX}(\text{W}, \Psi) \triangleq \exists B, pool.\ \text{isBag}(B, pool) * \boxed{\bullet B} * \text{espec}_{\text{W}}\ \text{conc}\ pool\ \langle \Psi \rangle$$

spec context

Since spec_t satisfies the requirements of being a spec, we obtain for free

$$\text{espec}^t_{\text{W}}\ e\ \langle \Psi \rangle$$

which gives a unified mechanism for talking about both local and global effects!

Per-thread effect spec resource

$$\frac{\mathsf{espec}^{t}_{\mathsf{W}}\ N[\mathsf{raise}\ \mathtt{FORK}(e)]\ \langle\Psi\rangle \qquad \mathsf{CTX}(\mathsf{W}',\Psi') \sqsubseteq \mathsf{W} \qquad \mathsf{FORK_s} \sqsubseteq \Psi}{\Rrightarrow_{\mathsf{W}} \mathsf{espec}^{t}_{\mathsf{W}} N[()]\ \langle\Psi\rangle * \mathsf{espec}^{t}_{\mathsf{CTX}(\mathsf{W}',\Psi')}\ e\ \langle \mathsf{FORK_s} \oplus \Psi' \rangle}$$

where $\mathsf{FORK_s}(v,\Phi) \triangleq \exists e.\, v = \mathtt{FORK}(e) * (\mathsf{spec}_{\mathcal{C}}(e) \mathrel{-\!\!*} \Phi\ ())$

$$\frac{\mathsf{espec}_{\mathsf{W}}\ \mathsf{run_{conc}}\ \mathit{main}\ \langle\Psi\rangle \qquad \mathtt{fork} \notin \mathsf{tags}(\Psi)}{\Rrightarrow_{\mathsf{W}} \exists\gamma.\, \mathsf{CTX}^{\gamma}(\mathsf{W},\Psi) * \mathsf{espec}^{\gamma;\mathcal{M}}_{\mathsf{CTX}^{\gamma}(\mathsf{W},\Psi)}\ \mathit{main}\ ()\ \langle \mathsf{FORK_s} \oplus \Psi \rangle}$$

Contextual refinement

Using Banyan, we define a completely **standard binary logical relation** for a classical System-F-like type system to show contextual refinement:

$$e_1 \precsim^{H}_{\text{ctx}} e_2 : \tau \quad \triangleq \quad \forall b, C.\, H[C[e_1]] \rightarrow^* b \quad \Rightarrow \quad H[C[e_2]] \rightarrow^* b$$

For example, for

$$H_{\text{HeapLang}} \triangleq \text{run}_{\text{state}}\ (\lambda\_.\ \text{run}_{\text{heap}}\ (\lambda\_.\ \text{run}_{\text{atomheap}}\ (\lambda\_.\ \text{run}_{\text{conc}}\ [])))$$

we show

$$\text{yield} \simeq^{H_{\text{HeapLang}}}_{\text{ctx}} ()$$

Other case studies

- **Prophecy variables:** local from global variable, implicit prophecies
- **Crash-recovery:** crash invariants & borrows, crash-aware prophecies
- **Distributed execution** with unreliable communication, Ironfleet-style atomicity

That’s it, folks!