Programming Coroutines via Effect Handlers

Or How Effect Handlers are Structured Coroutines

Daniel Hillerström

Computing Systems Laboratory
Zurich Research Center
Huawei Technologies, Switzerland

June 27, 2024

Barcelona Supercomputing Center and Huawei
3rd Workshop on Nanos, FunctionFlow, and HiCR
Shanghai, China

`https://dhil.net`

Coroutines are everywhere

OCaml JS ERLANG . . .

Powering programming idioms

- Async/await (e.g. C++, C#, Dart, JavaScript, Rust, Swift)
- Lightweight threads (e.g. Erlang, Go, Haskell, Java, Swift)
- Yield-style generators (e.g. C#, Dart, Haskell, JavaScript, Kotlin, Python)

Powering programming models

- User interface programming (e.g. widgets)
- High performance programming (e.g. tasking)
- Probabilistic programming (e.g. sampling)

Coroutines are everywhere

OCaml JS . . .

Powering programming idioms

- Async/await (e.g. C++, C#, Dart, JavaScript, Rust, Swift)
- Lightweight threads (e.g. Erlang, Go, Haskell, Java, Swift)
- Yield-style generators (e.g. C#, Dart, Haskell, JavaScript, Kotlin, Python)

Powering programming models

- User interface programming (e.g. widgets)
- High performance programming (e.g. tasking)
- Probabilistic programming (e.g. sampling)

Coroutines are an instance of **first-class continuations**

Coroutines are all great, what's the problem?

Classical coroutines do not offer **modular composition**

Problem: one type to embed them all

```
R suspend<R,S>(S)

union<A,S> resume<R,S,A>(coroutine_t<R,S,A>, R)
```

Coroutines are all great, what’s the problem?

Classical coroutines do not offer **modular composition**

Problem: one type to embed them all

```
R suspend<R,S>(S)

union<A,S> resume<R,S,A>(coroutine_t<R,S,A>, R)
```

Programming over a fixed type! Or worse no types

Coroutines are all great, what's the problem?

Classical coroutines do not offer **modular composition**

Problem: one type to embed them all

```
R suspend<R,S>(S)

union<A,S> resume<R,S,A>(coroutine_t<R,S,A>, R)
```

Programming over a fixed type! Or worse no types

This is known as the "universal control effect"

Coroutines are all great, what's the problem?

Classical coroutines do not offer **modular composition**

Problem: one type to embed them all

```
R suspend<R,S>(S)

union<A,S> resume<R,S,A>(coroutine_t<R,S,A>, R)
```

Programming over a fixed type! Or worse no types

This is known as the "universal control effect"

Solution: name control effects

```
effect eff :  S -> R
```

R **suspend**($\texttt{eff}_{S,R}$)

union<A,$\texttt{eff}_{S,R}$...> **resume**[$\texttt{eff}_{S,R}$ ...]<A>(coroutine_t<A>, R)

Coroutines are all great, what's the problem?

Classical coroutines do not offer **modular composition**

Problem: one type to embed them all

```
R suspend<R,S>(S)

union<A,S> resume<R,S,A>(coroutine_t<R,S,A>, R)
```

Programming over a fixed type! Or worse no types

This is known as the "universal control effect"

Solution: name control effects

effect eff : S -> R

R **suspend**($eff_{S,R}$)

union<A,$eff_{S,R}$...> **resume**[$eff_{S,R}$ ...]<A>(coroutine_t<A>, R)

Now we have discovered effect handlers

Demos

Demo programs in **libseff** (Alvarez-Picallo et al. 2023)

Warm-up: Hello World

`src/hello.c`

Dynamic binding

`inc/env.h`

`src/env.c`

Lightweight threading

`inc/lwt.h`

`src/lwt.c`

Obtaining actors via modular composition

`src/actor.c`

Conclusions

Summary

- Effect handlers allow programmers to name control effects
- Differentiating control effects enables modular composition
- Customisable and flexible interpretation of effects

Future considerations

- A HiCR frontend for effect handler oriented programming (EHOP)?
- FunctionFlow as the universal runtime? A bespoke API for EHOP
- Abstracting coroutine/continuation/stack allocation policies

References

Plotkin, Gordon D. and Matija Pretnar (2013). “Handling Algebraic Effects”. In: *Logical Methods in Computer Science* 9.4. DOI: 10.2168/LMCS-9(4:23)2013.

Alvarez-Picallo, Mario et al. (Nov. 2023). *High-level effect handlers in C.* https://homepages.inf.ed.ac.uk/slindley/papers/libseff-draft-november2023.pdf.