Concurrent System Programming with Effect Handlers

Stephen Dolan[1]
Spiros Eliopoulos[3]
Daniel Hillerström[2]
Anil Madhavapeddy[1]
KC Sivaramakrishnan[1]
Leo White[3]

[1] University of Cambridge
[2] The University of Edinburgh
[3] Jane Street Capital

Effect handlers

```
let do_stuff x =
  if x.is_bad then
    log “oh no”;
  process x
```

Effect handlers

```
let do_stuff x =
  if x.is_bad then
    raise PrettyBad;
  process x
```

```
match ... with
| result -> result
| exception PrettyBad ->
  log “it's pretty bad”;
  exit 1
```

Effect handlers

```
let do_stuff x =
  if x.is_bad then
    raise PrettyBad;
  process x
```

```
match ... with
| result -> result
| exception PrettyBad ->
 (* wish it kept going *)
```

Effect handlers

```
let do_stuff x =
  if x.is_bad then
    raise PrettyBad;
  process x
```

```
match ... with
| result -> result
| exception PrettyBad ->
  continue
```

Effect handlers

```
let do_stuff x =
  if x.is_bad then
    perform PrettyBad;
  process x
```

```
match ... with
| result -> result
| effect PrettyBad k ->
  continue k
```

Scheduling tasks

```
let run_q =
  Queue.create ()

let enqueue k =
  Queue.push k run_q

let rec dequeue () =
  if Queue.is_empty run_q then ()
  else continue (Queue.pop run_q) ()
```

Scheduling tasks

```
effect Yield : unit
effect Fork : (unit -> unit) -> unit

let rec schedule f =
  match f () with
  | () -> dequeue ()
  | effect Yield k ->
      enqueue k; dequeue ()
  | effect (Fork f) k ->
      enqueue k; schedule f
```

Callbacks

Callback-style iteration is easy to implement:

```
val iter :
  (int -> unit) -> tree -> unit

let rec iter f t =
  match t with
  | Leaf n -> f n
  | Branch (a, b) ->
    iter f a; iter f b
```

Callbacks

With explicit iterators, we can zip:

```
type 'a iterator =
  unit -> 'a option

let rec iterBoth f xs ys =
  match xs (), ys () with
  | Some x, Some y ->
    f x y; iterBoth f xs ys
  | _ -> ()
```

Avoiding callbacks

Effects let you turn this:

```
val iter :
  (int -> unit) -> tree -> unit
```

into this:

```
type iterator =
  unit -> int option
```

without turning half the code inside out

Avoiding callbacks

```
effect Next : int -> unit
let to_gen t =
  let step = ref (fun () -> assert false) in
  let first_step () =
    match
      iter (fun x -> perform (Next x)) t
    with
    | () -> None
    | effect (Next v) k ->
      step := continue k;
      Some v in
  step := first_step;
  fun () -> !step ()
```

Direct-style I/O

```
let handle conn =
  let request = read conn in
  write conn (respond_to request)
```

Callback-based I/O

```
let handle conn =
  let ongoing = async_read conn in
  when_completed ongoing (fun req ->
    async_write conn (respond_to req))
```

Direct-style I/O

- Simple I/O interfaces use direct calls
- Efficient ones use callbacks, for overlapping
- With effects, we can write the simple code but run the fast code
  - Just as fast as callbacks!

Managing resources

```
let file = open_in “words.txt” in
match parse_contents file with
| result ->
  close file;
  result
| exception e ->
  close file;
  raise e
```

Managing resources

- Computations holding resources are linear
  - so their continuations are too!
- Linear continuations are very, very fast
- Rather than types, we fake linearity dynamically

Interrupts

- Cancelling an ongoing computation is hard

- Should we cancel
  - synchronously, by polling?
  - asynchronously, by interrupting?

- Pure code should be cancelled asynchronously
  - but transitions are hard

Ctrl-C as a callback

- Callbacks
  - Communicate with program only via globals.
- Higher order mutable state
  - Signal runs the “current signal handler”
- Resource handling
  - Don't even know which ones!

Ctrl-C as an asynchronous effect

- Nestable match statements
  - use local variables in the interruptible code
  - turn on and off signal handling, without getting scoping wrong
  - disable signals for resource handling

Summary

- Effect handlers are a great new tool!
- They work really well for system programming
  - as long as we use the linear version
- They make nasty OS interfaces easier to use
  - and save us from callback hell