# Implementing UNIX with Effects Handlers

Ramsay Carslaw

October 16, 2023

# Contents

**1 Introduction** **2**

**2 Background** **3**
2.1 Algebraic Effects and Effect Handlers . . . . . 3
2.1.1 Example: Heads and Tails . . . . . 3
2.2 Affine and 'Multi-Shot' Handlers . . . . . 5
2.3 The State of Effect-Oriented Programming . . . . . 5
2.3.1 Library Based Effects . . . . . 5
2.3.2 First-Class Effects . . . . . 5
2.4 UNIX . . . . . 5
2.5 Effect Handlers and UNIX . . . . . 5

**3 Methods** **6**

**4 Results** **7**

**5 Conclusion** **8**

# Chapter 1

# Introduction

# Chapter 2

# Background

## 2.1 Algebraic Effects and Effect Handlers

Algebraic effects [1] and their corresponding handlers [2] [3] are a programming paradigm that when paired together offers a novel way to compose programs. It starts with the definition of the effect or the *effect signature* that gives the effect a name in scope and specifies any input and the return type otherwise known as the *effect operation*. At this stage the effect operation has no implementation and is more an acknowledgement to the compiler that it should expect an implementation. For this reason any function that references these effect operations is known as an *effectful function* or a function whose definition is not complete without an effect handler. The *effect handler* provides one implementation of the given effect operation. In this way we can change the semantics of an effectful function by handling it with a different handler that provides an alternative implementation to the effect.

When programs are written that are 'black boxes', that is to say their outputs are defined entirely by their inputs and all functions are pure computation [4], it is safe to make assumptions about the inputs. Assumptions like an age will always be given as an integer or all strings will not exceed the length allocated for them. When programs interact with the real world it is no longer safe to make these assumptions. Effects allow the programmer to encapsulate these side effects and 'handle' them with control flow.

### 2.1.1 Example: Heads and Tails

Consider the following example of heads or tails written conventionally.

```
function choose() -> bool {
  if (rand() > 0.5) {
    return true
  }
  return false
}

```

```
function flip() -> string {
  if (choose() == true) {
    return 'Heads'
  } else {
    return 'Tails'
  }
}
```

The function choose returns true or false randomly, with equal probability and flip uses the result of chose to print heads on true and tails on false. This works fine but if we wanted multiple definitions of choose where we alter the probability or add an option for failure we would have to rewrite each function for each implementation.

We can use effects handlers as described by Pretnar et al [3] to provide a generic implementation of choose that can be handled by flip. The following is an example of choose implemented with effects.

```
effect Choose {
  choose : () -> bool
}

function choose() -> bool {
  invoke choose()
}

function flip() -> string {
  if (choose() == true) {
    return 'Heads'
  } else {
    return 'Tails'
  }
}
```

The effect Choose is defined with a single operation choose that takes no arguments and returns a boolean. The function choose is defined as a wrapper that simply invokes the choose effect. The function flip is unchanged from the previous example. We can now define a *handler* that provides the implementation details for choose. To get the same result as before we could do the following.

```
function fairToss(value, resume) {
  if (rand() > 0.5) {
    resume(true)
  }
  resume(false)
}

handle flip() with fairToss
```

Here we split the effect into the value and the remaining computation, *resume*. Depending on the rand function we continue the computation with either true or false as the value for choose. The true power of effects comes when we define multiple handlers for one effect for example we can make the coin weighted:

```
function unfairToss(value, resume) {
  if (rand() > 0.75) {
    resume(true)
```

```
  }
  resume(false)
}

handle flip() with unfairToss
```

Or even get heads and tails at once by returning the result of both true and false in a list.

```
function both(value, resume) {
  resume(true) ++ resume(false)
}

handle flip() with both
```

We get all of these without changing either of our original definitions. This is the expressive power of algebraic effect handlers.

## 2.2 Affine and 'Multi-Shot' Handlers

## 2.3 The State of Effect-Oriented Programming

### 2.3.1 Library Based Effects

### 2.3.2 First-Class Effects

## 2.4 UNIX

Unix [5] is an operating system designed and implemented by Dennis M. Ritchie and Ken Thompson at AT&T's Bell Labs in 1974. It provides a file system (directories, file protection etc.), a shell, processes (pipe, fork etc) and a userspace. Since it's first release it has been reimplemented for a variety of systems.

## 2.5 Effect Handlers and UNIX

In chapter 2 of his 2022 thesis, Daniel Hillerström [6] outlines a theoretical implementation of UNIX using the effects syntax outlined by Kammar et. al. [7]. In this he provides an implementation of the original Unix paper [5] that includes a filesystem and timesharing. Hillerström makes several assumptions about the effect system that would need to be taken into account in order to implement this with a real language. The main assumption is multi-shot handlers. For example the implementation of `fork` uses multi-shot handlers to copy the full stack on both branches.

# Chapter 3

# Methods

# Chapter 4

# Results

# Chapter 5

# Conclusion

# Bibliography


[1] Gordon Plotkin and John Power. Computational effects and operations: An overview. 2002.

[2] Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In *European Symposium on Programming*, pages 80–94. Springer, 2009.

[3] Matija Pretnar. An introduction to algebraic effects and handlers. invited tutorial paper. *Electronic notes in theoretical computer science*, 319:19–35, 2015.

[4] John Hughes. Why functional programming matters. *The computer journal*, 32(2):98–107, 1989.

[5] Dennis M Ritchie and Ken Thompson. The unix time-sharing system. *Bell System Technical Journal*, 57(6):1905–1929, 1978.

[6] Daniel Hillerström. Foundations for programming and implementing effect handlers. 2022.

[7] Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in action. *ACM SIGPLAN Notices*, 48(9):145–158, 2013.