A Survey of Algebraic Effect System

Ryan J. Kung
ryankung@ieee.org

July 21, 2018

1 Introduction

Algebraic eects are an approach to computational eects based on a premise that impure behaviour arises from a set of operations such as get and set for mutable store, read and print for interactive input and output, or raise for exceptions. This naturally gives rise to handlers not only of exceptions, but of any other eect, yielding a novel concept that, amongst others, can capture stream redirection, backtracking, co-operative multi-threading, and delimited continuations [10].

1.1 Syntax and Terms

value v	$::=$	x	variable
		$\mid$ **true** $\mid$ **false**	boolean constants
		$\mid$ **fun** $x \mapsto c$	function
		$\mid$ h	handler
handler h	$::=$	**handler** $\{$**return** $x \mapsto c_r$,	(optional) return clause
		$\mathsf{op}_1(x; k) \mapsto c_1, \ldots, \mathsf{op}_n(x; k) \mapsto c_n\}$	operation clauses
computation c	$::=$	**return** v	return
		$\mid$ $\mathsf{op}(v; y.\, c)$	operation call
		$\mid$ **do** $x \leftarrow c_1$ **in** c_2	sequencing
		$\mid$ **if** v **then** c_1 **else** c_2	conditional
		$\mid$ $v_1\, v_2$	application
		$\mid$ **with** v **handle** c	handling

Figure 1: Syntax of Terms [10]

Syntax *do* is quiet similar to the *do* of Monad which is present for Sequencing, and the operation calls, it can be considered as a lazy functional call like $v.c := op(x)$. The intro paper [10] give out definition of Generic effect as:

$$op \stackrel{def}{=\!=} fun\ x \to op(x; y.return\ y) \tag{1}$$

1.2 Basic Example

Consider a simple handler is:

$$getUserName \stackrel{def}{=\!=} fun\ x \to readline(\_; k) \to k.promot\ s) \tag{2}$$

And this simple input & output example can be implemented with Python and Effect Library [2]:

```
class ReadLine(object):
    def __init__(self, prompt):
        self.prompt = prompt

def get_user_name():
    return Effect(ReadLine("Enter a candy> "))

@sync_performer
def perform_read_line(dispatcher, readline):
    return raw_input(readline.prompt)

def main():
    effect = get_user_name()
    effect = effect.on(
        success=lambda r: print("I like {} too!".format(r)),
        error=lambda e: print("sorry. {}".format(e)))

        dispatcher = TypeDispatcher(
            {ReadLine: perform_read_line}
        )
    sync_perform(dispatcher, effect)
```

2 Theory

2.1 Algebraic Operation

Operations of algebraic Effect was first introduced by Gordon Poltkin and John Power in 2002 [8] as Algebraic Operation, based on Eugenio Moggi's work [6, 7] in 1989-1992 about logics for reasoning and proving equivalence about programs with a strong monad T on a base category C with finite products [7, 8]. And made the notion system for effects such as: nondeterminism, probabilistic nondeterminism, exceptions, interactive input/output, side-effects, and continuations by identifying it with the notion of *algebraic* operation.

> "Algebraic operations are, in the sense we shall make precise, a natural generalisation, from Set to an arbitrary symmetric monoidal $V - category\ C$ with cotensors." [8]

2.2 Algebraic Handlers

Effect and handeler was introduced by Plotkin and Pretnar in 2009 [9] as computaional effects that can be represented by an equational theory whose oerations produce the effect as hand, which are based on Moggi's monad work, but as a restriction on general monads, algebraic effects have many various advantages: can be freely composed, and there is a natual separation between interfaces and sematics (as handler) [5]

3 Implementations

3.1 Eff

Eff is a programming language based on the algebraic approach to computational effects, in which effects are viewed as algebraic operations and effect handlers as homomorphisms from free algebras. [3] Eff supports rst-class effects and handlers through which we may easily dene new computational effects, seamlessly combine existing ones, and handle them in novel ways.

3.2 Koka

Koka is a function-oriented programming language that seperates pure values from side-effecting computations, where the effect of every function is automatically inferred.

3.3 Other Platforms

Algebraic Effect usually related to Eff which support Algebraic Effects and Handlers as first class, Algebraic Effect can also be widely use in common platform such as ECMAScript, .net, JVM, or other programming languages by using a type directed selective CPS translation [5].

Since Algebraic Effect and Its Operators are build based on a Monad System on category-V, it can be used for both Strong type languages line Haskell or OCaml and weak type languages like Python or ECMAScript, but there is some challenge that, a Typing algebraic effects is that inferred types became very large or difficult to understand. And for a library based implementation, it do not have full control over the runtime stack. [5].

- Idris is a general purpose pure functional programming language with dependent types. From version 0.9.12 Idris includes a library for side-effect management, Effects.
- Python Effect is a library of Python for helping write purely functional code by isolating the effects [2].
- Haskell Effect-handlers is a library for writing extensible algebraic effects and handlers with haskell [1].

3.4 Other Extensions

Niki Vazou and Daan LeiJen introduced how to combine Algebraic Effect with Monadic System [11]. In Leijen and Vazou's work, they combined *monads* and $effecttyping$ by using monad for defining the semantics of effect types and then using algebraic effect types to program with those monads. The idea was implemented as an extension of $Kola$.

Martin Hyland, Paul Blain Levy, Gordon Plotkin, and John Power has introduced an idea of combining Algebraic Effect with continuations [4].

References

[1] Blaz Repas Andraz Bajt. effect-handlers: A library for writing extensible algebraic effects and handlers. similar to extensible-effects but with deep handlers. `http://hackage.haskell.org/package/effect-handlers`.

[2] Christopher Armstrong. Effect. `https://github.com/python-effect/effect`, 2014.

[3] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *CoRR*, abs/1203.1539, 2012.

[4] Martin Hyland, Paul Blain Levy, Gordon Plotkin, and John Power. Combining algebraic effects with continuations. *Theoretical Computer Science*, 375(1):20 – 40, 2007. Festschrift for John C. Reynoldss 70th birthday.

[5] Daan Leijen. Algebraic effects for functional programming. Technical report, August 2016.

[6] E. Moggi. Computational lambda-calculus and monads. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, Jun 1989.

[7] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55 – 92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[8] Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, Feb 2003.

[9] Gordon D Plotkin and Matija Pretnar. Handling Algebraic Effects. *Logical Methods in Computer Science*, Volume 9, Issue 4, December 2013.

[10] Matija Pretnar. An introduction to algebraic effects and handlers. invited tutorial paper. *Electron. Notes Theor. Comput. Sci.*, 319(C):19–35, December 2015.

[11] Niki Vazou and Daan Leijen. From monads to effects and back. In Marco Gavanelli and John Reppy, editors, *Practical Aspects of Declarative Languages*, pages 169–186, Cham, 2016. Springer International Publishing.