# BACK TO BECK:
# THE STATE MONAD AND THE CONTINUATION MONAD FROM THE VIEWPOINT OF MONADICITY

by

YUNING FENG

A thesis submitted to the University of Birmingham for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
April 2024

## Abstract

We introduce control algebras, which are algebras of two operations called call/cc and abort. The two operations are natural ones among programs that manipulate their plan of work. The significance of the formulation is that it is operationally natural, yet the notion of algebras is equivalent to the established notion of monad algebras, where the monad is the continuation monad.

We also consider the control effect and the state effect from the perspective of Beck's monadicity theorem. In particular, we notice connections between operationally meaningful objects in the two effects and coequalisers of Beck's pair; the latter are important ingredients in Beck's monadicity theorem. In the case of control, Beck's coequaliser is the set of continuations of the algebra in concern; in the case of state, that coequaliser is the set of determined elements, i.e., elements in the algebra where preceding with update operations would have no effect.

## Acknowledgements

I would like to express my gratitude to Paul Levy for his supervision. I thank the School of Computer Science for granting me funding, including the tuition fee waiver for the PhD programme and the stipend that covered the first three years of the study. I thank the School of Computer Science, and in particular the Theory Group, for nurture and friendship. They are of no doubt the best part of this challenging journey.

I thank my parents, Huilin Tan and Tao Feng, for their long-standing support. I also thank Guoling Chen, Dafeng He, Robert Simon Fong, Wolfgang Brumetz and Rita Muhrer for their thoughtfulness and help. I have also received kind encouragement from many others. I thank all of them, and sincerely ask their forgiveness for not naming everyone here. Finally, if I may, I wish to thank the past selves that trod through mud and left bread crumbs. The calculations and notes were of great help in the last moments of thesis writing.

The title 'Back to Beck' is both a tribute to Jon Beck for his monadicity work, and to the city of Birmingham for her hosting over the years. Among other things, the city is known for its terraced houses, often called 'back-to-backs', built in the industrial revolution period for the working class.

# Contents

# Chapter 1

# Introduction

## 1 Spotting similarity between state and control

The work here can be traced back to research in call-by-push-value languages, introduced by Levy [47, 56, 75]. The language is devised for the study of effects, including state and control ([56] §5.4, §7.8 for control). Levy observes that there is similarity between operations of state and control.[1] Specifically, the *letstk* operation ('let stack') in control can be compared with the *lookup* operation in state; and the *changestk* operation ('change stack') in control can be compared with the *update* operation in state.

| State | Control |
|---|---|
| *lookup* | *letstk* |
| *update* | *changestk* |

An informal description of the operators is as follows. To run a program with the state effect, we need to accompany that program with a variable (in the mathematical sense, outside of the program), whose value is understood as the *current state*, or simply the *state*; it is only accessed by the two state operations, *lookup* and *update*.

- '*lookup* $x$' declares a new identifier $x$; the operation reads the current state and present the value as an identifier $x$.[2]
- *update* takes a state as an argument; '*update* $s$' puts $s$ as the current state.

To run a program with the control effect, we accompany the program with a (*frame*) *stack*. It is a bookkeeping variable (in the mathematical sense, outside of the program); it remembers what the program needs to do after it finishes a small piece of work at hand.[3] The value of the variable is called the *current stack*, or by the same name as the variable, the *stack*.

- '*letstk* $\alpha$' declares a new identifier $\alpha$; the operation reads the current stack, and presents it as the identifier $\alpha$.
- *changestk* takes a stack as an argument; '*changestk* $K$' changes the current stack to $K$.

[1] By personal communication.

[2] We try to avoid the term 'read' in a formal description, as it is the name for the interactive input operator. But using it informally shall be acceptable given that the context is clear.

[3] This corresponds to the frame stack in sequential, procedural programming, hence the name.

The similarity that Levy sees can be phrased as follows: in the control effect, we may consider the current stack as the 'state of control', then *letstk* reads the state of control, and *changestk* changes it. For readers familiar with Crolard's operators *catch* and *throw* [44], we shall mention that Levy's *letstk* and *changestk* are *catch* and *throw* formulated in the call-by-push-value setting.

That similarity between state and control is intuitive, but it is not obvious how to make that precise. A more formal statement of our eventual goal is as follows.

**The state-control similarity problem.** *Find a suitable presentation for algebras of letstk and changestk. Find some algebraic structure common to lookup-update algebras and letstk-changestk algebras; explain the similarity with that common algebraic structure.*

In this work we make a start towards the problem. We shall do the following:

1. Work out algebras for control similar to those of state, possibly with a different choice of control operators.

2. Look for common features of state and control on the semantic side.

The first line of work leads us to control algebras. In the second line of work, we take a close look into monadicity, and make some observations in the instances of state and control.

## A guide to the rest of the introduction

We have made the eventual goal clear in the opening section. In the following section, we shall explain the two specific research problems. Section 3 is on methodology. We aim to show awareness of the Lawvere theory approach to effects, and explain our choice of the monad approach. We do not intend to reinitiate an argument of superiority between the two approaches; rather, it is my wish that readers see the usefulness of both, and adopt any method where their sharpness applies. Section 4 is a brief overview to control operators; it is meant to serve as a preview to chapter 2. That way readers who do not intend to go over the details could still get some idea about relevant control operators. Such readers may return for more details when time is ripe. A guide to other chapters is provided in the end of this chapter.

# 2 An Overview of the two research problems

The two research problems are about control algebras and monadicity. For monadicity, our interest is in the two instances related to state and control respectively.

## Research problem 1: control algebras

We motivate the problem by an introduction to the state case.

### State algebras

Moggi introduced monads to the study of computational effects around 1990 [27]. Monads were associated with effects, thus providing interpretations for call-by-value languages with effects. One thing remained to be addressed: effects should be characterised with their natural-occurring operations. To be more precise, we should define an effect by presenting their natural-occurring operations, together with equations that characterise those operations. In other words, we should define algebras for each effect. About a decade later, Plotkin and Power came up with algebras for the state effect [53]; they found equations for *lookup* and *update* operations that characterise the state effect. About a decade later, Melliès reduced the number of equations to three [73, VI], giving a satisfying presentation of the algebras. In this work we only consider the case where the state is held in a single cell. That is a simplification from the original work, but the change is inessential for our purpose.[4] The formal definition of such algebras is given below. To keep the introduction straightforward, we present algebras over the base category Set. Let $S$ be a set. For the following definition, we shall call it the *set of states.*

**Definition** (Plotkin & Power, Melliès)**.** A *state algebra* consists of a set $X$ and two functions

$$\xi\colon X^S \to X \qquad \zeta\colon S \times X \to X$$

such that the following equations hold:

$$\zeta\big(s, \xi(f)\big) = \zeta\big(s, f(s)\big) \quad \text{for any } s \in S \text{ and } f \in X^S \tag{1.1}$$

$$\zeta\big(s, \zeta(s', x)\big) = \zeta(s', x) \qquad \text{for any } s,\, s' \in S \text{ and } x \in X. \tag{1.2}$$

We call $X$ the *carrier*, $\xi$ the *lookup* operation, and $\zeta$ the *update* operation.

An intuitive reading of the equations is as follows. The $f \in X^S$ in (1.1) may be considered as a function on $S$. The equation reads: an update followed by a lookup can be reduced to a single update, with the resulting value given by $fs$, i.e., feeding the new state $s$ to $f$. Equation (1.2) reads: two consecutive updates can be reduced to one; the latter update (the inner one) overrides the earlier. The two equations are also called the *update-lookup* equation and the *update-update* equation, respectively.

We have mentioned that the equations characterise the state effect. The precise meaning of that is given in the following theorem.

**Theorem** (Plotkin & Power, Melliès)**.** *The category of state algebras is isomorphic to the category of algebras of the state monad.*

That is, we have a one-to-one correspondence of state algebras and algebras of the state monad. In Plotkin and Power's paper, they appeal to Beck's monadicity theorem to argue that algebras of lookup and update are essentially algebras of the state monad. Usually in applications of the monadicity theorem, we are content with an equivalence, i.e., the category in consideration being equivalent to the Eilenberg-Moore category. (It could well be that equivalence is the best we can get.) However, an equivalence here is an understatement. Although we can use a strict version of the monadicity theorem to obtain the isomorphism, the direct way is to construct the structure map of the corresponding Eilenberg-Moore algebra, and verify the unit and multiplication laws by calculation. Plotkin and Power's argument actually carries the needed calculation. The point we would like to make is the following: state algebras and algebras of the state monad only differ in presentation. That is quite different from the usual situations where we consider to apply the monadicity theorem.[5]

[4] In the original work, the state may be held in multiple cells. Semantically, the set of states is considered as a power $V^L$, where $V$ is the set of values, and $L$ is the set of locations. That is the origin of the term *lookup*: the operation shall look up a value at a particular location.

[5] Category theorists may find it evil to speak about isomorphisms between categories. Although this is true for the development of category theory, it is less so for specific instances, like the one we are facing here. It is significant that a state algebra and the corresponding Eilenberg-Moore algebra only differ in operations, but share the same carrier.

### The control case

With state algebras sorted, we look for algebras of the control effect, with natural-occurring operations. This is a more difficult problem than in the state case. For the control effect, operations are more subtle, and there are a few natural-occurring operations to choose from. Let us state the research problem first, then explain how we get there.

**Research Problem 1.** Find suitable equations for control operators *call/cc* and *abort*. Such equations should characterise the control effect presented as the continuation monad, much like how equations of state algebras characterise the state effect.

Operations for the control effect are often called *control operators* in a programming language setting. Here we tend to use 'operations' if we are speaking of them in an algebraic setting, and we tend to use 'operators' if we are speaking of them as programming language constructs.

We have said that operations in the control effect is more subtle than those in the state effect. One may ask where the subtlety is. Informally, the subtlety comes from the higher-order nature of operations and their arguments: they are often functionals. In the category Set, a functional is a function whose arguments are again functions. For example, an element in the set $R^{R^X}$ is a functional; an argument of such a functional is an element in $R^X$, i.e., a function $X \to R$. In an algebra of the continuation monad (over Set), the structure map $\theta\colon R^{R^X} \to X$ is also a functional, whose arguments are elements of $R^{R^X}$; this time $\theta$ is a higher-order functional, as its arguments are again functionals. Due to the higher-order nature, equations of control operations take more effort to formulate and understand. For readers who would like more details, we offer the following exercise. The interest is in (iv); steps (i)–(iii) are meant to serve as a ladder to that. Readers who are more used to diagrammatic reasoning may also want to take a look at the Eilenberg-Moore equations in (iv). The functions-and-elements style allows an operational reading, which we will consider in Chapter 2.

**Exercise.** Work in the base category Set. Fix a set $R$.

(i) For any function $f\colon A \to B$, define $R^f\colon R^B \to R^A$ as precomposition with $f$, i.e., $R^f(g) = g \circ f$, where $g \in R^B$. Verify that this is the curried form of a function, namely the obvious function $R^B \times A \to R$ given by $f$, and curried at $A$. Verify that the definition extends the exponential $R^{(-)}$ to a functor $\mathrm{Set} \to \mathrm{Set}^{\mathrm{op}}$.

(ii) Verify that the functor $R^{(-)}$ is self-adjoint;[6] the unit and counit are both the evaluation function $\mathrm{ev}_X\colon X \to R^{R^X}, \quad x \mapsto \lambda k^{\in R^X}.k(x)$. [7]

(iii) By the general theory of monads, the self-adjoint functor $R^{(-)}$ gives a monad. This is called the *continuation monad*. By the general theory, the multiplication of the monad can be constructed from the counit. Instantiate that to obtain the multiplication in this case, namely

$$R^{\mathrm{ev}_{R^X}}\colon R^{R^{R^{R^X}}} \to R^{R^X}.$$

[6] For pedantic readers, we are abusing the language here in a benign way. From a functor $F\colon \mathcal{C} \to \mathcal{D}$ we can define a functor $F^{\mathrm{op}}\colon \mathcal{C}^{\mathrm{op}} \to \mathcal{D}^{\mathrm{op}}$. Thus we should have said the functor $R^{(-)}$ is adjoint to $(R^{(-)})^{\mathrm{op}}$ to be very precise. There is a similar abuse of language in the description of the counit, where the precise description would be the function corresponding to $\mathrm{ev}_X$ in the opposite category.

[7] Readers new to programming language theory should note that the $\lambda$ here is not a variable, but a piece of notation that introduces the new variable immediately following it. Thus for some $x \in X$, the expression $\lambda k^{\in R^X}.k(x)$ reads: a functional that given any element $k \in R^X$ (a function), produces the value $k(x) \in R$. The $\lambda$-notation is a convenient way to introduce anonymous functions. In a more conventional notation, the function $\mathrm{ev}_X$ can be defined as $\mathrm{ev}_X(x)(k) = k(x)$. Although we introduce $\lambda$ as an alternative notation, it does have mathematical substance. Specifically, it is part of the internal language for cartesian-closed categories. Interested readers may look at P. J. Scott and Lambek [22], and literature mentioned there.

(iv) From the general definition of Eilenberg-Moore algebras, calculate the definition for the continuation monad on Set, with equations written in the usual functions-and-elements style (as opposed to a diagrammatic style). In other words, verify the following statement. An Eilenberg-Moore algebra of the continuation monad on Set can be formulated as follows: it is a set $X$ together with a function $\theta\colon R^{R^X} \to X$ such that the following equations hold:

$$x = \theta\bigl(\mathrm{ev}_X(x)\bigr) \qquad \text{for any } x \in X$$

$$\theta\bigl(\lambda k^{\in R^X}.\, \Phi(k \circ \theta)\bigr) = \theta\Bigl(\lambda k^{\in R^X}.\, \Phi\bigl(\mathrm{ev}_{R^X}(k)\bigr)\Bigr) \qquad \text{for any } \Phi \in R^{R^{R^{R^X}}}.$$

(In later text we shall introduce conventions to simplify the notation; for the moment we shall be content with this presentation.)

The last step of the exercise concerns with the structure map of a monad algebra. One may say, the Eilenberg-Moore algebra equations already characterise structure maps as control operations. Why are we not content with them? Our answer is, although the structure map is associated with some natural-occurring operations (e.g., Felleisen et al.'s $\mathcal{C}$ and Parigot's $\mu$; see Section 4 in this chapter and Chapter 2), we are interested in operations like letstk-changestk (Levy's control operators, as mentioned in Section 1), and call/cc-abort, partly because they are more convenient in programming practice.[8] At this point we shall ask the obvious question: we begin with an interest in letstk and changestk, how come they become call/cc and abort in the research problem? Our answer is as follows. If we try to formulate an algebraic presentation of letstk and changestk, then we will notice that to capture them needs further idea than to do so for call/cc and abort. We are not in a good position to explain it right now, but we can say more towards the end of Chapter 2. As I see it, we shall consider call/cc and abort as the stepping stone towards letstk and changestk.

Such a choice of control operators follows Jaskelioff and Moggi [72, Example 3.8]. In the relevant example, they choose call/cc and abort as the operators of the control effect. They present the two operators for free algebras, defining them as $\lambda$-terms, but equations for general algebras are not stated. To my knowledge, there seems no explicit description of equations for call/cc and abort in the literature. It may seem contradictory that we can define operations for free algebras yet do not have equations for all algebras; after all, for a monad, free algebras are prototypes of all algebras. The explanation is as follows. Although operations of free algebras can be defined explicitly, say as $\lambda$-terms, operations of a general algebra can only be characterised by equations (e.g., compare the multiplication of a monad and the structure map of an Eilenberg-Moore algebra). Also, by 'not having equations' above, we really mean not having a suitable set of equations. On one extreme, we have equations of Eilenberg-Moore algebras, which we are not content with. On the other extreme, we may take all equations that call/cc and abort are supposed to satisfy. That does not provide more understanding either.[9]

We have explained why the research problem may be a non-trivial exercise, and why we are not content with the existing description. We have also mentioned that the research problem is

[8] This also occurs in the state case. The structure map of an algebra of the state monad can be considered as the combination of a lookup followed by an update. (To be precise, the combination is $\xi \circ \zeta^S\colon (S \times X)^S \to X$, where $\xi\colon X^S \to X$ is the lookup operation, and $\zeta\colon S \times X \to X$ is the update operation.) We may say that lookup and update are defined by the Eilenberg-Moore algebra equations, but we are certainly not content with such a set of equations. If that does not seem obvious, then the reader may consider the case of groups. It is known that groups can be defined by divisions alone (see Manes [9, 1.2 Definition] for a presentation). A division may be considered as the combination of a multiplication and an inverse, but we rarely define groups that way.

[9] Informally, we may say equations that characterise call/cc and abort in a general algebra (as opposed to a free algebra) are all those equations that are satisfied by call/cc and abort of free algebras, but do not mention freeness. Obviously, even if we make that precise, such a generic description does not offer much understanding of call/cc and abort.

an intermediate step towards the eventual goal, and brought up where the choice of the control operators come from. Our answer to the research problem is the definition of control algebras (Chapter 3). The justification for that is the following theorem.

**Theorem.** *The category of control algebras is isomorphic to the category of algebras of the continuation monad.*

Chapter 3 is devoted to control algebras and their correspondence with Eilenberg-Moore algebras. We shall present the proof there.

## Research problem 2: monadicity

### Motivation

In the previous section, we point out that the category of state algebras is isomorphic to the category of Eilenberg-Moore algebras of the state monad. We also mention that it is not necessary to use monadicity theorems in the argument. Nevertheless, Plotkin and Power's mentioning of monadicity raises our interest of the topic. Although monadicity theorems are not necessary in the argument of such isomorphisms, they are the right tools if we consider natural resolutions of the state and continuation monads. By those natural resolutions, we mean the adjunction $S \times (-) \dashv (-)^S$ (for a non-empty set $S$), and the self-adjoint $R^{(-)}$ (for a set $R$ consisting of at least two elements). Here we take Set to be the base category.[10] My interest on the topic stems from the following thought.

> The two adjunctions just mentioned are known to be monadic (see the brief introduction to background below). Given that we are interested in specific instances of monadicity, rather than the general theory, shouldn't we also pay attention to relevant data in those two cases, and not just the conclusion? After all, if monadicity is an important property, then there should be a good chance that the relevant data is significant for state and control as well. What could such data say about the two effects?

Guided by the above thought, and by our wish to find common features between state and control, we take a close look at monadicity arguments of the above two adjunctions. Our research problem can be stated as follows.

**Research Problem 2.** Study monadicity in the cases of state and control. We shall

(i) Identify common structure of the monadicity arguments.

(ii) Pay attention to data. More specifically, we consider Beck's pair (to be explained below), and possible operational interpretation of it.

[10] Ideally we would like to be more general. For the base category, we would like to take a cartesian-closed category, or even a monoidal-closed category, possibly with additional properties and structures. However, in terms of monadicity, such generalisations are not completely trivial. For the case of state, Mesablishvili [66, §4] generalises the base category to a cartesian-closed and Cauchy-complete category. For the case of control, I am not aware of further generalisation in the literature. Paré's monadicity theorem [6] requires the base object $R$ to be the subobject classifier, and the monadicity of control we use here works in Set [67]. It seems that we need a suitable abstraction from both works for a generalisation in the control case.

### Background and Beck's pair

Classically, monadicity concerns whether the comparison functor of a given adjunction is an equivalence. If we call the codomain category of the left adjoint the *left category*, then monadicity concerns whether objects in the left category may be considered as algebras, in the sense of Eilenberg-Moore algebras. Popular conditions of monadicity consider diagrams of particular shapes. For example,

($a$) Mac Lane considers *split coequalisers* [40, VI §6, 7]. This version is named 'Beck's theorem' in the book. Other variants can also be found (see §7 exercises; the last paragraph before exercises serves as an overview).

($b$) Johnstone considers *reflexive pairs* [51, A1.1, Theorem 1.1.2]. A textbook treatment can be found in Riehl's book [74, Proposition 5.5.8]. For motivation of the terminology, one may see Pedicchio and Rovatti [62, §1.2].

Let us consider the adjunctions $S \times (-) \dashv (-)^S$ (for a non-empty set $S$), and the self-adjoint $R^{(-)}$ (for a set $R$ consisting of at least two elements). We shall call monadicity of them the monadicity of state and of control respectively. The monadicity of state with the base category Set seems to be a folklore. Métayer's notes [61] presents a proof. Although Beck's theorem is mentioned in passing, the proof in this case is given from scratch. For monadicity of control, Hyland et al. present a proof with the reflexive coequaliser condition, also with Set as the base category [67, §4.1]. Given that there are many versions of monadicity theorems, it would be good if we can have a somewhat canonical condition, at least for state and control. It would be even better if the condition has semantic meaning in each case. It seems Beck's original approach has ingredients we need. One may wonder, there are many later versions of monadicity theorems (like the ones we have mentioned above), why not take one of them? My short response is, the parallel pair that Beck looked at is a particular one, as opposed to a class of parallel pairs with certain diagrammatic properties (e.g., split coequalisers and reflexive pairs mentioned above); importantly, Beck's pair is semantically significant in state and control. We briefly introduce Beck's pair below. We will say more about its semantic meaning later, towards the end of this introduction to monadicity.

Let $F \dashv U$ be an adjunction with counit $\varepsilon$. We shall adopt the following terminology.

**Terminology.** We call the domain category of the left adjoint the *base category*, and the codomain category the *left category*.

The adjunction defines a monad, whose functor part is $T = UF$. For an Eilenberg-Moore algebra of the monad, say $(X, \theta)$, the structure map $\theta$ is a morphism $TX \to X$ in the base category. It is mapped by the left adjoint to the morphism $F\theta \colon FTX \to FX$ in the left category. On the other hand, at the object $FX$ in the left category, the counit is $\varepsilon_{FX} \colon FUFX \to FX$, where $FUF = FT$. Thus for each Eilenberg-Moore algebra $(X, \theta)$ of the monad, we have a parallel pair

$$\begin{tikzcd} FTX \arrow[r, shift left, "F\theta"] \arrow[r, shift right, "\varepsilon_{FX}"'] & FX \end{tikzcd} \qquad \text{in the left category.}$$

That is the parallel pair that Beck uses in the original proof of his monadicity theorem [3, §1, Theorem 1]. We shall call it *Beck's pair*. There is a more symmetric way to formulate Beck's pair, but we shall leave it to Chapter 4.

If for every Eilenberg-Moore algebra, the coequaliser of Beck's pair exists, then such coequalisers define a functor, which is the left adjoint to the comparison functor [3, loc. cit.]. It can be shown that the existence of such coequalisers is necessary for monadicity, and we propose to emphasise adjoint equivalence rather than mere equivalence in monadicity. With that in mind, we phrase the following variant of the monadicity theorem, which centres around Beck's pair. To concentrate on the overall narrative, we choose to leave some technical details unspecified in the statement. They will be made clear later in Chapter 4.

**Theorem.** *Let $F \dashv U$ be an adjunction with counit $\varepsilon$. If, for every algebra of the induced monad, Beck's pair has a coequaliser (consisting of a quotient object and a quotient-forming morphism to that object), then the quotient objects of chosen coequalisers define a left adjoint to the comparison functor, as pointed out by Beck. The corresponding quotient-forming morphisms form a natural transformation; call it $\pi$. If furthermore,*

(i) *Components of $\pi$ have $U$-sections, natural in a particular way (to be made precise in Chapter 4), and*

(ii) *For any object $A$ in the left category, the counit $\varepsilon_A$ is a coequaliser of the evident Beck's pair (defined with the algebra that $A$ is sent to, by the comparison functor),*

*Then the comparison functor becomes an adjoint equivalence. In that case, the data of this adjoint equivalence can be further described as follows.*

1. *The unit of this adjoint equivalence consists of homomorphisms in the Eilenberg-Moore category. It can be written in terms of $\pi$, the quotient-forming morphisms. A component of the inverse can be written in terms of the corresponding $U$-section in* (i)*, together with the structure map of the involved algebra.*

2. *The counit and its inverse are given by the universal property of $\pi$'s components.*

Our variant is slightly more specific than Beck's original argument, in particular in condition (i) and the description of the inverse of the unit. Mesablishvili [66, 2.1. Theorem] formulates Beck's overall argument as a detailed statement (the only such formulation in the literature I am aware of). That formulation also presents Beck's pair (without explicitly naming it), and speaks about an adjoint equivalence rather than mere equivalence. For the difference, the reader may compare our condition (i) here to the condition (ii) in Mesablishvili's formulation.[11] I hope to convince the reader, at least those interested in state and control, that our formulation here is not driven by an artificial interest in inessential reformulations. In general, it's usually good to know what exactly the isomorphisms are for the purpose of applications (whereas such data is often not relevant for general category theory); and in our case, the $U$-sections we mention here do have correspondence in the cases of state and control. We leave details to Chapter 4.

Other expository material of monadicity includes Borceux [34, §4.4], MacDonald and Sobral [60, §2], and Barr and Wells [15, §3.3].[12]

### Beck's pair in state and control

The coequaliser of Beck's pair has meaningful reading in each case of state and control.

($a$) In the state case, the quotient object of such a coequaliser coincides with the set of *determined elements*. These are elements that would not be affected by preceding updates. This seems well-known among researchers familiar with the state effects. For example, it is

[11] It seems that Mesablishvili's formulation confuses the conditions for the unit and counit of the adjoint equivalence. The condition (i) there should be about counit (exactly our condition (ii)), whereas the condition (ii) there should be about the unit.

[12] Barr and Wells' introduction also mentions alternative terminology, e.g., *descent type* and *effective descent type*. That could be quite helpful, as that terminology seems widely used in other subjects, say algebraic geometry.

explained in Métayer's notes [61], without using that terminology. We notice that the set is also the set of global points of the monad algebra in consideration. (In the category Set, an element of a set $X$ can be viewed as a function $1 \to X$, where 1 is a chosen singleton set. A global point is a generalisation of that formulation, to any category with a terminal object. The terminal object takes the role of the singleton set 1.)

(*b*) In the control case, the coequaliser diagram of Beck's pair is the formulation of a particular set of homomorphisms. As before, we still assume Set as the base category; and by homomorphisms, we mean homomorphisms between Eilenberg-Moore algebras. Readers should note that the left category in this case is the opposite category of Set, hence we are actually considering an equaliser diagram in Set, which is a comprehension. We shall omit details of those homomorphisms for now. The important point is that this formulation gives us a new semantic view on continuations. In this view, continuations are seen as homomorphisms (see Chapter 2, §2.3, p44ff., 'the homomorphism principle'). We also see that the Eilenberg-Moore equations of the continuation monad suggest such a view as well (see later part of that same section). In the case that the unit of the monadicity adjunction is an isomorphism, we can even take this view further, and state that all continuations can be viewed as such homomorphisms.

Similar to the state case, we notice that homomorphisms defined by Beck's pair are exactly global copoints, of the algebra that the Beck's pair is defined with. (Global copoints are the dual notion of global points.)

This is probably as much as we can say for a non-technical overview. Interested readers may follow Chapter 4 for details. So far we have introduced both research problems. In the next section we turn to methodology. It is of theoretical interest; readers interested in the control effect may skip it for the first reading, and come back if such theoretical interest arises.[13]

## 3 Approaching effects via Lawvere theories and monads

Effects can be studied as algebras. This can be done via monads, which is applicable to all effects, or via Lawvere theories, which is applicable to many of them. We study algebras of the control effect with the monad approach. This choice of methodology is a necessity: for a technical reason, usual Lawvere theories do not apply to the control effect. We shall give more details later. Although the choice is clear, it is still useful to have an overview of both approaches. The Lawvere theory approach has had a good development in the past two decades or so, and an overview shall give us a better sense of how the current work fits into the broader research. To make sense of the overview, we need to understand at least the definitions of monads and Lawvere theories. In contrast to monads, Lawvere theories are less likely to be on one's learning path. So we present definitions of Lawvere theories and their algebras below, in the hope that the overview be more accessible.

**Lawvere theories and their algebras** Lawvere theories are universal algebras formulated in a categorical way. We consider theories rather than their instances (e.g. the equational theory of monoids rather than specific monoids). Thus operations here only have a formal presence. We only speak about how many inputs they accept, and what equations they satisfy. At least for the definition, we do not distinguish basic operations from other operations; all operations are considered, and they have the same status. Similarly for equations: we do not distinguish axioms

[13] I am tempted to say it is also of metatheoretical interest. It is about 'what is algebra' and 'how to do algebra', as well as particular algebras.

from other equations; all equations are considered, and they have the same status. Choosing basic operations and suitable axioms is a matter afterwards.

**Definition.** A *Lawvere theory* is a category that consists of a countable set of distinct objects $1$, $\mathbf{u}$, $\mathbf{u}^2$, ..., $\mathbf{u}^i$, ..., where $i$ in each $\mathbf{u}^i$ is a natural number, $\mathbf{u}^i$ is the $i$-th power of $\mathbf{u}$, with $\mathbf{u}^0 = 1$ and $\mathbf{u}^1 = \mathbf{u}$.

We call morphisms with the codomain $\mathbf{u}$ *formal operations*, or simply *operations* if confusion is unlikely. Morphisms with codomains $\mathbf{u}^m$ where $m > 1$ can be considered as an $m$-tuple of operations. We call equalities within a homset *equations*. It shall be obvious that the object $\mathbf{u}$ only has a formal presence. It is only formal, and may be considered as a placeholder for possible carriers. For example, when we consider the equational theory of monoids, it does not matter what the carrying set is, although it has to be non-empty in this case.

**Example.** Let us take a closer look at monoids. The theory of monoids consists of a constant—the unit, considered as a nullary operation, and the multiplication, which is a binary operation. They satisfy the unit and associativity laws. Correspondingly, the Lawvere theory of monoids has operations $e\colon 1 \to \mathbf{u}$ and $m\colon \mathbf{u}^2 \to \mathbf{u}$. Other morphisms can be derived using composition and the universal property of powers. The equations on $e$ and $m$ rephrase the corresponding laws in an element-free style:

$$\text{(right unit)} \qquad \mathrm{id}_{\mathbf{u}} = m \circ \langle \mathrm{id}_{\mathbf{u}}, e \circ_1 \langle\rangle \rangle$$

Similarly for the left unit law,

$$\text{(associativity)} \quad m \circ \big\langle \mathrm{pr}_1, m \circ \langle \mathrm{pr}_2, \mathrm{pr}_3 \rangle \big\rangle = m \circ \big\langle m \circ \langle \mathrm{pr}_1, \mathrm{pr}_2 \rangle, \mathrm{pr}_3 \big\rangle \colon \mathbf{u}^3 \to \mathbf{u}.$$

The morphism $\langle\rangle\colon \mathbf{u} \to 1$ (the empty tuple) is the unique morphism given by the terminal object $1$, and the subscript $1$ in $e \circ_1 \langle\rangle$ means to compose at the object $1$. The morphisms $\mathrm{pr}_i$ are the $i$-th projections. Except for $e \circ_1 \langle\rangle$, compositions (those not decorated) are at $\mathbf{u}^2$. Our point is certainly not about insisting on a particular style of notation. With the categorical language understood, we shall be happy to write down equations in the usual style, while keeping in mind that it is a convenient notation for the categorical formulation.

Algebras are defined as follows.

**Definition.** Let $\mathcal{L}$ be a Lawvere theory and $\mathcal{C}$ be a category with finite products. An *algebra* of $\mathcal{L}$ in $\mathcal{C}$ is a finite-product preserving functor $\mathcal{L} \to \mathcal{C}$.

**Example.** We continue with monoids. Classically, a monoid can be defined as a functor from the Lawvere theory of monoids, to the category Set. The formal carrier $\mathbf{u}$ is associated to a set (the carrier), say $X$; the operation $e$ is mapped to a function $1 \to X$ that designates the unit, and the operation $m$ is mapped to the multiplication. We may also say that the functor is a model of the Lawvere theory, and that $\mathbf{u}$, $e$ and $m$ are interpreted as the corresponding set and functions.

It is also useful to have the notion of terms, in particular terms with arguments.

**Definition.** Fix a Lawvere theory and a set $X$, and let $f$ be an $n$-ary formal operation, i.e., a morphism $f\colon \mathbf{u}^n \to \mathbf{u}$. Then for any tuple $x \in X^n$, we call the pair $(f, x)$ an *$n$-ary term with arguments in $X$*, or simply *an $n$-ary term on $X$*. We usually write it as $f(x)$, mimicking the function notation.[14]

[14] Strictly speaking, we shall say a term is a triple $(n, f, x)$, so that the arity number is also part of the data. We have striven for a balance in formality here. For the definition above, the point of introducing some formality is to distinguish the syntactic part from the rest in the convenient notation, and not to be completely formal. In practice the arity number is often left implicit.

The customary notation in the definition extends further. In the case that the operation is unary and it is the identity morphism, we simply write the term as $x$. In the case that the operation is nullary, we simply write the term by the operation (see the monoid example below). In the case that the operation is a composition, say

$$f \circ_{\mathbf{u}^m} \langle t_1, \dots, t_m \rangle \colon \mathbf{u}^n \to \mathbf{u},$$

where $f$ is an $m$-ary operation, and the $m$ operations $t_1$, …, $t_m$ are all $n$-ary, we simply write $f\big(t_1(x), \dots, t_m(x)\big)$ for the term $\big(f \circ \langle t_1, \dots, t_m \rangle, x\big)$.

**Example.** Let us continue with the Lawvere theory of monoids, where $e$, $m$ are the unit and multiplication operations respectively. Let $X$ be a set.

*a.* Any tuple $x \in X^n$ can be viewed as an $n$-ary term on $X$. It is the customary notation for the term $(\mathrm{id}_{\mathbf{u}^n}, x)$.

*b.* The unit $e$ can be viewed as a nullary term on $X$. It is the customary notation for the term $(e, *)$, where $*$ is the sole element of $X^0 \cong 1$.

*c.* For any $x \in X$, $x \cdot x$ can be viewed as a unary term on $X$. It is the customary notation for the term $\big(m \circ \langle \mathrm{id}_{\mathbf{u}}, \mathrm{id}_{\mathbf{u}} \rangle, x\big)$, and we also use the infix notation $\cdot$ in place of $m$.

Thus we can simply write terms $x$, $e$, $x \cdot y$, $(x \cdot y) \cdot z$, etc., as we are used to.

Similar to our earlier comment, here we have a formal definition of terms and a customary notation for them. Once the definition is understood, we are happy writing terms in the convenient way. In the above monoid example, if we consider the binary terms $\big(m \circ \langle \mathrm{pr}_2, \mathrm{pr}_1 \rangle, (x, y)\big)$ and $\big(m, (y, x)\big)$, then we would naturally want to identify them, so that $y \cdot x$ may denote both. That identification touches on a construction; given a Lawvere theory, the construction produces its corresponding monad.[15] We shall not look into details here. Interested readers can learn from the literature, e.g., Borceux [34] §3.3, Pedicchio and Rovatti [62], Robinson [54], and Hyland and Power [68]. Our presentation here has drawn inspiration from them. The original formulation is in Lawvere's thesis [1], Chapter II. There, algebraic theories are defined by presenting the accommodating category altogether (Def. 'the category $\mathcal{T}$ of algebraic theories').

**Literature** We present literature of the two approaches below. It is grouped into themes; a brief description of them is as follows.

*a.* Associate algebraic structures to effects. One may ask, why we would like to do so? And, why this kind of algebraic structures (monads, resp. Lawvere theories)? Literature in this group answers these questions. It is probably not too much over-interpreting, that by answering those questions, literature in this group also proposes answers to the basic question: what is an effect? For the monad approach, the proposal is 'an effect is a monad'; for the Lawvere theory approach, the proposal is 'an effect with only finitary operations is a Lawvere theory'. [16]

[15] The terms we introduced here is one way to associate a Lawvere theory with a monad. This corresponds to the 'concrete way' in Robinson's exposition [54, after Def. 1.5], and the construction can be written as a coend formula, as shown in Hyland and Power's survey [68, Prop. 4.1]. There is an alternative approach, as explained in Robinson's exposition. The equivalence of the two approaches is captured in the bijection

$$\textstyle\int^{n \in \aleph_0} L(n, 1) \times X^n \cong L(X, 1)$$

when $X$ is a natural number. See Hyland and Power's survey for the notation used in the bijection.

[16] The finitary condition can be relaxed, so long as those arities (each being a cardinal) are bounded. For example, a similar proposal can be, 'an effect whose operations all have countable arities is a countable Lawvere theory'.

*b.* Characterise effects with their natural-occurring operations. This problem is more visible for the monad approach. One may say, an Eilenberg-Moore algebra has exactly one operation, i.e., the structure map, whereas an effect's natural-occurring operations are usually not the structure map. How shall we address the difference? For Lawvere theories, tension is on the form of an operation. A Lawvere theory operation takes a (homogeneous) tuple as the argument, i.e., $f(x_1, \ldots, x_n)$ where all $x_i$ come from the same carrier set, when the base category is Set. Can all operations be formulated this way? And, for operations that can be so formulated, what are the equations? [17] Literature in this group answers the above questions. Maybe unexpectedly, sometimes one answer responds to questions from both approaches (Plotkin and Power [53]).

*c.* Characterise effect operations. This only needs to be done on free algebras, as they are prototypes of all algebras. Here we ask, what is an effect operation? Literature in this group proposes its answers.

*d.* Combine effects. Effects can coexist, and the theory needs to address that.

Those problems, together with the emergence of monads and Lawvere theories themselves, and with further developments, form a line of development in the algebraic study of effects. We list literature below, organised into a table.

Table 1.1: Literature of the two approaches

| | Monads | Lawvere theories |
|---|---|---|
| Emerge | From homology and homotopy in late 1950s–mid 1960s.<br>See Mac Lane [40] Ch. VI Notes, Lawvere [1] Author's comments, Barr & Wells [15] §3.8, Hyland & Power [68] etc. for details and historical notes. | From Lawvere's 1963 thesis [1], with the intention to develop a category theory of universal algebra in the broad sense (op. cit., §A.1). |
| | Linton in mid 1960s, showed that Lawvere theories induced monads (and more). See Hyland & Power [68, §4] for references and an exposition. | |
| Associate algebras to effects | Moggi [27] around 1990, observed the connection of effects and *monads* via the sequencing construct, in a call-by-value setting. | (Moggi [27] also mentioned Lawvere theory in passing [following remark 2.1].)<br>Plotkin & Power [48] in early 2000s, introduced $n$-ary effect operators $f(x_1, \ldots, x_n)$. |

Continued on next page

[17] The short answer to the first half is no. This is due to certain operations of the control effect. For example, the call/cc operation is not an $n$-ary operation for any cardinal $n$. We shall say a few more words after presenting the literature. Some operations can be reformulated into this form, which may be less natural for such operations. These include, for example, the output operation of interactive input-output, and the update operation of the state effect.

Table 1.1: Literature of the two approaches (Continued)

| | Monads | Lawvere theories |
|---|---|---|
| Present effects with their natural-occurring operations | (Moggi [27, 23] around 1990, mentioned lookup and update of the state effect with multiple locations.) | Plotkin & Power [48] in early 2000s, formulated nondeterminism and probabilistic nondeterminism as binary operations. |
| | Plotkin & Power [53] in early 2000s, established *lookup-update algebras.* | |
| | Melliès [73] in 2010; simplified lookup-update algebra equations.<br>(Moggi [37] in late 1990s, Jaskelioff & Moggi [72] around 2010; took *call/cc* and *abort* as operations for the control effect.)<br>Our work on control algebras may be placed here in terms of problem and approach. | Plotkin & Power [53] also formulated raising exceptions and interactive I/O as operations. Local state, i.e., local variables was also treated in detail. Plotkin & Power [52, 57] summarised known examples, including partiality that involves order [52]. The Lawvere theory approach does not apply to operations of control in the same way as other effects. |
| | Plotkin & Pretnar [70] in late 2000s, modelled exception handlers. | |
| Algebraic operations (call-by-value) | (Moggi [23, Notation 4.1.2] around 1990s, considered operations in very permissive form.) | |
| | Jaskelioff [69, §4, Def. 15], Jaskelioff & Moggi [72, Def. 3.1] around 2010, introduced *algebraic operations.* They were considering lifting operations of an effect to settings that combine other effects. The key idea was actions on monoids, i.e., morphisms $A \otimes M \to M$ in a monoidal category. Here $M$ is a monoid, and the morphism was required to satisfy associativity. | Plotkin & Power [49, 52, 57] in early 2000s, defined *algebraic operations.* They were looking for suitable conditions to put on natural transformations $(TX)^n \to TX$, so that the natural transformations may be seen as operations come from some effect. The key idea was naturality respecting strength. |
| | It seems that Jaskelioff [69] and Jaskelioff & Moggi [72] chose the same name partly because their definition generalised that of Plotkin & Power's, at least in the non-enriched setting (Jaskelioff, op. cit., §4, Remark 16; Jaskelioff & Moggi, op. cit., Remark 3.7; also see Plotkin & Power [49], §2–3). Obviously the techniques that followed remained different, and led to results with different concerns. | |

Continued on next page

Table 1.1: Literature of the two approaches (Continued)

| | Monads | Lawvere theories |
|---|---|---|
| Combine effects | Moggi [23, Ch. 4], Cenciarelli & Moggi [77], Moggi [37] in late 1980 and 1990s, proposed *monad transformers* (under the name 'monad constructors' in Moggi [23]). This is termed the 'incremental approach' in this line of work. It is 'incremental' in the sense that one starts with an existing effect, then takes a second effect into account. One considers the addition of the second effect as a function that maps existing effects to combined ones. | (The corresponding method in the Lawvere theory approach would be the sum and tensor of Lawvere theories. See below. The difference is that sum and tensor are commutative, i.e., one does not need to distinguish which is the base theory, and which is the theory to be added. This is in contrast to the incremental approach.) |
| | Power [38], Power & Rosolini [41] in late 1990s, proposed a more general approach than monad transformers (consisting of an algebraic structure and an identity-on-object functor). The approach led to consideration of *Lawvere 2-theories*, and *enriched Lawvere theories* [45]. | |
| | Jaskelioff & Moggi [72] in around 2010, rephrased monad transformers in the context of monoidal categories. The corresponding notions were called *monoid transformers*. As in the case of monad transformers, the definition per se imposed no coherence requirements (naturality, functoriality, etc.). The point was, if a monoid transformer has enough coherence, then one can lift a broad class of operations. | Hyland, Plotkin & Power [65] in the 2000s, studied *sum* and *tensor* of Lawvere theories. Sum (+) is the disjoint combination of theories, and tensor (⊗) is a combination where operations from one theory commute with those from the other. They also defined *operation transformers*, which lift operations of a standalone effect to the combined setting.<br>Hyland, Levy, Plotkin & Power [67] investigated combining control with other effects, in the call-by-value setting. They approached the problem via *large Lawvere theories*. |

Continued on next page

Table 1.1: Literature of the two approaches (Continued)

| | Monads | Lawvere theories |
|---|---|---|
| Further developments | Levy [47, 56, 55, 64, 75] in early 2000s, introduced *call-by-push-value*. The language decomposed monads into adjunctions in the study of effects. | Plotkin & Power [49] also defined *generic effects* along with algebraic operations (§5). Informally, an $n$-ary generic effect is an $n$-ary operation symbol. |
| Overview | Hyland, Plotkin & Power [65]<br>Hyland & Power [68] in the 2000s. | |

Occasionally there are typos in the literature. They do not hinder experienced readers, but some of them may pose confusions for those new to the subject. Here are some corrigenda that I think could be helpful.

1. This note applies to the preprint version of Plotkin & Power [53], available from Plotkin's homepage (publication no. 64).[18] In the introduction, on generic effects, the sentence is meant to be '...e.g., to give a generic effect $e \colon \underline{n} \to T\underline{m}$ is equivalent to giving $n$(-many) $m$-ary algebraic families of operations, ...'.

2. This note applies to the preprint version of Plotkin & Power [52] (Plotkin's publication no. 66). In the explanation before Example 2.4, there is a mismatch between the effect equation and the program assertion. The effect equation is an update-update equation, whereas the program assertion corresponds to an update-lookup equation.

3. This note applies to the preprint version of Hyland, Levy, Plotkin & Power [67] (Plotkin's publication no. 71). Before Theorem 2, when $\mathcal{C}_X^{T_E+T}$ first appeared, its codomain is meant to be $(T_E+T)X = T(E+X)$. In the syntax after Theorem 2, the operation **raise** is meant to be a nullary operation. One may write it as $\mathbf{raise}_{\sigma,e}$, i.e., an operation for each type $\sigma$ and each exception $e$. In later terms, $\mathbf{raise}(e)$ was a convenient way to write $\mathbf{raise}_{\sigma,e}$. In Example 1 of §3.1, the isomorphism is meant to be $Z \cong R^{R^{Z+SX}}$.

4. In the 'further developments' theme, we mention generic effects in the Lawvere theory approach. Formally, a generic effect is a morphism $1 \to Tn$ in the base category, where $T$ is the monad corresponding to the Lawvere theory. In Set, we consider $Tn$ as the set of $n$-ary operation symbols; that is the basic idea in the correspondence of Lawvere theories and finitary monads. This correspondence is explained in Robinson's exposition [54], for example. The following corrigendum applies to the preprint version of that exposition. In the proof of Lemma 1.8, first bullet, the equation is meant to be

$$[T(\theta)(f)](y_1, \dots, y_m) = f\big(\theta(x_1), \dots, \theta(x_n)\big).$$

   Here the natural number $n$ is identified with the set of variables $\{x_1, \dots, x_n\}$, and similarly for $m$, which may be identified with $\{y_1, \dots, y_m\}$ to go with the above corrected equation.

[18] The series of papers on Lawvere theories and effects, coauthored by Plotkin, Power and others, often have similar titles. On Plotkin's webpage, he organises his publications into a numbered list, in chronicle order. It seems convenient to refer to those papers by their numbers in the list.

**The two approaches complement each other** We have mentioned that usual Lawvere theories do not apply to the continuation monad $R^{R^{(-)}}$. We shall fill-in some details here. Usual Lawvere theories do not apply in this case for a technical reason: on Set with a not-too-small $R$ (having at least two elements), the monad does not have a rank. (See the expository literature mentioned on p12 for the definition of a rank; similarly for filtered diagrams and filtered colimits below.) A less technical way to say the above is, if we attempt to describe the monad by throwing-in enough $n$-ary operations, then we would need operations of any arity, i.e., arity of any cardinal. A proof of that may be found in Robinson's exposition [54], near the end of §1. The following is a special case that allows an elementary reading, yet preserves the basic idea of the argument.

**Proposition.** *Let $|R| \geq 2$ but finite. Then the continuation monad $R^{R^{(-)}}$ on* Set *is not finitary.*

*Proof.* We consider the category of natural numbers and injections. This is a filtered category, and the colimit is the countable infinite cardinal $\aleph_0$. Now consider its image under $R^{R^{(-)}}$, with $|R| \geq 2$ but finite. We can verify that the functor $R^{(-)} \colon \mathrm{Set} \to \mathrm{Set}^{\mathrm{op}}$ preserves monomorphisms, so the monad preserves injections. The sets in the image of $R^{R^{(-)}}$ are all finite but unbounded, so the colimit under the monad is also $\aleph_0$. However, $R^{R^{\aleph_0}}$ is larger than $\aleph_0$. Thus the monad does not preserve the filtered colimit in this case, i.e., not finitary. □

The basic idea of the correspondence of a Lawvere theory and its monad $T$ is the following: for any set $X$, the value $TX$ may be considered as the set of $X$-ary operations. Thus we have the following interpretation of the above argument. To describe the continuation monad in this setting, we need uncountably many $\aleph_0$-ary operations, $|R^{R^{\aleph_0}}|$-many to be precise. The cardinality means we must have proper $\aleph_0$-ary operations other than essentially finitary ones. By essentially finitary operations, we mean those $\aleph_0$-ary operations that only use finitely-many of its arguments, and there are only countably-many of them. The existence of proper $\aleph_0$-ary operations is in contrast to Lawvere theories, where finitary operations suffice. In a Lawvere theory, all $\aleph_0$-ary operations should be essentially finitary. That is the interpretation of the above argument. We have been careful in saying that usual Lawvere theories do not apply. The unusual Lawvere theories hinted at are large Lawvere theories. Those are used in Hyland, Levy, Plotkin and Power [67, introduction and §4]. However, the large Lawvere theory of a monad is the opposite of the Kleisli category. That essentially brings us back to the monad approach.

We end this overview with some observations. Before going further, I would like to stress the following: the Lawvere theory approach has shown its penetrating power, in particular in analysing combinations of effects. With that put forward, I express softly the following feeling: in some cases the capability of the monad approach seems underestimated, whereas the role of Lawvere theories seems over-emphasised.

(*a*) Plotkin & Power [53] show that the state effect can be defined by the lookup and update operations. More precisely, they present equations for lookup and update, and show that the category of such lookup-date algebras is monadic, i.e., equivalent to the Eilenberg-Moore category. We observe that the Lawvere theory presentation of lookup and update is orthogonal to the argument of monadicity. We may consider lookup and update simply as morphisms $X^S \to X$ and $S \times X \to X$ (with a carrier $X$ for example), and essentially the same calculation applies. Moreover, the calculation can show that the monadicity is an isomorphism: the correspondence between a lookup-update algebra and an Eilenberg-Moore algebra is a correspondence between two structures defined on the same carrier.

(*b*) The operations call/cc and abort define the non-delimited control effect, as we shall show in a later chapter. This is in much the same way as in the state case: we have an isomorphism between the category of call/cc-abort algebras and the category of Eilenberg-Moore algebras.

($c$) Plotkin and Pretnar explain exception handlers in the setting of Lawvere theories [70]. We may give an alternative description with Eilenberg-Moore algebras. Consider the exception-handling term

$$\gamma^{FX}\ \mathsf{handle}\ \{e \mapsto {x_e}^{FX}\}_{e\in E} \colon A$$

('handle exceptions raised within $\gamma$ by $\{e \mapsto x_e\}_{e\in E}$'). Here we type the term in the call-by-push-value style: $FX$ is the computation type whose terms either return a value of $X$ or raise an exception, and $A$ is a computation type with $UA = UFX$. The type $FX$ is interpreted as the free Eilenberg-Moore algebra on $[\![X]\!]$ as usual. To interpret the term and the type $A$, we recall the following fact of the Eilenberg-Moore adjunction: let $f \colon X \to Y$ be a morphism in the base category; if we add to $Y$ an Eilenberg-Moore structure $\delta$, then $f$ extends to a homomorphism, from the free algebra on $X$ to the algebra $(Y, \delta)$. With that in mind, '$\mathsf{handle}\ \{e \mapsto x_e\}_{e\in E}$' can be interpreted as a homomorphism from the free algebra on $[\![X]\!]$: it is a homomorphism that extends the unit $\eta_{[\![X]\!]} \colon [\![X]\!] \to T[\![X]\!]$, and the Eilenberg-Moore structure given on $T[\![X]\!]$ is defined by the exception-handling table $\{e \mapsto x_e\}_{e\in E}$. That Eilenberg-Moore algebra on $T[\![X]\!]$ also interprets the computation type $A$.

($d$) Lookup-update algebras have a combinatorial nature, as seen in Melliés' analysis [73, VI]. The combinatorial nature can be summarised in a sentence: any combination of lookup and update can be equivalently reduced to a lookup followed by an update, with a suitable transformation of the argument. This is similar for call/cc-abort algebras as well, with more subtle combinations than lookup-update ones. We shall present more details in the conclusion.

In summary, we approach algebras of the control effect with monads, and this choice of methodology is more than necessity. The monad approach and the Lawvere theory approach complement each other.

# 4 Control operators

We are interested in control operators $\mathcal{C}$, call/cc, $\mathcal{A}$ in the $\mathcal{C}$ setting; control operators $\mu$, catch, abort, throw in the $\mu$ setting; and letstk, changestk in the call-by-push-value setting. Control operators need to be formulated based on some language facilities, and the '$\mathcal{C}$ setting', '$\mu$ setting' and 'call-by-push-value setting' refer to specific language settings. The $\mathcal{C}$ setting operators were formally described by Felleisen, Friedman, Kolhbecker and Duba, and Felleisen and Friedman in a series of three papers [17, 16, 20]. It was an untyped description, and the typing was given by Griffin later [25], by drawing an analogue to classical logic. Felleisen et al.'s description was remarkably precise given that the subtle typing was unknown at the time. The control operator $\mu$ was introduced by Parigot with an interest in intuitionistic logic [31]. Crolard formalised catch and throw in the $\mu$ setting [44], and presented operational descriptions of them along with $\mu$. Selinger [50], Streicher & Reus [42] each considered categorical aspects of $\mu$. Selinger considered both call-by-value and call-by-name cases of $\mu$. Streicher & Reus studied both $\mathcal{C}$ and $\mu$ in the call-by-name evaluation strategy, made a connection to classical logic, and derived an operational description of $\mathcal{C}$ and $\mu$ using the Krivine machine. The operators letstk and changestk were defined by Levy in the study of control effect [56, §5.4, §7.8]. The study was carried out with the call-by-push-value language, which was discovered by careful observations on the semantics of various effects, and has adjunctions as the core idea.

Some operators in different settings correspond to each other, either roughly or precisely. They can be organised into a table as shown below. The essence of these operators can be described by operations in some algebraic structures. For convenience, we assume that the algebras are carried by sets, and accordingly, the operations are formulated as functions.

| $\mathcal{C}$ setting | $\mu$ setting | CbPV setting | In an algebra with carrier $X$ |
|---|---|---|---|
| $\mathcal{C}$ | $\mu$ | – | $\theta \colon R^{R^X} \to X$ |
| call/cc | catch | letstk | $\xi \colon R^{R^X} \to X$ |
| $\mathcal{A}$ | (abort) | – | $\zeta \colon R \to X$ |
| – | throw | changestk | $\zeta_Y \colon R^Y \times Y \to X$ |

The blanks (–) indicate that such operators are not considered in the literature, or we are less interested in them here. In particular, a blank does not mean that such an operator-language combination cannot exist. For example, it is easy to formulate an operator that corresponds to $\mathcal{C}$ and $\mu$ in the call-by-push-value setting.

We shall give a quick and informal explanation of the operators. The central notion to all control operators is that of current continuations. *Current continuations* are planned future work of a program, or one may say the rest of a program. Current continuations are only 'planned work', as they may be changed by control operators, and that is how control operators deviate a program's evaluation from its planned route. A piece of work that may be assigned as future work is a *continuation.* Thus in evaluating a program with control operators, the state of evaluation is determined by two parts: a program term that are being worked on, and the current continuation. The $\mathcal{C}$ and $\mu$ operators work as follows: they take the current continuation, feed that to a subterm, and leave an empty continuation which will be current in the next evaluation state. For the $\mathcal{C}$ operator, the transition can be written as

$$\mathcal{C}M\,,k \rightsquigarrow Mk\,,\text{nil}.$$

Here $\mathcal{C}M$ is the term we are working on, $k$ is the current continuation, '$\rightsquigarrow$' means 'transits to', and $Mk$ is the term to work on in the new state; the continuation, current in that new state, will be 'nil', which means the empty continuation.[19] Thus $\mathcal{C}$ and $\mu$ incorporate planned future work into the term we work on, and consider that all the work we need to do. The operators call/cc, catch and letstk are similar to $\mathcal{C}$ and $\mu$, but with the notable difference that they leave the current continuation intact. In the case of call/cc, the transition will be written as

$$\text{call/cc}\,M\,,k \rightsquigarrow Mk\,,k.$$

The operators $\mathcal{A}$ and abort discard the current continuation, and consider the subterms they lead are all the work to work on. The transition of $\mathcal{A}$ can be written as

$$\mathcal{A}M\,,k \rightsquigarrow M\,,\text{nil}.$$

We have bracketed the abort operator in the table. This is because it was not in the literature, but we define it to exhibit the same idea in the $\mu$ setting parallel to the $\mathcal{C}$ setting. The operators throw and changestk also discards the current continuation, but they also install a new one, together with a term to work on. In the throw case, the transition can be written as

$$\text{throw}\,M\,K\,,k \rightsquigarrow M\,,K.$$

[19] The formal name of such evaluation states is 'configurations'. We are also using the term 'state' by its informal meaning here.

That is, the current continuation $k$ is discarded, and for the next evaluation state, we will install the continuation $K$ and consider it current. The term to work on will become the term $M$.

Our interest is to get to the algebraic presentations of those control operators. For convenience, we shall work in the category Set. We fix a set $R$, and call it the set of responses.[20] Then $R^{R^{(-)}}$ is the continuation monad with the response set $R$. The essence of $\mathcal{C}$ and $\mu$ is the structural map of an Eilenberg-Moore algebra, say $\theta\colon R^{R^X} \to X$, on a set $X$, of the continuation monad. The essence of call/cc, catch, and letstk is an operation $\xi\colon X^{R^X} \to X$. To form an algebra, the operation $\xi$ needs to be paired with a second operation, namely $\zeta\colon R \to X$, which is the essence of $\mathcal{A}$ and abort. The two operations $\xi$, $\zeta$, together with equations they need to satisfy, form a control algebra on a set $X$. There is a good correspondence between control algebras and Eilenberg-Moore algebras of the continuation monad, and that is one thing we would like to show in this work. That result can be seen as the theoretical formulation of some observations on the control operators: the control operator $\mathcal{C}$ and the pair call/cc-and-$\mathcal{A}$ can be expressed in each other; so do the operator $\mu$ and the pair catch-and-abort.

On the other hand, catch can be paired with throw, and letstk is paired with changestk (Crolard [44], Levy [56] §7.8). The essence of throw and changestk is an operation $\zeta_Y : R^Y \times Y \to X$ with a parameter $Y$, also a set. From existing experience with such operators, we expect that operations $\xi$ and $\zeta_Y$ also form an algebra, with suitable equations. This is yet to be proved. It is beyond the scope of the current work.

Even not considering delimited control (to be mentioned next), there are other interesting control phenomena that we do not study here. There are Thielecke's CPS calculus and control operators inspired categorically [39]. There is Escardo's selection monad $X \mapsto X^{R^X}$ [71], whose algebras have the $\xi$ operation described above as structural maps, and they corresponds to call/cc, catch and letstk. Remarkably, this is a common feature between state and control: there is a smaller monad (the reader monad, the selection monad) that resides in the bigger monad (the state monad, resp. the continuation monad), and, the structural map of an algebra of the smaller monad is one operation in the two-operation algebra of the bigger monad (lookup-update algebras, resp. control algebras). There is also Plotkin and Pretnar's algebraic analysis of exception handlers [70]; the practical significance of the analysis is quickly recognised.

**Delimited control** Delimited control is closely related to the understanding of the control effect, but it is out of the scope for this thesis. Some early papers that originate the research include the following: Felleisen [21] proposed the *prompt* operator; Sitaram and Felleisen considered practical aspects of the delimited $\mathcal{C}$ (there named *control*) and prompts [26]; Danvy and Filinski introduced control operators *shift* and *reset*, which are similar to Felleisen's prompts and $\mathcal{F}$ (the operator $\mathcal{C}$ in the delimited setting) [24]. We shall say a few more words in the next chapter.

# 5 An outline of other chapters

In Chapter 2, we introduce continuations and a few control operators in detail, including Felleisen et al.'s $\mathcal{C}$, Parigot's $\mu$, call/cc and the abort operator $\mathcal{A}$, and Crolard's catch. We consider operational equations of them, in particular those equations that correspond to the defining equations of a control algebra. We use this chapter to motivate the defining equations of a control algebra, from an operational perspective. Then in Chapter 3, we introduce control algebras. We give the definition, and consider intrinsic control algebras as example. We also

[20] $R$ was considered as the set of system states in early work on the semantics of the 'goto' command and on continuations. See, e.g. Strachey and Wadsworth [7].

look into homomorphisms, and propose that continuations may be defined as homomorphisms to the result set algebra. The isomorphism theorem is about the isomorphism between the category of control algebras, and the Eilenberg-Moore category of the continuation monad. By establishing the isomorphism, we show that our definition of control algebras are correct. The proof of the isomorphism is split into two part. The first part is about the main line of argument, whereas the second part is about an intermediate step that takes some effort to make. In Chapter 4, we explain Beck's pair, and show that the set of continuations, and the set of determined elements, are each the coequaliser of Beck's pair in their own setting. We also explain the viewpoint of global copoints and global points, on continuations and on determined elements respectively. Finally, we summarise results of the thesis, and explain future work.

# Chapter 2

# An operational and comparative introduction to control operators

The chapter is about operational origins of control algebras. In principle we can regard control algebras as a pure mathematical theory, ignoring any operational consideration. That could be a reasonable approach once we have made the initial step of abstraction; such a step brings us from basic observations (operational observations in this case) to ideal models (semantic equations in this case). Before that initial step, we are rightful to question whether the symbol game has any meaning at all. For the study of programming languages, a large part of such meaning comes from operational description.[1] For example, we encounter many higher-order functionals in the study of control (e.g., see explanation in Chapter 1, 'Research problem 1'; Chapter 3 on control algebras is spread with higher functionals). The interest to such functionals would become quite mysterious if the operational origins are completely ignored. Similarly, if operational origins are ignored, then significance of control algebra equations could hardly be appreciated, as we already have equations of Eilenberg-Moore algebras. Thus to provide some motivation, we give an introduction to control operators here, tailored to our algebraic view.

As it happened, I obtained the equations of control algebras by drawing inspiration from the state case; some equations (the multiplication ones) were mysterious to me initially. It was afterwards that I eventually saw the operational reading. However, for a clear exposition, we may leave that historical incidence aside, and consider an alternative line of development. In this alternative development, we know what operational equations are significant; we observe them, and associate them with corresponding equations of relevant algebras (i.e., Eilenberg-Moore algebras and control algebras). That shall motivate us for Chapter 3 on control algebras.

**A guide to the chapter** The chapter serves multiple purposes. It is an extended literature review on continuations (§1) and some well-known control operators (§2–5). It draws informal analogues between operators $\mathcal{C}$, $\mu$ and the structure map (§2), and between call/cc etc. and operations of a control algebra (§3). The latter analogue helps us build some intuition of control algebras, and get us prepared for Chapter 3.[2] The current chapter also brings up the homomorphism view of continuations (§2.3, 'Continuations as homomorphisms', p44), and tries to explain the difficulty of our eventual goal in more detail (§5). Among the sections, Section 1 on

[1] Personally I take it a little further. I tend to think operational description provides the golden standard or ground truth of semantic work.

[2] Chapter 3 is logically self-contained. However, readers may find the mathematical development more interesting if they have some operational intuition, especially at the very beginning of the chapter.

continuations should be helpful to readers new to the control effect. Later part of Section 1 and sections 2 and 3 have more technical details. The central analogy I hope readers pay attention to is the one between operational equations and equations of algebras (either Eilenberg-Moore algebras or control algebras). Section 4 mentions control operators whose control-changing effect is bounded, as opposed to control operators we study in detail here. Those control operators are out of scope for this thesis, but they are more general, more useful, and semantically more challenging.

For readers with limited time, there are the following reading options. If they are new to the subject, the beginning of Section 1 (on continuations) should still be an easy read. If time permits, they may want to be aware of the 'Continuations as homomorphisms' principle (p44ff). Although reading that without knowing earlier material may leave them with some questions, it should still be beneficial to have the idea in mind. For readers new to the subject, getting to know call/cc and abort with sufficient detail may take some time; for a while, they may have to treat control algebras in Chapter 3 as a purely mathematical subject, taking for granted that the equations are operationally motivated. They may pick up operational descriptions in this chapter along the way, bit by bit. For readers familiar with control, they may want to read the 'Continuations as homomorphisms' principle, and our analogy of equations for call/cc and abort (presented in §3, p50). I imagine that for such readers, other parts of the chapter should become trivial by now, with the description given above.[3]

For readers new to the subject and have the time to read the chapter, I do hope that it serves as a reasonable tutorial, complemented with literature mentioned in the text.

# 1 Continuations

We study the control effect. At the centre of this study is the notion of continuations, and in particular, the notion of the current continuation in an operational setting. Informally, the current continuation is the work that a program needs to continue with. It is also said to be 'the rest of a program'. Then a continuation is work that may be the rest of some program, not necessarily current. Without bringing in any formal setting, we make a compromise and describe its basic idea in an analogy.

**Example.** Suppose that we have a task that needs three steps, and suppose we have finished step 1. Then the rest of the task, steps 2–3, may be considered as the current continuation. The steps 1–3, 2–3, and the step 3 alone can all be regarded as continuations.[4]

The concept of continuations is useful. It helps understand jumps in imperative programs in a precise manner, where these jumps include goto's, labels, function and procedure returns ([7], also see the monograph [10, §1.5], and textbook treatments [18, 11, 12, 28][5]). The key idea there is that we can describe a working state extensionally. In the analogy above, we can describe the state after finishing step 1 by saying, if we proceed with steps 2–3 and finish them, then we will finish the task successfully. If we are able to do so for any steps that we can proceed with, together with the corresponding new states we shall arrive at, then we fully describe the state

[3] I hesitate to say that there may be minor novelty in the presentation here and there.

[4] Mathematicians should note that there is no technical connection, between continuations here and continuations in complex analysis. The terminology is a coincidence. One analogy that could help is the notion of dual space in the subject of vector spaces. Informally, we may think of continuations as functionals in some dual space. That analogy is technically wrong but quite useful initially. Once we see typing and interpretations of control operators, that analogy will no longer be needed.

[5] Despite being decades old, these are probably still the only textbook treatments on the topic. Other textbooks, e.g. Gunter's [29] and Winskel's [33] have their own strengths and emphases.

after finishing step 1. In later part of this introduction, we shall formulate continuation-passing style translations, which formalise that idea in a simple setting.

On the other hand, continuations can be manipulated to deviate a program from its planned route. With the same analogy above, suppose that after finishing step 1, we find the outcome good enough so that step 2 can be skipped. Then we drop the original plan, which is to continue with steps 2–3, and replace it with a new one, i.e., to continue with step 3. That replacement can be considered as a manipulation of continuations. In functional programming languages, control operators can be added to achieve such manipulations. For example, Common Lisp's *catch* and *throw* can be used to skip planned work [13, §7.10 Dynamic non-local exits]; Scheme's *call-with-current-continuation* (call/cc) can be used for the same purpose [14], but is more powerful than the usual exit in an imperative language [46]. Example uses of such control operators can be seen in Felleisen and Friedman's motivating example in an analysis of a different operator [16], Thielecke's motivating examples in his thesis [39], and literature mentioned in those two pieces of work. On the theoretic side, Landin's **J** [2] and Reynolds' **escape** [5] are early but influential control operators.[6] Thielecke gives an introduction to Landin's original paper later, putting the work in context, i.e., with modern understanding of control [43]. Reynolds' historical survey traces the discovery of continuations [32].

Manipulating continuations is also a way to implement concurrency. This is an interesting topic, and it is a potential application of the theory of control. Nevertheless, it is beyond the scope of our study, and we shall not get into details.

**Operational descriptions** Continuations have different presentations in different operational formalisms. We list three formalisms: CK-machines, deterministic reduction with applicative contexts, and continuation-passing style translations. In the call-by-value variants, there is a distinction of values and computations. A variable $x$ and a abstraction $\lambda x.M$ may be considered as a value or a computation. Applications $MN$ are always considered as computations. In the description of CK-machines and deterministic reduction systems, we reserve the meta variable $V$ for terms that may regard as values.

($a$) In a CK-machine, continuations are presented as stacks. We describe CK-machines below, adapted from [76]. A CK-machine consists of configurations and a partial endofunction on them. Each pair in the graph of the function is called a *transition*.

   (i) A *configuration* is a pair $(M, K)$ where $M$ is a term and $K$ is a stack.

   (ii) A *stack* is a finite list where the only ways to manipulate the list is either to prepend an element ($e :: K$ for an element $e$ and a list $K$), or to remove the first element from a non-empty list. By convention, we write an empty stack by nil. Exactly what elements can be added to the stack depends on the variant of the machine. This can be given together with the definition of transitions.

   (iii) We write a transition from $(M, K)$ to $(M', K')$ as

$$M\,,K \rightsquigarrow M'\,,K'\;.$$

[6] To understand Landin's operator, it may be more efficient to start from Felleisen's description [19]. There the transition rules are presented in a concise format, which is easier to understand and is also conventional by now. The reformulation also incorporates a clarification to the original transition algorithm, attributed to Burge. As I understand, the issue in the original algorithm is as follows. If **J** is considered as an operator, not an identifier, then there is no branch to push **J** to the stack. Thus the existing branch to look for **J** in the stack would not happen. If **J** is considered as an identifier by the machine, then there should be an entry in the environment that binds the operator to itself. Felleisen's reformulation solves the issue by considering $\mathcal{J}$ as an operator. The corresponding corrected algorithm is included in Thielecke's introduction to the original paper, mentioned below.

In a CK-machine, a stack is considered as a continuation.

**Call-by-value** The call-by-value CK-machine has two variants, depending on whether the function or the argument is handled first in applications.

(i) For the function first variant, the transitions are as follows.

$$\begin{aligned} MN\,, k &\rightsquigarrow M\,, N :: K \\ V, N :: K &\rightsquigarrow N\,, V[\cdot] :: K \\ V, (\lambda x.M)[\cdot] :: K &\rightsquigarrow M[V/x]\,, K. \end{aligned}$$

Thus stacks are, inductively

$$\text{nil} \quad N :: K \quad V[\cdot] :: K$$

where $K$ is a stack.

(ii) The argument first transitions are

$$\begin{aligned} MN\,, k &\rightsquigarrow N\,, M[\cdot] :: K \\ V, M[\cdot] :: K &\rightsquigarrow M\,, V :: K \\ \lambda x.M\,, V :: K &\rightsquigarrow M[V/x]\,, K\ . \end{aligned}$$

Thus stacks are, inductively

$$\text{nil} \quad M[\cdot] :: K \quad V :: K$$

where $K$ is a stack.

**Call-by-name** The call-by-name transitions are as follows.

$$\begin{aligned} MN\,, K &\rightsquigarrow M\,, N :: K \\ \lambda x.M\,, N :: K &\rightsquigarrow M[N/x] :: K\ . \end{aligned}$$

Thus stacks are, inductively

$$\text{nil} \quad N :: K$$

where $K$ is a stack.

**Continuations** In this formalism, *continuations* are stacks. With a configuration in consideration, the *current continuation* is the stack of the configuration.

($b$) In a deterministic reduction system, continuations are presented as evaluation contexts.

**Call-by-value** Like in the CK-machine case, we shall distinguish the function-first variant and the argument-first variant. *Evaluation contexts* are, inductively,

$$\begin{aligned} \text{function-first variant:} &\quad [\,] \quad CN \quad VC \\ \text{argument-first variant:} &\quad [\,] \quad MC \quad CV\ , \end{aligned}$$

where $[\,]$ is a distinguished free variable, called the *hole* of the context, $C$ is an evaluation context, $N$ is any term, and $V$ is any term that may regard as a value. In either case, we verify that

**Lemma.** *A closed term is either $\lambda x.M$ for some $M$, or there is a unique evaluation context $C$ such that the term is*

$$C\big[(\lambda x.M)V\big]$$

*for some $M$ and, some $V$ that may regard as a value.*

Then we can define a partial endofunction on terms by defining

$$C\big[(\lambda x.M)V\big] \mapsto C\big[M[V/x]\big].$$

We call the partial function the call-by-value reduction system.

**Call-by-name** In this case, *evaluation contexts* are, inductively,

$$[\,] \quad CN$$

where $C$ is an evaluation context, and $N$ is any term. We verify that

**Lemma.** *A closed term is either $\lambda x.M$ for some $M$, or there is a unique evaluation context $C$ such that the term can be written as*

$$C\big[(\lambda x.M)N\big],$$

*where $M$, $N$ are any terms.*

Then we define a partial endofunction on terms by defining

$$C\big[(\lambda x.M)N\big] \mapsto C\big[M[N/x]\big]\ .$$

We call the partial endofunction the call-by-name reduction system.

**Continuations** In this formalism, *continuations* are evaluation contexts. In order to speak about current continuations, we need to recover the notion of configurations. Fix a variant of the reduction system introduced above. we shall say that a *configuration* is a pair $(M, C)$ where $M$ is a term and $C$ is an evaluation context. The arguments that establish the the lemmas above also show that starting from a configuration $(M, [\,])$, we can recover transition rules in analogue to those of CK-machines. Then, with a configuration in consideration, a *current continuation* is the evaluation context component of the configuration.

$(c)$ In a continuation-passing style translation, continuations are presented as the free variable $k$ passed to a translated term $[\![M]\!]$.

**Call-by-value** The call-by-value continuation-passing style translation is as follows. Since there is a distinction between values and computations, there are two kinds of translations: one for value, $[\![-]\!]_v$ and one for computation, $[\![-]\!]$, defined by a mutual induction.

1. Value translations are

$$[\![x]\!]_v = x$$
$$[\![\lambda x.M]\!]_v\, nh = \big(\lambda x.[\![M]\!]h\big)n$$

[In the translation of abstractions, the free variable $h$ is considered as a continuation.]

2. Computation translations are

$$\begin{aligned}
[\![x]\!]k &= k[\![x]\!]_v \\
[\![\lambda x.M]\!]k &= k[\![\lambda x.M]\!]_v \\
[\![MN]\!]k &= \begin{cases} [\![M]\!]\big(\lambda m.[\![N]\!](\lambda n.mnk)\big) & \text{function first} \\ [\![N]\!]\big(\lambda n.[\![M]\!](\lambda m.mnk)\big) & \text{argument first} \end{cases}
\end{aligned}$$

   In the translation $[\![MN]\!]$, either translation can be used, so long as the choice is consistent. [In all clauses $[\![M]\!]k$, the free variable $k$ is considered as a continuation to $M$.]

**Call-by-name** For the call-by-name translation, we need some preparation. We adopt a suitable coding for pairs and projections, i.e.,

$$\begin{aligned}
\langle M, N\rangle &= \mathrm{ev}_{M,N} = \lambda f.fMN \\
\pi_1 P &= P(\lambda x.\lambda y.x)
\end{aligned}$$

and similarly for $\pi_2$. It is obvious how to define currying and uncurrying. Then the translation $[\![-]\!]$ is defined inductively as follows:

$$\begin{aligned}
[\![x]\!]k &= xk \\
[\![\lambda x.M]\!]\langle n, k\rangle &= \big(\lambda x.[\![M]\!]k\big)n \\
[\![MN]\!]k &= [\![M]\!]\big\langle [\![N]\!], k\big\rangle.
\end{aligned}$$

In the translations $[\![x]\!]$ and $[\![MN]\!]$, the free variable $k$ is considered as a continuation. In the translation $[\![\lambda x.M]\!]$, the pair $\langle n, k\rangle$ is considered as the continuation to $\lambda x.M$, whereas in the translated term, in the subterm $[\![M]\!]k$, the free variable $k$ is considered as a continuation to $M$.

Some comments follow.

1. In the literature, the translation is often given in a form that uses successive applications, and does not use pairs. We use pairs as this form generalises to the typed case.
2. Here we use the call-by-name translation where an abstraction is natively a computation, described by Hofmann and Streicher [36, §4.4] (mentioned in Selinger [50]), and by Streicher and Reus [42, §1]. This is an improvement to Plotkin's original translation, where an abstraction is natively a value, but turned into a computation that returns that value [8, p153] (also see Streicher and Reus' description [42, §1]).

**Continuations** In the untyped continuation-passing style translations, continuations are assumed and used, but the concept is not explicitly defined. For example, in $[\![M]\!]k$, the free variable $k$ is supposed to bind to some continuation. This is resolved in the typed setting, where types of continuations are explicitly formulated. This flexibility contains a subtle difference between continuations in continuation-passing style translations and those in a CK-machine or evaluation-context descriptions. Namely, continuations in a CK-machine or evaluation-text descriptions *have definite ends*, whereas those in the continuation-passing style *have indefinite ends* in usual formulations. It is probably not fruitful to attempt an explanation in the untyped setting, so the above note serves as a confirmation to the reader that there is some difference here.

Nevertheless, the basic idea that a continuation is ‘the rest of some program’ still stand, and here is a useful view that connects continuation-passing style translations with previous operational descriptions. In working with continuation-passing style translations, we may consider the expression $[\![M]\!]k$ as an interpretation of a configuration. In this configuration, $M$ is the term in consideration, and $k$ refers to a current continuation.

For convenience, we have introduced continuations in an untyped setting. We shall consider typed continuations with control operators added.

# 2 The operator $\mathcal{C}$ and the operator $\mu$

There are only a few basic control operator ideas. We shall introduce the following ones, organised according to the corresponding algebraic structures.

1. Felleisen et al.’s $\mathcal{C}$ and Parigot’s $\mu$. They are related to the structure map $R^{R^X} \to X$ of an Eilenberg-Moore algebra, of the continuation monad with the answer object $R$. We use this as the basic case to build up understanding for later operators and algebras.

2. The operator call/cc, which appears in programming languages Scheme and ML, Crolard’s catch and Levy’s letstk. All of these are related to the operation $X^{R^X} \to X$ in a control algebra, and we shall refer to it as the call/cc operation. The operators here and the abort-like operators below are our main interest, and we would like to pay attention to the syntactic equations that correspond to control algebra equations.

3. The operator abort as in Scheme and Felleisen et al.’s $\mathcal{A}$. They are related to the operation $R \to X$ in a control algebra, and we shall refer to it as the abort operation.

4. Crolard’s throw, and Levy’s changestk. They do not directly relate to operations in a control algebra. Existing studies suggest that pairing the call/cc operation with the operation similar to *throw* or *changestk* should give us an algebraic structure that is essentially the same as a control algebra (e.g. Levy’s language with *letstk* and *changestk* [56]). This is the ‘whisper in the wind’ that we mention at the very beginning of the thesis.

We shall address Felleisen et al.’s $\mathcal{C}$ and Parigot’s $\mu$ in this subsection, then address other control operators in subsequent ones.

There are other control operators, e.g. Thielecke’s categorical combinators: force, $\phi$, $\neg$ and thunk [39, §1.2, §4.1]. Thielecke is interested in categorical structures that efficiently organise the knowledge of continuation-passing style, e.g. $\otimes\neg$-categories (§4.3), but we are mainly interested in one particular presentation of Eilenberg-Moore algebras, namely call/cc and abort.

## 2.1 The operator $\mathcal{C}$

**Operational descriptions** In a series of papers [17, 16, 20], Felleisen, Friedman, Kohlbecker and Duba introduce the control operators $\mathcal{C}$ and $\mathcal{A}$ in an untyped setting. The operator $\mathcal{C}$ captures the current continuation, whereas the operator $\mathcal{A}$ discards it. Later studies include Griffin's work on typing, which drew an analogue from classical logic [25]. De Groote linked a variant of the operators to a variant of Parigot's $\mu$ [35] (note Crolard's comments on the difference to the original $\mu$ operator [44]), and Streicher and Reus connected the operators with classical logic and abstract machines [42]. Here we shall concentrate at the operator $\mathcal{C}$, as it is related to the structural map of an Eilenberg-Moore algebra. Our introduction to the abort operator $\mathcal{A}$ will come later.

**Definition.** The language of $\mathcal{C}$ is obtained by adding new term constructions $\mathcal{C}M$ to the untyped $\lambda$-calculus. That is, terms of the new language are, inductively,

$$x \quad \lambda x.P \quad MN \quad \mathcal{C}M$$

where $x$ is any variable, and $P$, $M$, $N$ are terms of this language.[7]

In $\mathcal{C}M$, the term $M$ is intended to take a continuation as an argument. In an untyped setting, that intention is not visible from the syntax, but only from operational descriptions. Regardless of the formalism and the evaluation strategy, the operational intention of the control operators are as follows.

1. Suppose we would like to evaluate $\mathcal{C}M$, where the current continuation is $k$. Depending on the formalism, such $k$ may be a stack in an abstract machine [16, §3, the CK-machine], a suitable term context ([20, Definition 2.2], also see de Groote's description [35]), or simply a free variable in continuation-passing style translations. The operator $\mathcal{C}$ turn the current continuation into an argument for $M$, and clear the current continuation.

2. Eventually, some $\lambda$ abstraction will pick up the argument.

3. Further into the future, $k$ may be used as a function and applied to some term $N$. At that point, the continuation current at that moment will be replaced by $k$, and evaluation continues with $N$.

We put this description into three formalisms, all adapted from Felleisen et al's papers.

($a$) In a CK-machine formalism, the above description may be presented as the following transition rules:

$$\begin{aligned} \mathcal{C}M\,, k &\rightsquigarrow M\,, \mathrm{coer}\,k :: \mathrm{nil} && && (2.1)\\ V, \mathrm{coer}\,k\,[\cdot] :: k_0 &\rightsquigarrow V\,, k && \text{call-by-value, any order} && (2.2)\\ \mathrm{coer}\,k\,, N :: k_0 &\rightsquigarrow N\,, k && \text{call-by-name.} && (2.3) \end{aligned}$$

Here $k$, $k_0$ are stacks, nil is the empty stack, and $a :: k$ is the result of pushing $a$ to the stack $k$; $\mathrm{coer}\,k$ is $k$ considered as a term, which may be regarded as a value, $\mathrm{coer}\,k\,[\cdot]$ is similar but used as a function; $M$, $N$ are terms of a suitable extension of the language, to include terms $\mathrm{coer}\,k$.[8]

($b$) In a deterministic reduction formalism, the reduction rules may be:

$$C[\mathcal{C}M] \mapsto M\,\mathrm{coer}C$$

[7] Felleisen et al. named the language $\Lambda_c$.

[8] In Felleisen and Friedman [16], the notation for $\mathrm{coer}\,k$ was $\langle \mathbf{p}, k\rangle$.

$$C\big[\mathrm{coer}\,D\,N\big] \mapsto D[N].$$

Here $C$, $D$ are suitable evaluation contexts, e.g. call-by-value or call-by-name applicative contexts; $C[M]$ is the term obtained by filling the hole of the context with $M$, $\mathrm{coer}\,C$ is the context $C$ considered as a term, to distinguish the term '$M\,\mathrm{coer}C$' from the context $MC$ (coer binds tight). We shall extend the language accordingly to incorporate new terms $\mathrm{coer}\,C$.

($c$) The continuation-passing style translations are as follows. Recall the coding of pairing

$$\langle M, N\rangle = \mathrm{ev}_{M,N} = \lambda f.fMN$$

and projections. For any term $K$, we define the *continuation-function coercion* of $K$ as

$$[K] = \begin{cases} \lambda x.\lambda k_0.Kx & \text{call-by-value} \\ \lambda\langle x, k_0\rangle.xK & \text{call-by-name.}^9 \end{cases}$$

Then the translation clauses are

$$[\![\mathcal{C}M]\!]k = [\![M]\!]\,\mathrm{ev}_{[k],\mathbf{I}} = [\![M]\!]\big\langle[k], \mathbf{I}\big\rangle\,,$$

where $[\![-]\!]$ is the translation function from $\Lambda_c$ to the $\lambda$-calculus, $k$ is a free variable, and $\mathbf{I}$ is the identity combinator $\lambda x.x$, considered as the trivial continuation. In the two presentations of $[\![\mathcal{C}M]\!]$, the one with ev argument generalises to typed call-by-value case, and the other with pair argument generalises to typed call-by-name case. For the non-effectful clauses, the call-by-value translation is as usual, whereas the call-by-name translation uses pairs. See the introduction to continuations at the beginning of the section.

Some comments follow. The call-by-value and call-by-name translations of $\mathcal{C}$ differ only in the definition of coercion. For the similarity, both variants discard the continuation $k_0$ that would receive the remainder of the program current at that moment, as $K$ is taken as the new continuation. The difference is in the use of $K$. It is consistent with the translations of free variables:

$$[\![x]\!]k = \begin{cases} kx & \text{call-by-value} \\ xk & \text{call-by-name.} \end{cases}$$

This difference is only due to what the free variables refer to in the target language, and is irrelevant to the control operator. (In call-by-value, a free variable in a translated term refers to a value, and it is fed to a continuation; in call-by-name, a free variable refers to a computation, which applies to some continuation.)

(*End of operational descriptions*)

**Equations of $\mathcal{C}$** We are interested in operational equations that correspond to Eilenberg-Moore algebra equations, of the continuation monad. For different formalisms, the congruences that occur in the equations may be different, but we need them to satisfy at least the following conditions.

[9] The notation $[K]$ is inspired by Parigot's notation $[\alpha]M$ in the $\lambda\mu$-calculus. The intended meaning wasn't explicitly stated there, and we could only speculate that coercion might be the intention. We shall use the notation sparingly to avoid confusion with the hole-in-context notation $[\cdot]$.

($a$) For the CK-machine description, the congruence should identify terms $M$, $N$ such that, paired with the same stack $K$, the configurations $(M\,,K)$ and $(N\,,K)$ eventually reach the same configuration.

($b$) For the deterministic reduction description, the congruence should identify terms $M$, $N$ such that, in the same context $C$, terms $C[M]$ and $C[N]$ eventually reduce to the same term.

($c$) For the continuation-passing style description, the congruence should be $\beta\eta$-equality of the translated $\lambda$-terms. That is, $M$, $N$ are identified if $[\![M]\!]k =_{\beta\eta} [\![N]\!]k$ for any free variable $k$.

Let us write $\simeq$ for such a congruence. We expect the following equations to hold, regardless of the operational description, and for both call-by-value and call-by-name variants:

$$\begin{aligned} M &\simeq \mathcal{C}\,\lambda k\,.\,kM \quad k \text{ does not occur free in } M \\ \mathcal{C}\,\lambda k\,.\,P[k\mathcal{C}M] &\simeq \mathcal{C}\,\lambda k\,.\,P[Mk]\,. \end{aligned}$$

In the second equation, $P$ is any $\Lambda_c$-context that does not bind $k$, and $M$ is any $\Lambda_c$-term. We may consider the equations transcribe those of Eilenberg-Moore algebras. In Set with the result set $R$, and on an algebra $(X,\theta)$, the equations are

$$\begin{aligned} x &= \theta\,\mathrm{ev}_x \\ \theta\lambda k.\Phi(k\circ\theta) &= \theta\lambda k.\Phi(\mathrm{ev}_k) \end{aligned}$$

where $x \in X$, $\mathrm{ev}_x(k) = kx$ for any $k \in R^X$, the $k$ in the multiplication equation ranges in $R^X$, and $\Phi$ is any function $R^{R^X} \to R$.

As our purpose is to formulate the operational equations for the sake of motivation, we shall not get into further details, i.e. to formulate the correct notions of congruence, or to attempt proofs of the equations for each operational description. However, some cases are easy to check. For example, the unit law can be easily verified for continuation-passing style descriptions, for both call-by-value and call-by-name. Also, taking special cases of $M$ and $\Phi$, the equations may be verified for the CK-machine and deterministic reductions as well.

The operational equations appear in literature in some way, although not in the view of Eilenberg-Moore algebras.

1. Griffin gives typing rules for $\mathcal{C}$ and $\mathcal{A}$ [25]. He notices that with types taken into account, the reduction rule for $\mathcal{C}$ needs to be wrapped in a context $\mathcal{C}\,\lambda k.[\;]$. He writes (§3, before the definition of the reduction rules $\mapsto_t$ and Definition 1),

   > Instead of evaluating an expression $M^\alpha$ with the $\mapsto_u$ rules [The untyped reduction rules], the expression $\mathcal{C}(\lambda k^{\neg\alpha}.kM)$ is evaluated with the rules of $\mapsto_u$ being applied only inside of the expression $\mathcal{C}(\lambda k.\cdots)$. The rules now make "type sense" since the body of the $\lambda$-expression [refering to $kM$] is of type $\perp$.

   This could be seen as an implicit use of the unit law, i.e. the first operational equation.

2. Felleisen et al. seems also interested in equality related to the second equation, but for the relation between the rewriting semantics and reductions. They say ([20, p231], following Corollary 4.9),

   > Informally, these results mean that the $C$-machine is characterised by a standard computation function (and sequence) of a calculus modulo some syntactic difference. In order to eliminate the difference, we would have to change the standard reduction function [Definition 3.9, pp220–221] in such a way that a term $K(\mathcal{C}M)$ evaluates to $MK$ for a continuation $K$.

**Typing $\mathcal{C}$** The language $\Lambda_c$ is introduced as an untyped language by Felleisen et al.. Griffin gives typing rules for a subset of the language, and the typing rules for $\mathcal{C}$ and $\mathcal{A}$ correspond to the 'reductio ad absurdum' and 'ex falso' rules of classical logic [25]. Although the observation is remarkable, we shall interpret it with care:

1. Semantic understanding, e.g. from continuation-passing style translations, shows that the corresponding logic shall be viewed as the result of the negative translation. We shall see that in more detail below.

2. Once we have the typing rules, we shall see that we can derive $A$ and $\neg\neg A$ from each other; the term constructions are $\lambda k^{\neg A}.k[\cdot]$ and $\mathcal{C}$ respectively. However, the derivations are not reversible. In particular, the operational equation
$$\lambda k^{\neg A}.k\mathcal{C}M \simeq M$$
is the one in trouble, e.g. in the call-by-value setting, with the continuation-passing style translation. We shall not get into the details.

The typing of $\mathcal{C}$ is as follows.

(i) We assume a type constant $\bot$, and there is no constant of this type. We abbreviate $A \to \bot$ by $\neg A$. (Taking $\bot$ as constant may be too weak for our intention, but it is the original formulation. We shall be content with it and comment later.)

(ii) The typing rule of $\mathcal{C}$ is
$$\frac{\Gamma \vdash M\colon \neg\neg A}{\Gamma \vdash \mathcal{C}M\colon A}\,.$$

We refer to Griffin's analysis for the reasoning that leads to the rule [25, §3]. We shall not characterise the typable sublanguage according to the rule, only to note that some terms, like $\mathcal{C}\lambda k.k$, are untypable under such typing rules, yet they can still be useful [46].

**Semantics** With the typing rule added, we can type the operational descriptions. We shall only consider the continuation-passing style translations, as the typing of them is more informative. Although the translations are standard [36, 50, 56], we still give enough detail for convenience. The translations are as follows.

1. For call-by-value, we translate to simply-typed $\lambda$-calculus; for call-by-name, product types are also needed.

2. We define translations of types. For call-by-value, we define inductively, value translations $\llbracket\cdot\rrbracket_v$, negative translations $\llbracket\cdot\rrbracket_n$ and computation translations $\llbracket\cdot\rrbracket$; for call-by-name, we define negative translations $\llbracket\cdot\rrbracket_n$ and computation translations $\llbracket\cdot\rrbracket$. In both cases, the negative translation $\llbracket A\rrbracket_n$ is the type of continuations from $A$. We list them in the following table.

| *Call-by-value* | *Call-by-name* |
|---|---|
| $\llbracket G\rrbracket_v = G$ | $\llbracket G\rrbracket_n = G \to \bot$ |
| $\llbracket A\rrbracket_n = \llbracket A\rrbracket_v \to \bot$ | |
| $\llbracket A\rrbracket = \llbracket A\rrbracket_n \to \bot$ | $\llbracket A\rrbracket = \llbracket A\rrbracket_n \to \bot$ |
| $\llbracket A \to B\rrbracket_v = \llbracket A\rrbracket_v \to \llbracket B\rrbracket$ | $\llbracket A \to B\rrbracket_n = \llbracket A\rrbracket \times \llbracket B\rrbracket_n$ |

3. Then, for a judgement

   $$\Gamma \vdash M\colon A,$$

   call-by-value translates a free variable $x_i\colon A_i$ in $\Gamma$ by the value translation, into $x_i\colon [\![A_i]\!]_v$; the conclusion $M\colon A$ is translated as a computation, into $[\![M]\!]\colon [\![A]\!]$. Call-by-name translates all types by the computation translation.

4. Translations of terms are suitably typed versions of the untyped formulations. It may take some calculation to work out the details, e.g. translations of $[\![\neg\neg A]\!]$, the type of $\mathrm{ev}_{[k],\mathbf{I}}$ and the type of $\langle [k], \mathbf{I}\rangle$, but they are all straightforward.

(*End of translation*)

With the translations formulated, we are also able to explain the connection of $\mathcal{C}$ and the structural map of an Eilenberg-Moore algebra. This seems not in the literature, but it is along the line of our narrative. Suppose we work in some cartesian-closed category, say Set to be specific, to model the targeting $\lambda$-calculus in the continuation-passing style translations. We also choose a set $R$ to interpret $\perp$ in the target language. Then the operator $\mathcal{C}$ can be interpreted as a function $[\![\neg\neg A]\!] \to [\![A]\!]$, whose detail depends on call-by-value and call-by-name, and can be read out from the translations. This typing suggests we may consider possible connections with the structural map of an Eilenberg-Moore algebra. This turns out to be correct, and we only need to factor out the part depending on call-by-value or call-by-name. We shall omit the details, but it is a fact that each computation interpretation $[\![A]\!]$ carries an Eilenberg-Moore algebra, say with the structural map $\theta_A\colon R^{R^{[\![A]\!]}} \to [\![A]\!]$.[10] Then the interpretation of $\mathcal{C}$ can be factored into two parts:

$$[\![\neg\neg A]\!] \longrightarrow R^{R^{[\![A]\!]}} \xrightarrow{\theta_A} [\![A]\!]. \tag{2.4}$$

The first part is a natural transformation depending on call-by-value and call-by-name, and the second part is the structural map. We may come to the following view: the core part of $\mathcal{C}$ is interpreted by the structural map, and the rest is reorganising data.

**Summary** In the above, we introduce the operator $\mathcal{C}$, from its operational descriptions, to operational equations that hold or are expected to hold, and to typing. We note that

1. We expect operational equations that correspond to the Eilenberg-Moore algebra equations to hold. Conversely, we may consider reading the structural map as a control operator. We have not given the argument for the latter, but we shall do so in the explanation of monadicity.

2. We may consider the structural map as the essential content of $\mathcal{C}$.

We can now give a critical comment on $\mathcal{C}$. It is an economic formalism, as only one new term constructor is needed in the language. Compared to the operator $\mu$, which is to be considered later, the interpretation of $\mathcal{C}$ varies between call-by-value and call-by-name. This could be considered as a defect.

**The type $\perp$ as a type variable** In the typing part, we mention that taking $\perp$ as a type constant may be too weak for our intention. An alternative is to consider $\perp$ as a type variable. We may follow Griffin, and consider the evaluation rule

$$D[\mathcal{C}M] \;\mapsto\; M\,\mathrm{coer}D\;.$$

[10] This algebraic viewpoint is due to Levy's call-by-push-value [56].

Here the context $D$ may vary, so the type of $D[\mathcal{C}M]$ and '$M\,\mathrm{coer}D$' may vary as well. Thus when we say the type of $M$ is $(A \to \bot) \to \bot$, we intend to say an open type expression $(A \to R) \to R$, where $R$ is a free type variable. It may be worth noting that the operator $\mu$, the operator that we shall look into next, was originally introduced in a second-order setting. Nevertheless, we are deviating from the main narrative and shall stop here.

## 2.2 The operator $\mu$

We have pointed out a theoretical shortcoming of $\mathcal{C}$, namely its interpretation involves a varying part depending on call-by-value or call-by-name. Technically, the immediate cause of this problem is its explicit inclusion of the type $\bot$. However, the real tension is that we intend to describe continuations with indefinite ends, but actually formulate them with definite ends, denoted by $\bot$. Tracing the problem further, we see that the source of the problem is that we hope to stay within facilities provided by the existing language, only adding a new control operator. The consequence is that continuations are presented as functions in the source language, rather than given their own place. Thus economic formulation, a merit of $\mathcal{C}$, is also the source of its problem. In the following, we formulate the control operator $\mu$. We shall introduce different variables for continuations, and make some room for continuations with indefinite ends. With the above fundamental issues sorted out, we may describe the operator $\mu$ as the result of combining $\mathcal{C}\lambda$, i.e., combining the control operator $\mathcal{C}$ with a binder of continuations.

The $\mu$ operator is introduced by Parigot, as a term calculus for the natural deduction of second-order classical logic [31]. In the literature, the calculus is usually referred to by its first-order part. One significant difference from Felleisen et al.'s $\mathcal{C}$ is that the language distinguishes control variables that hold continuations. The operator is later studied by many, including Hofmann and Streicher [36], de Groote [35], Crolard [44], Selinger [50] among others. With some modification, de Groote compares it to a variant of $\mathcal{C}$, and Crolard relates it with the operator catch (op. cit. for both).

**Operational descriptions of $\mu$** The language of Parigot's $\lambda\mu$-calculus is as follows

**Definition** (Parigot [31], with a simplification in notation)**.** The language of $\lambda\mu$-calculus makes use of two sets of variables: $\lambda$-variables ($x$, $y$, ...) and $\mu$-variables ($\alpha$, $\beta$, ...). The $\lambda\mu$-terms are, by a mutual induction,

(i) *Unnamed terms*

$$x \quad MN \quad \lambda x.M \quad \mu\alpha.Q$$

where $x$ is any $\lambda$-variable, $M$, $N$ are any unnamed $\lambda\mu$-terms, $\alpha$ is any $\mu$-variable, the symbol $\mu$ binds $\alpha$, and $Q$ is any named $\lambda\mu$-term.

(ii) *Named terms*

$$\alpha M$$

where $\alpha$ is any $\mu$-variable, and $M$ is any unnamed $\lambda\mu$-term. We say $\alpha$ names the remainder of $M$.

We also call $\lambda$-variables *ordinary variables* and $\mu$-variables *control variables.*

By definition, terms starting with $\mu$ can only be in the forms $\mu\alpha.\alpha M$ and $\mu\alpha.\beta M$. Crolard uses those terms directly in the definition, combining the $\mu$ construction and the naming construction into two constructions $\mu\alpha.\alpha M$ and $\mu\alpha.\beta M$ [44]. Separating the two constructions makes operational descriptions easier, and the language remains the same so long as we retain the restrictions on named and unnamed terms in term constructions. The change in notation is writing $\alpha M$ instead of $[\alpha]M$. We shall explain more in the typing part.

Regardless of the formalism, and regardless of call-by-value and call-by-name, the operational intention of $\mu$ and naming are as follows.

a. Suppose we would like to evaluate $\mu\alpha.M$, with current continuation $k$. Then the next step is to evaluate $M[k/\alpha]$ with the trivial continuation. In a formal description, we shall extend the language to incorporate continuations.

b. Eventually, we may come to evaluate $kM$ with current continuation $h$, where $k$ is a continuation. Then the next step is to evaluate $M$ with the continuation $k$.

We put the intention into the same formalisms as in the case of $\mathcal{C}$.

($a$) In the CK-machine formalisms, the transitions for $\mu$ and naming are

$$\begin{aligned} \mu\alpha.M,\, k &\rightsquigarrow M[k/\alpha],\, \mathrm{nil} \qquad (2.5) \\ kM,\, h &\rightsquigarrow M,\, k \qquad (2.6) \end{aligned}$$

where $k$, $h$ are stacks, and nil is the empty stack. We shall extend the language to incorporate stacks. The rules are simpler than in the $\mathcal{C}$ case, because we explicitly declare variables bound to continuations. The transition rules are adapted from Parigot's comments [31, the ending comment §3.5.2] and Crolard's description [44].

($b$) In the deterministic reduction formalism, the reduction rules are

$$\begin{aligned} C[\mu\alpha.M] &\mapsto M[\mathbf{p}\,C/\alpha] \\ C[\mathbf{p}DM] &\mapsto D[M] \end{aligned}$$

where $C$, $D$ are suitable evaluation contexts, and $\mathbf{p}\,C$ encodes the context $C$ to be used as a term.

($c$) In the continuation-passing style formalism, the translation clauses are

$$\begin{aligned} [\![\mu\alpha.M]\!]k &= \big(\lambda\alpha.[\![M]\!]\big)k \qquad (2.7) \\ [\![\alpha M]\!] &= [\![M]\!]\alpha \qquad (2.8) \end{aligned}$$

where the target language is $\lambda$-calculus, $k$ is a free variable, and $\alpha$ in the target language is an ordinary variable. The translation is the same for both call-by-value and call-by-name. This is an improvement over the $\mathcal{C}$ case, essentially due to the distinction of control variables. The translation is adapted from Selinger's with a change [50, §6.1, §7.1]. Here we do not need the identity combinator **I**. We shall see in the typing part that we leave out the type $\perp$ in the source language, and introduce indefinite judgements instead. This gives us a better typing where no **I** is needed.

**Equations of $\mu$** As in the case of the $\mathcal{C}$ operator, we need a congruence. Let $\simeq$ be a suitable congruence derived from the operational descriptions. We expect the following equations to hold, regardless of formalisms, for both call-by-value and call-by-name:

$$\begin{aligned} M &\simeq \mu\alpha\,.\,\alpha M \qquad \alpha \text{ does not occur free in } M \\ \mu\alpha\,.\,P[\alpha\,\mu\beta\,.\,Q] &\simeq \mu\alpha\,.\,P\big[Q[\alpha/\beta]\big]\,. \end{aligned}$$

In the second equation, $P$ is a context that does not bind $\alpha$, and $P[N]$ is named if $N$ is so, $\beta$ is any control variable that is not $\alpha$, and $Q$ is any named term. We can verify that the first equation holds using the CK-machine formalism and continuation-passing style translations. For the second equation, we can at least verify that the equation holds for some context $P$, e.g. when $P$ is the trivial context.

As in the case of $\mathcal{C}$, we may consider the operational equations correspond to the Eilenberg-Moore equations

$$\begin{aligned} x &= \theta\,\mathrm{ev}_x \\ \theta\lambda k.\Phi(k \circ \theta) &= \theta\lambda k.\Phi\,\mathrm{ev}_k \end{aligned}$$

for an algebra $(X, \theta)$, where $x \in X$, $\Phi$ is any function in $R^{R^{R^{R^X}}}$, and $k$ ranges over $R^X$. Informally, we may consider $\mu$ in the operational equations correspond to $\theta\lambda$ in the Eilenberg-Moore equations. For the first equations, the correspondence should be obvious. For the second equations, we may see the correspondence in special cases. For example, we may take $P$ as the trivial context in the operational equation, and $\Phi$ as $\mathrm{ev}_\psi$ in the Eilenberg-Moore equation, where $\psi$ is some function in $R^{R^X}$. Then the second equations become

$$\begin{aligned} \mu\alpha\,.\,\alpha\,\mu\beta\,.\,Q &\simeq \mu\alpha\,.\,Q[\alpha/\beta] \\ \theta\,\lambda k\,.\,k(\theta\lambda h\,.\,\psi h) &= \theta\,\lambda k\,.\,\psi k\ . \end{aligned}$$

The correspondence should be obvious in this case. In view of the correspondence, we shall also call the operational equations the unit law and the multiplication law, respectively.

The two operational equations occur in literature in some way.

1. The unit law appears in the literature as we present here, and is usually viewed as the $\eta$-law for $\mu$. For example, it appears as Parigot's additional reduction rule [31, §3.1 Remark], as Crolard's reduction ruled called the 'simplification rule' [44, §2], as Hofmann and Streicher's $\mu$-$\eta$ law [36, Def. 2.1] and similarly Selinger's $\eta_\mu$ axiom [50, §6.4, §7.4]. As a side point, we consider the $\eta$-law as an expansion, i.e. from $M$ to $\lambda x.Mx$, and from $M$ to $\lambda\alpha.\alpha M$. This is in analogue with the unit of adjoint functors: the right adjoint takes the role of the introduction rule, the left adjoint for the elimination rule, the unit for the $\eta$-law, and the counit for the $\beta$-law. For this reason, we have been consistently writing the operational equations and Eilenberg-Moore equations with such a direction in mind.

2. The multiplication law does not appear in the literature in the form we present. The essence of the law is the substitution of $Q[\alpha/\beta]$ for $\alpha\mu\beta.Q$, and the substitution appears as a reduction rule or an equation. This essential equation is often viewed as the $\beta$-law for $\mu$. For example, it appears as Parigot's renaming rule (§3.1), so as in Crolard's reformulatioin; it appears as Hofmann and Streicher's $\mu$-$\beta$ law, and similarly Selinger's $\beta_\mu$ axiom (see the unit law case for citations).

**Typing $\mu$** We type $\mu$ below. Informally, the feature of such typing is that, continuations are of types of a different nature; a continuation from a type $A$ should, be of a type dual to $A$. We have been very careful by putting 'informally' in the previous sentence. That is because, at least in the literature, dual types are only meta-theoretical; they are not explicitly formulated. In other words, dual types do not exist in the syntax. (This is the norm in the literature; we shall list some in comments.) The dual nature of continuations is only suggested by typing rules (of terms). The reader may pay attention to that peculiarity below, and we shall comment on it after the definition.

**Definition.** The language of typed $\lambda\mu$-calculus is defined as follows.

(i) Types are as before. They are, inductively,

$$G \quad A \to B$$

where $G$ is any ground type, and $A$, $B$ are types of the language.

(ii) There are two disjoint sets of variables: *ordinary variables* and *control variables.* They are typed. We name ordinary variables by $x$, $y$, $\ldots$, and control variables by $\alpha$, $\beta$, $\ldots$.

[Control variables are intended as variables of continuations. As we have hinted at, for control variables, $\alpha\colon A$ should read: the control variable $\alpha$ is of type $A$, dually; or, $\alpha$ is a continuation from $A$.[11] Obviously the word 'dually' needs to be supported by the type theory. This is done in judgements (iv), where control variables are given a separate context, and in typing rules of terms (v). Readers shall continue with the definition and see comments that follow.]

(iii) Accordingly, there are two kinds of contexts. An *ordinary context* is a finite list of ordinary variables, i.e. $x_1\colon B_1, \ldots, x_n\colon B_n$. A *dual context* is a finite list of control variables, i.e. $\alpha_1\colon A_1, \ldots, \alpha_m\colon A_m$.

(iv) There are two kinds of judgements: definite and indefinite. *Definite judgements* are in the form

$$\Gamma\,;\Delta \vdash M\colon C$$

where $\Gamma$ is an ordinary context, and $\Delta$ is a dual context, $M$ is the term of the judgement, and $C$ is an ordinary type. *Indefinite judgements* are in the form

$$\Gamma\,;\Delta \vdash M$$

where all components are the same except for the omission of the conclusion type $C$.

[The intended reading of the indefinite judgement is, assuming ordinary variables $\Gamma$ and control variables $\Delta$, we have term $M$, of some type we do not mention.]

(v) Terms are constructed as follows. Rules from $\lambda$-calculus are kept and suitably adapted: judgements involved are each expanded with a dual context, but the dual context is not used by the rules; also, terms involved in the rules are now terms of the new language. New rules are rules for $\mu$ and naming. These new rules use the dual context only; we list them below.

$$\text{(naming)} \quad \frac{\Gamma\,;\Delta \vdash M\colon A}{\Gamma\,;\Delta \vdash \alpha M} \quad \alpha\colon A \text{ in } \Delta$$

$$(\mu) \quad \frac{\Gamma\,;\Delta_1\,,\,\alpha\colon A,\,\Delta_2 \vdash M}{\Gamma\,;\Delta_1\,,\Delta_2 \vdash \mu\alpha^A\,.\,M\colon A}\,.$$

[The naming rule can be compared with the application rule in $\lambda$-calculus; the control variable $\alpha$ takes the role of a function. We may read $\alpha M$ by '$\alpha$ names the remainder of $M$', or 'apply $\alpha$ to $M$'. The $\mu$ rule can be compared with the abstraction rule in $\lambda$-calculus; $\mu$ is the binder, taking the role of $\lambda$. In the binding $\mu\alpha^A$, we have decorated the control variable $\alpha$ with its type $A$. As before, we shall say, $\alpha$ is dually of type $A$.]

[11] I feel reading the word 'dually' explicitly is quite important, as least for new comers. From a logical point of view, we may also say 'negatively' in place of 'dually'.

(*End of definition*)

Some comments follow.

1. As we have said, for control variables, the intended reading of $\alpha\colon A$ is, '$\alpha$ is dually of type $A$'. We may say that the typing colon (:) here is different from those typing colons of ordinary variables.[12] Historically, it took some steps to evolve into the formulation above; we leave that to the next comment and the comment 6. Here, let us see that the dual reading (or 'negative reading' from the logical perspective) is consistent with the type theory. Firstly, it is possible to make a distinction between the two kinds of colons, because control variables are given a separate context. Then we consider typing rules for naming and $\mu$. To make the argument easier, let us consider the simplest possible instances, as shown below (the two rules to the left). In such instances, ordinary contexts are empty, and the dual context is either empty or only contains the relevant control variable.

$$\begin{array}{ccc} \text{(naming)} & \dfrac{;\alpha\colon A\vdash M\colon A}{;\alpha\colon A\vdash \alpha M} & \dfrac{;A\vdash A}{;A\vdash} \\[2ex] (\mu) & \dfrac{;\alpha\colon A\vdash M}{;\vdash \mu\alpha^{A}.M\colon A} & \dfrac{;A\vdash}{;\vdash A}\ . \end{array}$$

To each rule, we also accompany it with a further abridged rule to the right: variables and terms are left out, leaving only types. In the abridged rules, judgements may be considered as sequents, and the rules may be considered as derivation rules of those sequents. Here the consequence of each sequent is assumed to have at most one proposition (when there is no proposition in the consequence, we consider the consequence to be falsum). Our bottom line is simple: the derivation rules should be valid. If we understand propositions obtained from the dual context negatively, then the abridged naming rule is a rule for contradiction; and the abridged $\mu$ rule is 'reductio ad absurdum'. Apparently, with the same 'types as propositions' analogy, there is no way we can read those propositions positively.[13]

2. From a logical perspective, it is rather unusual that we think of negations of propositions but choose to leave the negations to the ambient structure; yet that is the case here (for formulations in the literature, see e.g., Hofmann and Streicher [36], and Selinger [50]). To see how that comes to be, it may be helpful to take a historical perspective. The origin of the convention is in Parigot's original work on $\mu$ [31], where the typing appears as a term calculus for natural deductions. There, sequents are in the form

$$A_1,\ldots,A_n\vdash C\,;B_1,\ldots,B_m.$$

On the consequence side, the semicolon and commas are all considered as disjunctions as usual. As a term calculus, $C$ is the proposition that the proof term is aiming at, and $B_i$ are assumptions, witnessed by control variables. It is not hard to see the tension here: if we understand the consequence as a big disjunction, then the consequence should consists of only one proof term, without the addition of any variables; moreover, the proof term should

[12] We may call such a colon 'the inhabitant-of predicate'; but a less formal name seems more convenient for both the writer and the reader.

[13] Readers should beware that if the dual context consists of multiple types, say '$A, B$', then it is to be read as 'not $A$ and not $B$', rather than 'the negation of $A$ and $B$'. If the reader have doubts about the typing itself, and not just the reading (such doubts are understandable), they can be assured by considering the operational description. The typing here allows a decoration of types for the operational description we have introduced earlier.

aim at the following statement: 'it is either $C$, or $B_1, \dots,$ or $B_m$'. Apparently that is not the case (and we agree with Parigot's intention for the term calculus). To phrase it differently: in the sequents presentation, $B_i$ are part of the consequence, and they are understood positively; whereas in the term calculus presentation, $B_i$ are part of the premises, and they are understood negatively.[14] In later study, some authors move control variables to the premises side (as we follow here), while keeping the negative reading implicit (see the same papers [36, 50] as mentioned above). Thus it shifts the tension between the sequent calculus and the term calculus, as in Parigot's work, to the tension within the sequent calculus itself (i.e., implicit negations).

3. Despite being unusual from a logical point of view, leaving negations to the ambient structure can be justified by categorical semantics, at least in the call-by-name case. Quoting Hofmann and Streicher almost word-by-word, the crucial observation is that '$R^A \times B$ is an exponential of $B$ by $A$ in the category $\mathbf{C}_R$ which has the same object as $\mathbf{C}$ and homsets given by $\mathbf{C}_R(X, Y) = \mathbf{C}(R^X, R^Y)$' ([36], near the end of §3, before Theorem 3.3). Thus the dual context may be interpreted in the category $\mathbf{C}_R$, using the notation of Hofmann and Streicher. We shall not deviate into a discussion of semantics here, but refer the reader to the original paper. Selinger [50] also explains categorical semantics, for both call-by-value and call-by-name. Führmann and Thielecke [59] also consider relevant categorical structures, but they work in the context of Felleisen et al.'s $\mathcal{C}$ operator, call-by-value.

4. An alternative to the implicit negations is to make them all explicit. This is not a popular idea, and we shall not look any further for the purpose of exposition.[15]

5. It is easy to see that the definiteness of judgements formulates the same restrictions on the language as in the untyped case, where we distinguish named and unnamed terms. Thus, if we erase typing information from typed $\lambda\mu$-terms, then we get back the same language of the untyped $\lambda\mu$-calculus. The indefinite judgements we use here is inspired by Levy's non-returning command judgements [56, §7.2].

6. Hofmann and Streicher, following Ong, introduces a type constant $\bot$ in their variant [36]. There, ordinary types are the only types defined explicitly, and with the addition of $\bot$, definite judgements are the only judgements. As a consequence, the immediate subterm of $\mu$ need not be named, e.g. $\mu\alpha.(\lambda x.\beta M)x$. Thus the restriction on named and unnamed terms is relaxed, and the variant is different from Parigot's original one. Crolard notes this in the study of catch-throw operators [44]. His comment that Hofmann and Streicher's variant 'does not define catch and throw' should probably be understood as 'there is more in that variant than catch and throw'.

[14] It is conceivable that Parigot might have considered putting the dual context to the left of the turnstile sign ($\vdash$), thus making the relevant propositions assumptions; but then he would see the unusual implicit negation we have just explained. Then, to make the sequents and their derivations more conventional from the logical perspective, it seemed natural to lean towards his actual choice.

[15] Some researchers don't like this idea, probably partly because it does not survive Occam's razor, and partly because it oversteps, ruling out such categorical abstraction given by Hofmann and Streicher (cited above). Nevertheless, it may be useful to mention Crolard's work here. Crolard has written about subtractive logic and its connection with $\mu$-calculus [58]. Using subtraction types, continuations would be given subtraction types like $\top - A$. For this particular case (typing $\mu$), my personal preference of the terminology is 'logarithm types'. My argument is that by monadicity, a computation type can be interpreted as an exponential, where the exponent is the interpretation of the dual type. I beg the reader for excuse if the last few sentences seem rather mysterious.

**Reversible derivations** With the typing above, we observe reversible derivations

$$\beta[\cdot] \downarrow \frac{\Gamma \,;\, \Delta_1 \,,\; \Delta_2 \vdash A}{\Gamma \,;\, \Delta_1 \,, A,\; \Delta_2 \vdash} \uparrow \mu\alpha.[\cdot]\ .$$

They are given by the operational equations

$$M \simeq \mu\alpha.\alpha M \qquad (\eta\text{-law})$$

$$\beta\,\mu\alpha.M \simeq M[\beta/\alpha]\ , \qquad \text{(renaming)}$$

with the usual restriction that $\beta$ does not occur free in $M$. This is an improvement over the $\mathcal{C}$ case. The reason is that continuations are properly typed, and the $\bot$ type is left implicit by the introduction of indefinite judgements.

**Semantics** We shall start from continuation-passing style translations.

1. The target language is the same as in the $\mathcal{C}$ case, i.e., we fix a type constant $R$; for call-by-value, we translate to simply-typed $\lambda$-calculus; for call-by-name, product types are also needed.

2. We translate ordinary types as in the $\mathcal{C}$ case (page 32). Additionally, we interpret types in the dual context by negative interpretations $[\![A]\!]_n$. A useful equality to keep in mind is that for both call-by-value and call-by-name, the computation interpretation $[\![A]\!]$ is defined as $[\![A]\!]_n \to R$ in the target language.

3. Ordinary contexts are translated as in the $\mathcal{C}$ case, i.e., the call-by-value translation of an ordinary context $x_1\colon B_1, \ldots, x_n\colon B_n$ is a context of value translations $x_1\colon [\![B_1]\!]_v, \ldots, x_n\colon [\![B_n]\!]_v$. The call-by-name translation of the same ordinary context is a context of computation translations $x_1\colon [\![B_1]\!], \ldots, x_n\colon [\![B_n]\!]$. A dual context $\alpha_1\colon A_1, \ldots, \alpha_m\colon A_m$ is translated as a context $\alpha_1\colon [\![A_1]\!]_n, \ldots, \alpha_n\colon [\![A_m]\!]_n$.

4. For both definite and indefinite judgements, a context $\Gamma \,;\, \Delta$ is translated as $[\![\Gamma]\!], [\![\Delta]\!]$, the concatenation of the two translated contexts. For both call-by-value and call-by-name, a definite judgement $\Gamma \,;\, \Delta \vdash M\colon A$ is translated as a judgement

   $$[\![\Gamma]\!], [\![\Delta]\!] \vdash [\![M]\!]\colon [\![A]\!].$$

   That is, the concluding ordinary type is translated with the corresponding computation interpretation. For both call-by-value and call-by-name, an indefinite judgement $\Gamma ; \Delta \vdash M$ is translated as a judgement

   $$[\![\Gamma]\!], [\![\Delta]\!] \vdash [\![M]\!]\colon R.$$

5. Translation of terms are as in the untyped case, page 35. For

   $$[\![\mu\alpha^A.M]\!]k = \big(\lambda\alpha^{[\![A]\!]_n}.[\![M]\!]\big)k,$$

   the variable $k$ is of type $[\![A]\!]_n$. Recall that the intended meaning of $[\![A]\!]_n$ is the type of continuations from $A$. In

   $$[\![\alpha M]\!] = [\![M]\!]\alpha,$$

   suppose that $M$ is of type $A$, so that $\alpha$ is dually of type $A$, and $\alpha M$ is a term of an indefinite judgement. Then on the right hand side, $[\![M]\!]$ is of type $[\![A]\!] = [\![A]\!]_n \to R$, and $\alpha$ is of type $[\![A]\!]_n$.

(*End of translations*)

From the continuation-passing style translations, we can derive interpretations of the language into any cartesian-closed category. Let $\mathcal{V}$ be the cartesian-closed category of choice. Then modulo contexts, the control operator $\mu_A$ that binds continuations from $A$ is interpreted as the currying

$$\mathcal{V}(\llbracket A \rrbracket_n, R) \to \mathcal{V}(1, R^{\llbracket A \rrbracket_n}) = \mathcal{V}(1, \llbracket A \rrbracket).$$

Since $\llbracket A \rrbracket = R^{\llbracket A \rrbracket_n}$, we know that it carries an algebra of the continuation monad (with parameter object $R$). It has the structure map

$$\theta_A = R^{\mathrm{ev}_{\llbracket A \rrbracket_n}} : R^{R^{R^{\llbracket A \rrbracket_n}}} \to R^{\llbracket A \rrbracket_n},$$

Here $\mathrm{ev}_{\llbracket A \rrbracket_n}$ is the obvious evaluation morphism. Then, the connection of $\mu$ and the structure map may be formulated by the following commutative diagram:

$$\begin{tikzcd}
\mathcal{V}(R^{\llbracket A \rrbracket}, R) \arrow[r, "\text{curry}"] \arrow[d, "{\mathrm{ev}_{\llbracket A \rrbracket_n}, -}"'] & \mathcal{V}(1, R^{R^{\llbracket A \rrbracket}}) \arrow[d, "{-, \theta_A = R^{\mathrm{ev}_{\llbracket A \rrbracket_n}}}"] \\
\mathcal{V}(\llbracket A \rrbracket_n, R) \arrow[r, "\llbracket \mu \rrbracket"] & \mathcal{V}(1, \llbracket A \rrbracket) = \mathcal{V}(1, R^{\llbracket A \rrbracket_n}) .
\end{tikzcd} \tag{2.9}$$

**Summary** By properly typing continuations and leaving the $\perp$ type implicit, the $\lambda\mu$-calculus avoids issues we have seen with $\mathcal{C}$. It has reversible derivations as expected, and the meaning of $\mu$ is the same under call-by-value and call-by-name interpretations. (We shall emphasise that the $\mu$ operator was not formulated to improve on $\mathcal{C}$. It has its own logical origin, and we have mentioned that origin in the introduction to the operator, p34.) We have also compared $\mu$ with $\mathcal{C}$ to bring out the connection to the Eilenberg-Moore structure map.

Elegant as it is, the calculus is not very convenient for our analysis. Namely, it is not easy to extract a suitable definition of algebras from operational equations of $\mu$. For example, merely interpreting $\mu$ and the operational equations won't give us exactly the defining equations of Eilenberg-Moore algebras (the reader may want to give it a try). It is probably for this reason that $\mu$ is not often studied together with the structural map of the continuation monad. This is also part of the reason that formulating algebras for *letstk* and *changestk* is not straightforward: like the $\mu$ case just described, transcribing the operational equations won't give us an immediate answer (the reader may also want to give it a try). Nevertheless, the result from such a transcription may still serve as a good hint.

What will be a convenient language for our analysis, such that we can read from it a definition of algebras? We need $\mathcal{C}$ but with one adaptation: we shall leave the $\perp$ type implicit, and use indefinite judgements. Unavoidably, we lose the reversible derivations. We shall not formulate such a language explicitly here, but we will approach the operation-algebra connection from a different direction: Instead of seeing the Eilenberg-Moore equations from operational equations, we shall read the Eilenberg-Moore equations operationally. In the following, we shall first summarise the connection of $\mathcal{C}$ and $\mu$ with the structure map, then try an operational reading of the Eilenberg-Moore equations.

## 2.3 $\mathcal{C}$, $\mu$ and the Eilenberg-Moore structure map

**Relating $\mathcal{C}$ and $\mu$ to the Eilenberg-Moore structure map** From previous analysis of $\mathcal{C}$ and $\mu$, we see that both operators relate closely to the structure map of an Eilenberg-Moore algebra. We shall summarise the relevant equations and interpretations below, to emphasise that close connection.

*Equations* In the case of $\mathcal{C}$, we consider the following operational equations:

$$\begin{aligned} M &\simeq \mathcal{C}\lambda k\,.\,kM \qquad k \text{ does not occur free in } M \\ \mathcal{C}\lambda k\,.\,P[k\mathcal{C}M] &\simeq \mathcal{C}\lambda k\,.\,P[Mk]\,. \end{aligned} \tag{2.10}$$

In both call-by-value and call-by-name, we know that the first equation holds, and the second holds at least for a trivial context $P$. As for $\mu$, we consider equations

$$\begin{aligned} M &\simeq \mu\alpha\,.\,\alpha M \qquad \alpha \text{ does not occur free in } M \\ \mu\alpha\,.\,P[\alpha\,\mu\beta\,.\,Q] &\simeq \mu\alpha\,.\,P\big[Q[\alpha/\beta]\big]\,. \end{aligned} \tag{2.11}$$

Similarly, the first equation holds, and the second holds at least for a trivial context $P$. In the category Set, for an Eilenberg-Moore algebra $(X, \theta)$, the defining equations are

$$x = \theta\,\mathrm{ev}_x \tag{2.12}$$

$$\theta\,\lambda k\,.\,\Phi(k \circ \theta) = \theta\,\lambda k\,.\,\Phi\,\mathrm{ev}_k \tag{2.13}$$

where $x \in X$, $\Phi$ is any function in $R^{R^{R^{R^X}}}$, and $k$ ranges over $R^X$.

The equations are similar, but we do not expect the reader to grasp that at one glance. For all cases, in the second equation, the two sides of the equality have much in common. Within that common part, there is an expression that does not play much role in the equation: the contexts $P$ in the cases of $\mathcal{C}$ and $\mu$, and the function $\Phi$ in the Eilenberg-Moore equations. Those repetitive parts obscure the equations. As an attempt to reduce the noise, we introduce the following abbreviations. We shall abbreviate (2.10) as follows: under $\mathcal{C}\lambda k$,

$$k\,\mathcal{C}M \simeq Mk \quad \text{for any } M. \tag{2.14}$$

Similarly, we abbreviate (2.11): under $\mu\alpha$,

$$\alpha\,\mu\beta\,.\,Q \simeq Q[\alpha/\beta] \quad \text{for any } Q \text{ and any control variable } \beta \neq \alpha. \tag{2.15}$$

In both abbreviations, we make an announcement of a common binding context ($\mathcal{C}\lambda k$, $\mu\alpha$), leave out some context in the middle (the context $P$ in both cases), and put the subterms under attention in one equation. We formulate that in a more formal manner below. The reader should keep in mind that this is after all a convenience of speech, and formality is never the purpose.

*Convention.* Let $C_x$ be a context that binds a variable $x$. Let $M$, $N$ be terms. We may say: under $C_x$, $M \simeq N$. By that we mean, for any suitable context $P$ that does not bind $x$,

$$C_x\big[P[M]\big] \simeq C_x\big[P[N]\big].$$

By a suitable $P$, we mean a context that makes the two sides are well-formed. We shall call $C_x$ the *binding context*, $P$ the *padding context*, and $M$, $N$ *terms under attention.*

Meticulous readers will notice that we also leave out the notation of the hole, i.e. $[\cdot]$. For example, we write 'under $\mathcal{C}\lambda k$' rather than 'under $\mathcal{C}\lambda k\,.\,[\cdot]$' etc.. This is because the place of the hole is obvious.

Similar to the above abbreviations, we may abbreviate equation (2.13) as follows: under $\theta\,\lambda k^{R^X}$,

$$k \circ \theta = \mathrm{ev}_k$$

or, under $\theta\lambda k^{R^X}$,

$$k(\theta M) = Mk \quad \text{for any } M \in R^{R^X}. \tag{2.16}$$

Thus, we make an announcement of the common context $\theta\lambda k^{R^X}$, leave out $\Phi$, and put the expressions we pay attention into an equation. For readers who feel compelled to see the detail, we include a formulation of the rule below. Other readers may want to skip this, or come back later. We will use the abbreviation again when we motivate equations of control algebras (page 50).

*Convention.* Let $X$, $R$, $B$, $Y$ be sets. Let $\theta$ be a function $B^{R^X} \to X$. Let $f_k$, $g_k \colon Y \to R$ be functions with parameter $k \in R^X$. We may say: under $\theta\lambda k^{R^X}$,

$$\begin{aligned} & f_k = g_k \\ \text{or} \quad & f_k M = g_k M \quad \text{for any } M \in Y. \end{aligned}$$

By that we mean: for any function $\Phi \colon R^Y \to B$,

$$\theta\lambda k^{R^X}.\Phi f_k = \theta\lambda k^{R^X}.\Phi g_k \;.$$

Although not in a syntactic setting, we shall borrow terminology from the previous convention. We call $\theta\lambda k^{R^X}$ the binding context, $\Phi$ the padding function, and the two sides in the abbreviated equation terms under attention.

In the case above, we have $B = R$ and $Y = R^{R^X}$. Thus $\theta$ is a function $R^{R^X} \to X$, $f_k$, $g_k$ are functions $R^{R^X} \to R$ and $\Phi$ is a function $R^{R^{R^X}} \to R$. The definitions of $f_k$ and $g_k$ are obvious.

With the abbreviations, similarity among (2.14), (2.15) and (2.16) should become obvious. For convenience, we shall list the equations together, in a form suitable for comparison. For the $\mathcal{C}$ case, the equations are

$$\begin{aligned} M &\simeq \mathcal{C}\lambda k.kM && k \text{ does not occur free in } M \\ k\mathcal{C}M &\simeq Mk && \text{under } \mathcal{C}\lambda k\text{, for any } M. \end{aligned}$$

For the $\mu$ case, they are

$$\begin{aligned} M &\simeq \mu\alpha.\alpha M && \alpha \text{ does not occur free in } M \\ \alpha\mu\beta.Q &\simeq Q[\alpha/\beta] && \text{under } \mu\alpha\text{, for any } Q. \end{aligned}$$

And for an Eilenberg-Moore algebra $(X, \theta)$,

$$\begin{aligned} x &= \theta\lambda k^{R^X}.kx \\ k(\theta M) &= Mk && \text{under } \theta\lambda k^{R^X}\text{, for any } M \in R^{R^X}. \end{aligned}$$

*Interpretations* The interpretation of $\mathcal{C}$ factors through the structure map (formula (2.4), p33). The interpretation of $\mu$ and the structure map can be compared in a commutative diagram (diagram (2.9), p41).

In view of such a close connection to the structural map, we propose to call both operators $\mu$ and $\mathcal{C}$ *Eilenberg-Moore control operators.* Apart from emphasising the connection to the structure map, this is also a convenient way to distinguish them from other control operators like call/cc-abort, and catch-throw.

A priori, there could be other control operators that are similarly related to the structural map, and we would also consider them as Eilenberg-Moore operators. However, it seems $\mu$ and $\mathcal{C}$ are all there can be. To put it more carefully: modulo variations that are non-operational in nature, the operational ideas presented by $\mu$ and $\mathcal{C}$ are all there can be for Eilenberg-Moore control operators. To justify such a claim we need a formal set-up, in particular a formal definition of Eilenberg-Moore control operators. I do not have such a set-up here (partly because it seems not obvious how to do that, and partly because that deviates from the main narrative of this work). Nevertheless, I can offer the following observation, which should serve as the key argument if we have the needed formality. First of all, by previous analysis, we shall come to the agreement that $\bot$ shall not appear in the language, that is, we shall have indefinite judgements. This gives us a variant of $\mathcal{C}$ (we have mentioned this variant to the end of the analysis of $\mu$, p41). Then the claim is that with indefinite judgements, any Eilenberg-Moore control operator is either $\mu$ or that variant of $\mathcal{C}$ with indefinite judgements. The place we shall look at, is how continuations are typed. More specifically, we shall consider

> Whether continuations are typed coarsely or precisely.

By *coarsely typed*, we mean continuations are typed as $[\![A]\!] \to R$ under continuation-passing style translations. By *precisely typed*, we mean continuations are typed as $[\![A]\!]_n$ under continuation-passing style translations. If we decide to type continuations coarsely, then we shall end up with a calculus with $\mathcal{C}$; if we decide to type continuations precisely, then that would be a calculus with $\mu$. Since we do not have the necessary formal set-up here, we shall not get into further details. However, motivated readers may want to experiment with the idea to see that this is the case.

**Continuations as homomorphisms** As we have said before, the overall goal of this analysis of $\mathcal{C}$ and $\mu$ is the connection between Eilenberg-Moore algebras and the operators. In the above analysis, we try to establish the connection from one direction, i.e., to see Eilenberg-Moore equations in operational equations. In the following, we would like to approach the other direction, namely, to read Eilenberg-Moore equations operationally. In that reading, we will consider the structure map of an algebra, say $\theta$, as a control operator. For that to work out, we need an assumption: to consider continuations as homomorphisms. That assumption is suggested by the Eilenberg-Moore equations (the multiplication one in particular), by continuation-passing style translations (the negative interpretation of types in particular), and by other semantic study (call-by-push-value interpretation of stacks in particular).

For convenience, we shall work in the category Set. The same reading will work for any cartesian-closed category with equalisers. (To be precise, we do not need all equalisers, only those that allow the comparison functor from the adjunction formed by the self-adjoint $R^{(-)}$ to have a left adjoint. That is not our concern here.) Let $(X, \theta)$ be an Eilenberg-Moore algebra. We define,

> Continuations from $(X, \theta)$ are homomorphisms from that algebra to the canonical algebra on $R$.

When spelt out, the homomorphism equation is as follows: $k \in R^X$ is such a homomorphism if

$$k(\theta M) = Mk \quad \text{for any } M \in R^{R^X}. \tag{2.17}$$

This is a semantic viewpoint of continuations, and we shall call it **the homomorphism principle**. This viewpoint shall be natural if we have enough experience about the structure map. For the moment, we may also consider the following evidence.

1. The homomorphism equation (2.17) is the inner part of the multiplication equation (equation 2.13, p42, abbreviated as 2.16, p43). From our comparison of operational and Eilenberg-Moore equations, we see that the function $k \in R^X$, which occurs in the multiplication equation, corresponds to a continuation in operational equations. This suggests a possible connection between continuations and homomorphisms.

2. Then we have the following solid evidence. Consider continuation-passing style translations, both call-by-value and call-by-name (e.g. p32). For the purpose here, we take a further step and interpret the translated types and terms in Set, i.e. as sets and functions. For both evaluation strategies, we have the following statement: for any type $A$, we have an injection
$$[\![A]\!]_n \xrightarrow{\text{ev}} \hom\bigl(R^{[\![A]\!]_n}, R\bigr) = \hom\bigl([\![A]\!], R\bigr).$$
This is admittedly a terse statement, and we shall expand it in words. Here, hom means the homset between Eilenberg-Moore algebras, and $R^{[\![A]\!]_n}$ carries a canonical algebra whose structure map is $R^{\text{ev}}$. ($R^{[\![A]\!]_n}$ is considered as a power of the canonical algebra on $R$.) Given an element $k \in [\![A]\!]_n$, the evaluation function ev maps it to $\text{ev}_k$, i.e. to evaluate at $k$, which is a function $R^{[\![A]\!]_n} \to R$. This ev function is an injection. Moreover, we can show that for any set $X$ and $x \in X$, the function $\text{ev}_x$ is a homomorphism from $R^X$, the power of the canonical algebra on $R$, to the canonical algebra on $R$. Thus in the injection, we can trim the codomain, from the set of all functions $R^{[\![A]\!]_n} \to R$ to the set of homomorphisms, i.e. $\hom\bigl(R^{[\![A]\!]_n}, R\bigr)$. We recall that the negative interpretation $[\![A]\!]_n$ is the set of continuations from $A$, thus we arrive at the following reading of the injection:

   > Every continuation can be considered as a homomorphism.

   Up to here, we already understand the nature of continuations. Checking against the homomorphism principle, the remaining question is whether continuations are all homomorphisms. That is less important from the viewpoint of applications, and it is more of a mathematical question of its own. (To be more specific, we are looking at one isomorphism of the monadicity equivalence, i.e. the isomorphism $X \cong \hom(R^X, R)$. The monadicity is of the self-adjoint functor $R^{(-)}$, with the base category Set. We shall not get into monadicity here.) The short explanation is that the distinction between continuations (defined operationally) and homomorphism is very thin, so that in common cases they are the same. (In our case the base category is Set. If the result set $R$ has more than two elements, then we can show that continuations are indeed all homomorphisms from $[\![A]\!]$ to $R$.)

3. We also have circumstantial evidence from similar studies. In call-by-push-value languages, we interpret type constructors $F$ and $U$ by a pair of (strong) adjoints; closed stacks of a CK-machine are interpreted as morphisms in the left category (the codomain category of the left adjoint) [64]. If we choose the Eilenberg-Moore adjunction as the pair of adjoints, then closed stacks of a CK-machine are interpreted as homomorphisms between Eilenberg-Moore algebras. In the paper [64], our setting is the monad example, with the continuation monad in particular.

(*End of motivation*)

**Reading the Eilenberg-Moore equations operationally** With the homomorphism principle, we can try an operational reading of the Eilenberg-Moore equations. Continue with the same setting, we work with the base category Set, and let $(X, \theta)$ be an Eilenberg-Moore algebra of the continuation monad. The structure map $\theta$ is a function $R^{R^X} \to X$. The operational reading of $\theta$ is as follows:

> We read $\theta$ as a control operator. In particular, we read it as a non-binding one, much like $\mathcal{C}$. In reading $\theta M$, where $M \in R^{R^X}$, we consider $\theta$ takes the current continuation, and feeds it to $M$. Since continuations are defined as homomorphisms, to be more precise, we shall say $\theta$ feeds the underlying function of the current continuation to $M$.

Let us read the Eilenberg-Moore equations this way. The equations are

$$\begin{array}{rll} \text{(unit)} & x = \theta\lambda k^{R^X}.\, kx & \\ \text{(multiplication)} & k(\theta M) = Mk & \text{under } \theta\,\lambda k^{R^X}\text{, for any } M \in R^{R^X}. \end{array}$$

(Formulated on p43, and the multiplication equation is in the abbreviated form.)

(i) Consider the unit equation, and suppose the current continuation is $h$. In $\theta\lambda k^{R^X}.kx$, the operator $\theta$ takes the current continuation $h$, and feeds it to $\lambda k\,.\,kx$. The result is $hx$, the same as feeding $x$ to the current continuation $h$.

(ii) Consider the multiplication equation, and suppose the current continuation is $h$. Consider $\theta$ as an operator. It takes the current continuation $h$, and feeds it to $\lambda k\,.\,k(\theta M)$ and $\lambda k\,.\,Mk$ respectively. Thus under $\theta\lambda k$, all $k$ should be considered as the current continuation $h$. Then the equality $k(\theta M) = Mk$ is the defining equation of $k$ being a homomorphism. Its operational reading is described in the general principle.

In the reading of either equation, we have added into the assumption a current continuation. This is mathematically sound when we work with the base category Set. In this case, the self-adjoint functor $R^{(-)}$ is monadic. In such an operational reading, we are using one isomorphism of the monadicity equivalence, namely $(X, \theta) \cong R^{\hom\left((X,\theta),R\right)}$, in the Eilenberg-Moore category. (We have mentioned the other isomorphism when giving evidence to the homomorphism principle, namely the evidence 2.) Assuming the homomorphism principle, $\hom\big((X,\theta), R\big)$ provides the continuations we use in the operational reading, with the help of an uncurrying.

In the end of the above introduction to $\mu$, we considered a language that has a variant of the $\mathcal{C}$ operator but also has indefinite judgements. If we work with that language, then we will be able to formulate operational equations that exactly match the Eilenberg-Moore equations. This is worth knowing, even though we do not have to actually do that.

By now, we have built up a good understanding of $\mathcal{C}$ and $\mu$ towards control algebras. We see that $\mathcal{C}$ and $\mu$ are closely related to the structure map of an Eilenberg-Moore algebra, and we have examined some evidence that supports such a claim. If we would like to uncover the structure map and the Eilenberg-Moore equations with control operators in the literature, then that is as close as we can get. We also see how to read the Eilenberg-Moore equations operationally. That is very useful, especially when equations get more involved. That is as much as we would like to say about $\mathcal{C}$ and $\mu$ by themselves. In the following, we shall break them into other common control operators. We shall break $\mathcal{C}$ into call/cc and $\mathcal{A}$, and break $\mu$ into catch and abort.

## 3 The operators call/cc with $\mathcal{A}$, and catch with abort

In the following we shall break $\mathcal{C}$ and $\mu$ into other control operators. In particular, $\mathcal{C}$ is equivalent to call/cc paired with the operator $\mathcal{A}$; the operator $\mu$ is equivalent to the operator 'catch' paired with the operator 'abort'. The operator $\mathcal{A}$ is the abort operator in the $\mathcal{C}$ setting.[16] However, to avoid confusion, we shall usually refer to it by the original notation $\mathcal{A}$. The name 'abort' will refer to the $\mu$-setting operator without further explanation. The operator call/cc appeared in the second revised report of Scheme [14]. It was formalised by Felleisen et al. [17][16][20], where the main interest was $\mathcal{C}$. The origin of the catch operator can be traced back to Lisp. The formulation that we shall introduce here is due to Crolard [44]. These two pairs of operators motivate control algebras.

**The untyped languages and operational descriptions** We shall formulate call/cc and $\mathcal{A}$ first, following Felleisen et al. (see the citations above). As in the formulation of $\mathcal{C}$, the new operators coerce continuations into functions in the language. When typed, such coercion requires an explicit $\perp$ type, and continuations are typed as functions to $\perp$. In the analysis of $\mathcal{C}$ (and contrasted in $\mu$), we have seen that such a setting is economic in the language facilities, as no new syntax is needed except the introduction of new term constructors. The trade-off is that the theory is less satisfactory. Nevertheless, we will make other improvements in the following presentation, and for that reason we decide to retain the coercion. That way the connection to the existing treatment remains recognisable.

**Definition.** The language with call/cc and $\mathcal{A}$ have the following terms,

$$x \quad \lambda x.M \quad MN \quad \mathrm{call/cc}\, M \quad \mathcal{A}M$$

constructed inductively. Among them $x$ is any variable, and $M$, $N$ are terms of this language.

For call/cc $M$, the intention is that $M$ expects an argument; and for $\mathcal{A}M$, the intention is that $M$ expects no argument. This is invisible in the untyped setting, but operational descriptions below will clarify. It is tempting to replace the notation 'call/cc' with a single symbol for convenience. However, since there are already enough notations and variants around, we leave the bold act for future authors and in informal occasions.

The intended operational meaning of call/cc and $\mathcal{A}$ are as follows. In call/cc $M$, the operator call/cc duplicates the current continuation, say $k$, and feeds it to $M$, and the current continuation remains. The computation continues with $Mk$, and with the the same current continuation $k$. In $\mathcal{A}M$, the current continuation is discarded, the computation continues with $M$, with an empty continuation. Formal descriptions are as follows.

($a$) In the CK-machine formalism, the relevant transition rules are

$$\begin{gathered}\mathrm{call/cc}\, M\,,\, k \rightsquigarrow M\,,\, \mathrm{coer}\, k :: k \\ \mathcal{A}M\,,\, k \rightsquigarrow M\,,\, \mathrm{nil}\end{gathered}$$

together with transitions that use coer $k$, as in the definition of $\mathcal{C}$ (transitions (2.2), (2.3), page 29). Also as in the $\mathcal{C}$ case, the language shall be suitably extended, and coer $k$ may be regarded as a value in the call-by-value description.

($b$) The continuation-passing style translations are as follows.

$$\begin{aligned}[\![\mathrm{call/cc}\, M]\!]k &= \begin{cases}[\![M]\!]\, \mathrm{ev}_{[k],k} & \text{call-by-value} \\ [\![M]\!]\, \big\langle [k], k \big\rangle & \text{call-by-name}\end{cases} \\ [\![\mathcal{A}M]\!]k &= [\![M]\!]\, \mathbf{I}\,.\end{aligned}$$

[16] For example, Griffin [25] calls it the abort operator. The work is in the $\mathcal{C}$ setting.

Here $\mathrm{ev}_{[k],k}$ means to feed $[k]$ and $k$ subsequently, $[k]$ is the coercion of continuations to functions, whose definition depends on call-by-value and call-by-name, $\langle -,- \rangle$ is an encoding of an ordered pair, and $\mathbf{I}$ is the identity combinator. In the untyped setting, $\mathrm{ev}_{x,y}$ is the same as $\langle x, y \rangle$, but the different presentation allows generalisation to the typed setting. These notations are defined in the introduction to continuations (page 26), and in the introduction to $\mathcal{C}$ (page 30). The reader may refer there for details. The need for the coercion $[k]$ and the identity $\mathbf{I}$ is the consequence of having an explicit $\bot$ type.

(*End of operational descriptions*)

We shall formulate the typing later. We will see that the typed language has coarsely-typed continuations, with explicit result type $\bot$. In contrast, The *catch* operator is based on the same operational idea, but has precisely typed continuations in the typed setting. We will pair 'catch' with an operator *abort*. The only difference between this abort and $\mathcal{A}$ is the following: when typed, the language with $\mathcal{A}$ has explicit $\bot$ type. We have kept the operator names as they occur in the references. For convenience, we shall refer to $\mathcal{A}$ as the abort operator in the *functionalised-continuation* setting. In the typed setting, if we use indefinite judgements, then the difference of the two abort operators vanishes.

**Definition.** The language of *catch* and *abort* has two disjoint set of variables: there are ordinary variables, written as $x$, $y$, etc. and control variables, written as $\alpha$, $\beta$, etc.. The language has unnamed terms and named terms, defined by a mutual induction. *Unnamed terms* are

$$x \quad \lambda x\,.\,M \quad MN \quad \mathrm{catch}\,\alpha\,.\,M \quad \mathrm{abort}\,P$$

where $x$ is any ordinary variable, $\alpha$ is any control variable, $M$, $N$ are unnamed terms, and $P$ is any named term. The operator catch is a binder of control variables. Thus in catch $\alpha\,.\,M$, catch binds $\alpha$. *Named terms* are

$$\alpha M$$

where $M$ is an unnamed term. As in the case of $\mu$, we may say $\alpha$ names the remainder of $M$. The term catch $\alpha\,.\,M$ is read as 'catch the current continuation as $\alpha$, in M'.

The intended operational meaning of the two operators are the same as in the call/cc and $\mathcal{A}$ case. We shall give formal descriptions directly.

(*a*) In the CK-machine formalism, the relevant transition rules are as follows, for both call-by-value and call-by-name:

$$\mathrm{catch}\,\alpha.M\,,\,k \rightsquigarrow M[k/\alpha]\,,\,k$$
$$\mathrm{abort}\,M\,,\,k \rightsquigarrow M\,,\,\mathrm{nil}$$

together with the use of continuations, as formulated in the $\mu$ case (transition rule (2.6), page 35).

(*b*) The continuation-passing style translations are as follows, for both call-by-value and call-by-name:

$$[\![\mathrm{catch}\,\alpha.M]\!]k = \big(\lambda\alpha.[\![M]\!]k\big)k$$
$$[\![\mathrm{abort}\,M]\!]k = [\![M]\!]$$

together with the translation of $\alpha M$, as in the $\mu$ case (translation clause (2.8), page 35).

(*End of operational description*)

This finishes the definitions of call/cc, $\mathcal{A}$, catch and abort.

**The relation with Eilenberg-Moore control operators** From the above definitions, we see that call/cc and $\mathcal{A}$ share the same language facilities as $\mathcal{C}$, so do 'catch', 'abort' and $\mu$. Thus for the purpose of comparison, we may put $\mathcal{C}$, call/cc and $\mathcal{A}$ into one language, and the same for $\mu$, 'catch' and 'abort'.

Using both the CK-machine and continuation-passing style translations, we verify that call/cc and $\mathcal{A}$ can be expressed in $\mathcal{C}$, for both call-by-value and call-by-name:

$$\text{call/cc}\, M \simeq \mathcal{C}\, \lambda k.k(Mk) \tag{2.18}$$

$$\mathcal{A}M \simeq \mathcal{C}\, \lambda k.M \quad k \text{ not free in } M. \tag{2.19}$$

The calculations are straightforward, although some patience may be needed, e.g. the call-by-value continuation-passing style translation. We leave the details as an exercise to the reader, or to proof assistants. Due to these operational equivalence, the equalities are also taken as definitions of call/cc and $\mathcal{A}$. The equations can be seen in Felleisen et al.'s work for example. See the references mentioned earlier. Conversely, we can verify that $\mathcal{C}$ may be expressed in call/cc and $\mathcal{A}$:

$$\mathcal{C}M \simeq \text{call/cc}\, \lambda k\,.\,\mathcal{A}(Mk).$$

Thus, the language with $\mathcal{C}$ and the language with call/cc and $\mathcal{A}$ are equivalent.

For $\mu$, catch and abort, we have similar translations. The operators catch and abort can each be expressed in $\mu$:

$$\text{catch}\, \alpha\,.\,M \simeq \mu\alpha\,.\,\alpha M \tag{2.20}$$

$$\text{abort}\, P \simeq \mu\alpha\,.\,P \quad \alpha \text{ fresh.} \tag{2.21}$$

Conversely, $\mu$ can be expressed in catch and abort:

$$\mu\alpha\,.\,P \simeq \text{catch}\, \alpha\,.\,\text{abort}\, P\ .$$

Thus, the language with $\mu$ and the language with catch and abort are equivalent.

These equivalence results express the same structural fact: from an Eilenberg-Moore algebra we can construct a corresponding control algebra, and vice versa. and the constructions give an isomorphism. The constructions can be formulated as follows. We shall not present the defining equations for control algebras here, only to say that for a control algebra $(X, \xi, \zeta)$, the operations have the following forms respectively:

$$\xi\colon X^{R^X} \to X \qquad \zeta\colon R \to X.$$

From an Eilenberg-Moore algebra $(X, \theta)$, we can construct a control algebra $(X, \xi, \zeta)$, where

$$\xi M = \theta\, \lambda k^{R^X}.k(Mk)$$

$$\zeta r = \theta\, \lambda k^{R^X}.r\ \ .$$

Conversely, given a control algebra $(X, \xi, \zeta)$, we can construct an Eilenberg-Moore algebra $(X, \theta)$, where

$$\theta M = \xi\, \lambda k^{R^X}.\zeta(Mk).$$

We omit calculations that show the isomorphism here. Some details will be given in the chapter on control algebras.

**Equations of call/cc with $\mathcal{A}$, and catch with abort** With the new operators understood, we present candidate operational equations that correspond to defining equations of control algebras. In each case, say call/cc paired with $\mathcal{A}$, and catch paired with abort, we have one unit equation, and one multiplication equation for each operator. The unit equations are the same as before, only by expressing $\mathcal{C}$, and respectively $\mu$, in the new operators. In the call/cc and $\mathcal{A}$ case, we expect the following equations to hold. The unit equation is

$$M \simeq \text{call/cc}\,\lambda k\,.\,\mathcal{A}(kM) \quad k \text{ does not occur free in } M.$$

The multiplication equations are, under call/cc $\lambda k$,

$$\begin{aligned} k\,\text{call/cc}\,M &\simeq k(Mk) && \text{for any term } M && (2.22)\\ k\,\mathcal{A}M &\simeq M && \text{for any term } M. && (2.23)\end{aligned}$$

We are using the abbreviating convention introduced in the summary of $\mathcal{C}$ and $\mu$ (p42, eqs. 2.14, 2.15 and the convention that follows). We shall also explain typing later; regarding that typing, the simplest typable instances have $\mathcal{A}[\cdot]$ as the padding context, i.e. the part left out by the abbreviation (see the convention mentioned above). We can verify that the following instances hold, using CK-machine and continuation-passing style translations:

$$\begin{aligned} \text{call/cc}\,\lambda k\,.\,\mathcal{A}\big(k\,\text{call/cc}\,M\big) &\simeq \text{call/cc}\,\lambda k\,.\,\mathcal{A}\big(k(Mk)\big)\\ \text{call/cc}\,\lambda k\,.\,\mathcal{A}\big(k(\mathcal{A}M)\big) &\simeq \text{call/cc}\,\lambda k\,.\,\mathcal{A}M\ .\end{aligned}$$

Similarly, in the catch and abort case, we expect the following equations to hold. The unit equation is

$$M \simeq \text{catch}\,\alpha\,.\,\text{abort}\,\alpha M \quad \alpha \text{ does not occur free in } M.$$

Multiplication equations are, under catch $\alpha$,

$$\begin{aligned} \alpha\,\text{catch}\,\beta\,.\,M &\simeq \alpha\big(M[\alpha/\beta]\big) && \text{for any term } M \text{ and control variable } \beta \neq \alpha && (2.24)\\ \alpha\,\text{abort}\,Q &\simeq Q && \text{for any named term } Q. && (2.25)\end{aligned}$$

Hypothetically, had we known those are the equations to look at, then we shall come to the following equations for a control algebra $(X, \xi, \zeta)$. The unit equation is

$$x = \xi\,\lambda k^{R^X}.\,\zeta(kx).$$

The multiplication equations are, under $\xi\,\lambda k^{R^X}$,

$$\begin{aligned} k(\xi M) &= k(Mk) && \text{for any } M \in X^{R^X}\\ k(\zeta r) &= r && \text{for any } r \in R\,.\end{aligned}$$

The three equations are exactly the defining equations of a control algebra $(X, \xi, \zeta)$.

In presenting the multiplication equations, we have applied an abbreviation similar to that in an Eilenberg-Moore algebra (equation 2.16, page 43). The full forms of the equations are,

$$\begin{aligned} \xi\,\lambda k^{R^X}.\,\Phi(k \circ \xi) &= \xi\,\lambda k^{R^X}.\,\Phi(k \circ \mathrm{ev}_{X,k}) && \text{for any } \Phi\colon R^{X^{R^X}} \to X\\ \xi\,\lambda k^{R^X}.\,\Psi(k \circ \zeta) &= \xi\,\lambda k^{R^X}.\,\Psi(\mathrm{id}_R) && \text{for any } \Psi\colon R^R \to X\end{aligned}$$

In the same place we abbreviate the Eilenberg-Moore equations, we also formulate an abbreviation rule (page 43). Here we take $B = X$, $Y = X^{R^X}$ and $\theta = \xi$ for the multiplication equation of $\xi$; keeping $B = X$ and $\theta = \xi$, we take $Y = R$ for the multiplication equation of $\zeta$.

**Typing call/cc, $\mathcal{A}$, catch and abort** We have seen that these new operators can be expressed in $\mathcal{C}$ or $\mu$ (eqs. 2.18, 2.19, 2.20, 2.21, p49). By inspecting these equations, we can derive the typing rules of the new operators. Alternatively, we may obtain the same rules by considering the new operators only. Such derivations use the multiplication laws (eqs. 2.22, 2.23, p50; eqs. 2.24, 2.25, p50). As the laws are operational equations, we can say that the typing rules are supported by operational considerations. In the original work [25], Griffin motivated the typing rules by drawing an analogy to logic. We would like to show that operational considerations alone suffices, and the connection to logic emerges afterwards. We take this operational approach here. It should be noted that the derivations below are plausible reasoning.[17]

We first type call/cc and $\mathcal{A}$. We follow Griffin on the setting: there is a type $\bot$, and continuations are typed as functions to $\bot$. For example, a continuation from $A$ is typed as $A \to \bot$, abbreviated as $\neg A$. Strictly speaking, continuations are coerced into functions. The study of $\mathcal{C}$ and $\mu$ has shown that the theory is smoother if continuations are not considered as functions *in the language* (cf. summary of $\mathcal{C}$, p33; introduction to $\mu$ and summary of $\mu$, p34 and p41 resp.). Semantic analysis suggests more specifically that they should be homomorphisms (the homomorphism principle, p44). Nevertheless, we follow the original setting here. The only deviation is that we are more specific in the typing of call/cc. We will make it precise when we get there.

In call/cc $M$, the term $M$ expects an argument, so it is typed as a function. Putting typing contexts aside, we are looking for a typing rule

$$\frac{M\colon C \to B}{\text{call/cc}\, M\colon A}$$

where the relation of $A$, $B$, $C$ is to be determined. Consider the multiplication law (2.22, p50): under call/cc $\lambda k$,

$$\underset{A}{k}\, \text{call/cc}\, M \simeq \underset{B}{k}\, (M \underset{C}{k}) \quad \text{for any term } M.$$

($A$, $B$, $C$ are added to type the subterms to the right.) Looking at the equation, we see that $B$ has to be the same as $A$, whereas $C$ types continuations. In this setting, a continuation from $A$ is coerced into a function $A \to \bot$, abbreviated as $\neg A$. Thus the full typing rule is

$$\frac{\Gamma \vdash M\colon \neg A \to A}{\Gamma \vdash \text{call/cc}\, M\colon A}. \tag{2.26}$$

If we only display relevant types, then the rule is

$$\frac{\vdash \neg A \to A}{\vdash A}.$$

This may be seen as a special case of Peirce's Law: $\big((A \to B) \to A\big) \to A$, with $B$ taken as $\bot$. Fixing $B$ to be $\bot$ is a stricter requirement than Griffin's original formulation ([25] §5, written as $\mathcal{K}$). There the continuation argument of $M$ would be typed as $A \to B$ rather than $A \to \bot$, so that the typing rule would correspond to Peirce's Law in its full generality.[18]

[17] 'Plausible reasoning' as described by Polya (*Mathematics and Plausible Reasoning*, Vol. 1. Preface.). However, as usual in programming language theory and category theory, definitions, axioms and constructions are often forced upon us. What we may consider as guesses can often be obtained by performing some calculation. The typing rules here are such examples.

[18] Griffin's formulation is a good idea, but it probably requires delimited control to work out. Interested readers may try to formulate a full language with such a typing of call/cc, together with an operational description. The question to consider is: do all terms involving call/cc have a well-typed operational description?

For the operator $\mathcal{A}$, we look for a rule

$$\frac{M\colon B}{\mathcal{A}M\colon A},$$

with the relation of $A$ and $B$ to be determined. Consider the multiplication law (2.23, p50): under call/cc $\lambda k$,

$$\underset{\perp}{k}\,\underset{A}{\mathcal{A}M} \simeq \underset{B}{M} \quad \text{for any term } M.$$

As a continuation, $k$ is typed as a function to $\perp$. We conclude that $B$ should be $\perp$. Thus the full typing rule is

$$\frac{\Gamma \vdash M\colon \perp}{\Gamma \vdash \mathcal{A}M\colon A}\,. \tag{2.27}$$

The rule corresponds to the *ex falso* rule in logic. In Griffin's formulation, the rule is obtained by reducing $\mathcal{A}$ to $\mathcal{C}$. The typing rule of $\mathcal{C}$ is obtained by drawing an analogy to logic [25, §3]. We remark that there could be better call/cc and $\mathcal{A}$, just as there could be a better $\mathcal{C}$ (cf. summary of $\mu$, p41).

Next we type 'catch' and 'abort'. The typing adapts from previous work on $\mu$ (Streicher & Reus [42], Selinger [50]). The set-up is the same as in $\mu$: we have two kinds of contexts, one for ordinary variables and the dual context for control variables; we also have definite and indefinite judgements (p36). The deviation from the literature is also the same (comments on typing of $\mu$, p38): we do not use the $\perp$ type; a term that would otherwise be typed $\perp$ is defined in an indefinite judgement here, where no type is assigned. Terms defined in definite judgements are also called unnamed terms, whereas those defined in indefinite judgements are also named terms (the terminology is from the untyped case, p34).

For the *catch* operator, we need to fill-in missing type information in

$$\frac{\alpha \vdash M}{\vdash \text{catch}\,\alpha\,.\,M}.$$

The variable $\alpha$ is a control variable, so it is of a type in the dual context, say $C$. The typing rule should look like

$$\frac{;\,\alpha\colon C \vdash M\colon B}{;\ \vdash \text{catch}\,\alpha\,.\,M\colon A}$$

where the relation of $A$, $B$, $C$ is to be determined.[19] As in the call/cc case, we look at the multiplication law (2.24, p50), and conclude that $C$, $B$ should both be $A$. Taking other typing context into consideration, the typing rule is

$$\frac{\Gamma\,;\Delta_1\,,\,\alpha\colon A, \Delta_2 \vdash M\colon A}{\Gamma\,;\Delta_1, \Delta_2 \vdash \text{catch}\,\alpha\,.\,M\colon A}. \tag{2.28}$$

If we only display the relevant types, then the rule is

$$\frac{;\,A \vdash A}{;\ \vdash A}$$

Viewing $A$ in the dual context as the negation of $A$, we may also consider the rule a variant of Peirce's Law.

[19] Recall that in the typing of $\mu$, we separate the two kinds of contexts by a semicolon (;). We retain the same notation here. Also, for this particular argument, we choose to include no ordinary variables (thus the ordinary context is empty) and no control variables other than the relevant one; those variables are inessential for the type inference.

For the *abort* operator, we need to fill-in the missing type information in

$$\frac{P}{\text{abort}\, P}.$$

The term $P$ is named; named terms are defined in indefinite judgements, without being assigned any particular type. On the other hand, 'abort $P$' is unnamed; it should be assigned a type. Operational consideration (say, the multiplication law of abort, eq. 2.25) suggests that it can be of any ordinary type. Thus the typing rule is: for any ordinary type $A$,

$$\frac{\Gamma\,;\Delta \vdash P}{\Gamma\,;\Delta \vdash \text{abort}\, P \colon A}. \tag{2.29}$$

The typing rules motivate operations

$$\xi \colon X^{R^X} \to X \qquad \zeta \colon R \to X$$

for a control algebra on $X$. The operation $\xi$ corresponds to call/cc and catch, and $\zeta$ corresponds to $\mathcal{A}$ and abort.

# 4 Delimited control

Delimited control is a natural next step in the development of control operators. It is not part of the thesis, but we shall still say a few words. The description here is informal.

In the introduction to continuations (§1, p23), we mention that there is a subtlety in the understanding of continuations. In the CK-machine formalism, continuations are defined as (frame) stacks. The far end of such a continuation is equivalently the bottom of the stack, and it is reachable. By the far end being *reachable*, we mean that if the continuation is not discarded, such as in an abort operation, then overtime, the stack that presents it will eventually become empty. In the contrary, a continuation in a continuation-passing style translation has a far end that cannot be reached. That is visible in the typed setting: to obtain satisfying semantics, we are led to abandon the bottom type $\bot$ and adopt indefinite judgements (see the analysis of $\mu$). In view of such a difference, we say that continuations in a CK-machine have *real ends*, whereas continuations in a CPS transform have *imaginary ends*.[20]

Our semantic experience seems to suggest that continuations with imaginary ends are more fundamental. Importantly, the imaginary end of such a continuation need not (probably should not) exhibit as a type in the language. We have tried to convey that in the analysis of $\mathcal{C}$ and $\mu$. However, in the reduction-relation formulation of $\mathcal{C}$, if $\mathcal{C}M$ is considered as a full term (as opposed to a proper subterm), a special reduction rule applies, and it reduces $\mathcal{C}M$ to $M\mathbf{I}$ ($\mathbf{I}$ being the identity combinator $\lambda x\,.\,x$) [20]. From our viewpoint, the need for such a special rule is a manifestation of the subtle difference between continuations with real ends and with imaginary ends. Nevertheless, the problem can also be viewed syntactically (Felleisen [21], where $\mathcal{C}$ is written as $\mathcal{F}$ for the convenience of later development in the paper). In the above reduction-relation formulation of $\mathcal{C}$, the boundary of a continuation is implicit; to a subterm $\mathcal{C}M$, the continuation it sees can be understood as the evaluation context it is in. Thus in $D[\mathcal{C}M]$ where $D$ is an evaluation context, the continuation to $\mathcal{C}$ is $D$; but if we wrap it in further evaluation contexts, say $E[D[\mathcal{C}M]]$, then the continuation also extends to $E[D[\cdot]]$. Felleisen saw that the unannounced, moving boundary may be the source of the problem, and a place to change. In the same paper [21], Felleisen showed that setting boundaries for continuations leads to a

[20] Levy calls them *idealised stacks* [75].

satisfying reduction theory. He introduced a new term constructor to mark the boundary of continuations (written as #, an analogy to the prompt character in an interactive dialogue), and had the $\mathcal{C}$ operators respect that boundary (written as $\mathcal{F}$). Then continuations have their boundary explicitly announced, and are said to be *delimited*. With suitable reduction rules, the new control operators give a satisfying reduction theory. From a semantic viewpoint, I think prompts work because they reserve places in the syntax where imaginary ends can be introduced in an interpretation. That introduction resolves the tension between real and imaginary ends. To make that statement precise is work for the future. After prompts were introduced, Sitaram and Felleisen considered practical aspects of the delimited $\mathcal{C}$ (there named *control*) and prompts [26].

Danvy and Filinski introduced control operators *shift* and *reset*, which are similar to Felleisen's $\mathcal{F}$ and prompts [24]. The operators are compared in the paper (§5).

# 5 Catch-throw (letstk-changestk), and a difficulty in formulating them as algebras

*Catch-throw* [44] and *letstk-changestk* [56, §5.4][75] are two pairs of control operators that are essentially the same. Catch and throw arise from the study of $\mu$, whereas letstk and changestk are the corresponding operators in Levy's call-by-push-value setting. Recall that our long-term goal is to describe common features between state and control (the motivation of this work, p2). Formulating algebras for catch-throw is an intermediate step towards that goal. We have seen the operator *catch* (definition of catch and abort, p48; typing of catch, p48). We shall introduce *throw*, then explain what makes the formulation problem non-trivial.

**Definition.** The language of *catch* and *throw* has two disjoint set of variables: there are ordinary variables, written as $x$, $y$, etc. and control variables, written as $\alpha$, $\beta$, etc.. Terms are

$$x \quad \lambda x\,.\,M \quad MN \quad \mathrm{catch}\,\alpha\,.\,M \quad \mathrm{throw}\,\alpha\,M$$

where $x$ is any ordinary variable, $\alpha$ is any control variable, and $M$, $N$ are terms of the language. The operator catch is a binder of control variables. The term 'throw $\alpha\,M$' is read as 'throw $M$ to $\alpha$'.[21]

In the language of catch and throw, there is no distinction of unnamed and named terms (see the definition of catch and abort, p48). Conceptually, all terms are unnamed; in a typed setting, they are all defined in definite judgements, and are each assigned an ordinary type. The intended meaning of 'throw $\alpha\,M$' is as follows: in evaluating such a term, all planned future work is discarded, and the computation turns to $M$, with the continuation being that referred to by $\alpha$. The formal descriptions are as follows, for both call-by-value and call-by-name.

(*a*) In the CK-machine formalism, the transition rule of the throw term is

$$\mathrm{throw}\,K\,M\,,\,k \rightsquigarrow M\,,\,K. \tag{2.30}$$

Here $K$ is a stack term, and we shall extend the language accordingly to incorporate stack terms. We have a stack term $K$ here instead of a control variable $\alpha$, because in a closed term, an occurrence of a control variable must be encapsulated by a catch term that binds that variable. By the time we come to evaluate the throw term, $\alpha$ must have been substituted by a stack term (cf. the operational description of catch, p48).

[21] We have made a change to punctuation in the syntax. In Crolard's formulation, the catch term is written as 'catch $\alpha\,M$', whereas we have added a dot (.) following the bound variable $\alpha$. This is consistent with other writing convention involving bound variables, say $\lambda x\,.\,M$, $\exists x\,.\,\varphi$ etc..

($b$) The continuation-passing style translation for the throw term is

$$[\![\text{throw}\,\alpha\,M]\!]\,k = [\![M]\!]\,\alpha. \tag{2.31}$$

Here the target language is the untyped $\lambda$-calculus. The variable $\alpha$ on the left hand side is a control variable in the language of catch and throw, whereas $\alpha$ on the right hand side is a variable in the $\lambda$-calculus.

Other CK-machine transitions and CPS translations remain the same (cf. operational description of catch and abort, p48).

**The relation of catch, throw and $\mu$** The operators catch and throw can each be expressed in $\mu$. We have seen catch expressed in $\mu$ (eq. 2.20, p49). For the throw term, we have

$$\text{throw}\,\alpha\,M \simeq \mu\beta\,.\,\alpha M \quad (\beta \text{ fresh}). \tag{2.32}$$

(Crolard [44] §2.1, defines catch and throw from $\mu$.) Here $\beta$ is a fresh control variable that does not occur free in $M$. Similar to what we have done before (expressing catch in $\mu$, p49), we combine the language of catch and throw, together with the language of $\mu$, so that we can compare them directly. In the combined language, we define the catch terms and throw terms as unnamed terms.[22] Also as before, this is an operational equivalence. It means that

(i) In the CK-machine formalism, for any context $C$ so that the two sides under $C$ are closed, the closed terms evaluate to the same configuration, when paired with the same initial continuation $k$;

(ii) Under the continuation-passing style interpretation, the two sides translate to the same $\lambda$-calculus term up to $\beta\eta$-equivalence.

As $\mu$ can be expressed in the catch-abort pair, the throw operator can also be expressed in catch-abort:

$$\text{throw}\,\alpha\,M \simeq \text{catch}\,\beta\,.\,\text{abort}\,\alpha M \quad (\beta \text{ fresh}). \tag{2.33}$$

Conversely, $\mu$ can be expressed in the catch-throw pair. We have

$$\mu\alpha\,.\,\boldsymbol{\beta} M \simeq \text{catch}\,\alpha\,.\,\text{throw}\,\boldsymbol{\beta}\,M. \tag{2.34}$$

(Crolard, loc. cit..) Here $\boldsymbol{\beta}$ is a meta-variable of arbitrary control variables, including $\alpha$. Note that the left hand side $\mu\alpha\,.\,\boldsymbol{\beta} M$ is the general form of a $\mu$ term. This is because in $\mu\alpha.P$, the subterm $P$ is required to be named by definition.

**Equations of catch and throw** The *unit* equation is

$$M \simeq \text{catch}\,\alpha\,.\,\text{throw}\,\alpha\,M \quad (\alpha \text{ does not occur free in } M). \tag{2.35}$$

(Levy [56], §7.8.3, Fig. 7.11, p163, second to last of the $\eta$-laws; Crolard [44], Def. 2.1.1, as a consequence of reduction rules 7 and 8.) Counterparts of multiplication equations are more subtle. Recall that in the language of catch and abort, the multiplication equation of catch is

[22] It may be tempting to consider only $\alpha M$ on the right hand side. However, recall that $\alpha M$ is named, whereas the throw terms are intended as unnamed. Thus the dummy $\mu\beta$ is necessary. The intention will be clearer when we look at typing, where throw terms are assigned types, but not $\alpha M$ in the language of $\mu$ (typing rules for terms in the language of $\mu$, p37).

as follows (eq. 2.24, p50; also follow the cross reference there for the abbreviation ‘under …’): under catch $\alpha$,

$$\alpha\,\text{catch}\,\beta\,.\,M \simeq \alpha\big(M[\alpha/\beta]\big) \quad (\beta \neq \alpha).$$

However, in the language of catch and throw, we do not have named terms $\alpha M$. An analogue of that would be a throw term. Thus an approximation of the multiplication equation of catch would be: under catch $\alpha$,

$$\text{throw}\,\alpha\,\text{catch}\,\beta\,.\,M \simeq \text{throw}\,\alpha\,\big(M[\alpha/\beta]\big) \quad (\beta \neq \alpha). \tag{2.36}$$

(Crolard loc. cit., reduction rule 6; Levy loc. cit., second to last of the $\beta$-laws.) Similarly, the multiplication equation of throw may be: under catch $\alpha$,

$$\text{throw}\,\alpha\,(\text{throw}\,\boldsymbol{\beta}\,M) \simeq \text{throw}\,\boldsymbol{\beta}\,M. \tag{2.37}$$

Here $\boldsymbol{\beta}$ is a meta-variable that could be $\alpha$ or other control variables. (Crolard's reduction rule 7, loc. cit.; Levy's last $\beta$-law, loc. cit..) We know that these equations hold, but the interesting question is whether they are complete, i.e., suffice to derive all equations involving consecutive uses of catch and throw (e.g., Crolard's reduction rule 4, 7, 8, loc. cit.; in Levy's equational theory, the three equations 2.35, 2.36, 2.37 we propose here are all equations regarding letstk and changestk only, loc. cit.). On the semantic side, the corresponding problem is to find a good set of equations that define some algebras, whose operations correspond to catch and throw. We shall see in a moment that there is a more fundamental difficulty in formulating such algebras. We shall consider typing of the throw operator first.

**Typing the throw operator** To obtain a typing rule for throw terms, we need to fill-in the missing typing information in

$$\frac{\alpha \vdash M}{\vdash \text{throw}\,\alpha\,M}.$$

The basic setting is that we have a context for ordinary variables, and a dual context for control variables; we have only definite judgements $\Gamma\,;\Delta \vdash M\colon A$ (cf. the typed language of $\mu$, p37). We first assume that the control variable $\alpha$ has some type $C$, dually. Putting other parts of typing contexts aside, the typing rule should be

$$\frac{;\,\alpha\colon C \vdash M\colon B}{;\ \vdash \text{throw}\,\alpha\,M\colon A}.$$

The relation of $A$, $B$ and $C$ is to be determined. We consider the CK-machine transition rule (eq. 2.30, p54), decorated with types:

$$\text{throw}\,\underset{C}{K}\,\underset{B}{M}\underset{A}{\,,}\,k \rightsquigarrow M\underset{B(=C)}{\,,}\,K.$$

In the configurations, the type under the separating comma (,) is the type of a configuration ($A$, resp. $B$). It is the type of the term to the left (‘throw $K\,M$’, resp. $M$), as well as the type that the stack is from ($k$, resp. $K$). Other decorations indicate types of the subterms to the right: $K$ is dually of type $C$, and $M$ is of type $B$, as we have assumed ($K$ taking the place of $\alpha$).[23]

[23] Note that in CK-machine transition rules (of closed terms), when a throw term is the left configuration, the subterm that immediately following ‘throw’ must be a stack term. That is because, by the time the throw term becomes the left configuration, the control variable that originally following ‘throw’ would have been substituted.

Considering types of the two $K$'s on each side, we infer that $B$ must be the same as $C$. On the other hand, since the original stack $k$ is discarded, it does not matter what type $A$ is; in other words, $A$ can be arbitrary, with no particular relation to $B$ or $C$. Therefore the full typing rule of *throw* is: for any ordinary type $A$ and $B$,

$$\frac{\Gamma\,;\Delta_1,\alpha\colon B,\Delta_2 \vdash M\colon B}{\Gamma\,;\Delta_1,\Delta_2 \vdash \mathrm{throw}\,\alpha\,M\colon A}.$$

**A difficulty in formulating algebras for catch and throw** We consider formulating algebras for catch and throw, say in the category Set. We already know that on a set $X$, a catch operation is a function $X^{R^X} \to X$ (cf. operations catch and abort, p53). The typing of throw suggests that for a set $X$, an operation for throw should look like

$$\zeta_Y\colon R^Y \times Y \to X$$

for any set $Y$. Here $Y$ takes the role of the type $B$, and $X$ takes the role of $A$. An attempt to formulate catch-throw algebras may look like the following:

> A catch-throw algebra is a set $X$ together with a function
>
> $$\xi\colon X^{R^X} \to X \qquad \text{(the catch operation)}$$
>
> and for every set $Y$, a function
>
> $$\zeta_Y\colon R^Y \times Y \to X, \qquad \text{(the throw operation at } Y\text{)}$$
>
> such that some naturality condition is satisfied for $\zeta_Y$ in $Y$, and the functions $\xi$, $\zeta_Y$ satisfy certain equations, say a transcription of the unit and multiplication equations we have formulated earlier (eqs. 2.35, 2.36, 2.37).

In such a formulation, we have only requested the parameter $Y$ in $\zeta_Y$ be sets without asking for further structure, but that may need some consideration:

(i) In the catch-throw language, the term $M$ in 'throw $\alpha\,M$' may use catch and throw inside. Thus, if we choose to interpret $M$ as an element in $Y$, then $Y$ should also carry a catch-throw algebra. However, if we did refer to other catch-throw algebras in the definition, then we would have to define all catch-throw algebras together. In other words, the definition would start with: the *category of* catch-throw algebras consists of . . . '.

(ii) If we let $Y$ be sets as in the attempt, then we need to convince ourselves that sets are sufficient. As a consequence, in the naturality condition of $\zeta_Y$, we would be talking about functions and not homomorphisms. Would that be too stringent?

I presume that we are more willing to follow (ii) and see what we get. The expectation is that catch-throw algebras form a category isomorphic to that of control algebras (algebras of catch and abort, so to speak). At the very least, all control algebras should be catch-throw algebras (see below). In any case, the answer is not obvious.

One may try to bypass the question altogether, and propose that we define catch-throw algebras by mandating the following: the functions $\zeta_Y$ are in the form

$$R^Y \times Y \xrightarrow{\text{eval.}} R \xrightarrow{\zeta} X,$$

where the first morphism is the evaluation morphism, and the second morphism is the abort operation $\zeta$, which comes from a control algebra $(X, \xi, \zeta)$. This is possible, but we seem to miss some interesting mathematics here. The language of catch and throw seems to suggest that catch and throw give rise to an algebraic structure on their own.

Formulating catch-throw algebras is beyond the scope of the current work. However, once the definition is sorted out, we will be at a good position to consider common features of state and control, i.e., our eventual goal.

# Chapter 3

# Control algebras

## 1 Motivation and an overview to the chapter

Our main objective is to present the definition of control algebras (Definition 1, p63) and justify it. Control algebras are defined in the category of sets and functions; and the definition is justified by showing that the category of control algebras is isomorphic to the Eilenberg-Moore category of the continuation monad (The isomorphism theorem, p81). If we give a rather brief and plain description of them, then the chapter may seem rather unremarkable: a control algebra consists of merely two operations and three equations; and, the justification of the definition is essentially a very long calculation. Indeed, for readers who merely want to register notable results of the chapter, it suffices to read the definition and note the isomorphism. But for readers who can afford some leisure, I would like to take the effort to motivate the algebras.

Why we would like to present the definition of control algebras and its justification to some detail? My answer is the following understanding (that I barely dare to say):

> I consider the definition here a good candidate for 'the definition' of algebras of control, much like the three-equation definition of lookup-update algebras may be considered 'the definition' of algebras of state. The proof of isomorphism is not completely trivial; and, along the way, we see some laws of control algebras.

Obviously, whether a definition may be considered canonical is quite subjective, and I have no way to justify the canonicity formally.[1] Nevertheless, if the reader is willing to accept the canonicity of the three-equation definition of lookup-update algebras (which we shall recall below), then we can compare the defining equations of control algebras and those of state algebras, and point out the resemblance. That is what we shall do in the remaining part of this motivation. However, some readers may not be familiar with the state effect, and such a motivation would not make much sense. Then, such readers may consider the subject as a purely mathematical one, and take the defining equations as unmotivated. It may well be a good excursion nonetheless; I see elegance and beauty from the equations and the isomorphism proof, and I think such merits persist even without operational understanding.[2]

[1] I feel it possible that others find some way to characterise the canonicity, say for the definitions of state algebras and control algebras mentioned here. It would be great if that is the case.

[2] If I chat with peer researchers, I would probably say the equations are wonderful (and still a bit mysterious).

**Motivating the definition of control algebras** We shall work in the category of sets and functions, and fix a set $R$ with at least two elements.[3] On a carrier set $X$, a control algebra consists of two operations: functions

$$\beta\colon X^{R^X} \to X \quad \text{and} \quad \alpha\colon R \to X.$$

They may be considered as the ideal semantic counterparts of control operators *call/cc* (that $\beta$ corresponds to) and *abort* (that $\alpha$ corresponds to). In what sense do $\beta$ and $\alpha$ correspond to them? Let us quickly review what call/cc and abort do, then come back to explain in more detail.[4]

Informally, to run a program with control effect, the environment keeps track of two pieces of information: what the program is focusing at hand, and what is still to be done; the latter is called the *current continuation.*[5] It is probably not hard to see that if one manipulates what is still to be done, then they can deviate the program from its planned route (also see the example on p23). We consider that the aim of the program is to produce some result, taken from the set $R$ that we have fixed beforehand. Then, the current continuation, i.e., the remaining work to be done, is considered as a function towards $R$.

- The control operator call/cc is used in a term 'call/cc $M$', where the subterm $M$ is the operand of the operator. What *call/cc* does, is to take a copy of the current continuation, say $k$, and make $k$ available to $M$ by feeding it as an argument. (In view of $\beta$, the current continuation $k$ may be considered as an element of $R^X$; the subterm $M$ may be considered as an element of $X^{R^X}$, the domain of $\beta$.) Then at some later moment, the subterm $M$ can make $k$ the current continuation of that moment; and by doing that, reset the remaining work of the program to that of an earlier moment.

- The operator abort is used in a term 'abort $N$', where the subterm $N$ is the operand, and may be considered as an element in $R$. What *abort* does, is to discard the current continuation, and produce $N$ as the result.

The above is an operational description, and a precise formulation of that requires setting up some formalism (see §2.3 for the formulation, and the definition of CK-machine in §2.1 for the formal set-up). How do equations on $\beta$ and $\alpha$ express that? Actually, the control algebra equations only hint at the operational description, in a rather implicit way. The three defining equations for a control algebra are:

(i) the unit law,

(ii) the contextual call/cc-homomorphism law, and

(iii) the contextual abort-homomorphism law.

For the purpose here, we can leave the unit law aside, and consider the second one, the contexual call/cc-homomorphism law. In a colloquial form, the *contextual call/cc-homomorphism law* may be formulated as follows (recall that $\beta\colon X^{R^X} \to X$ is the operation that corresponds to call/cc):

[3] We shall consider exponentials with base $R$, i.e., $R^{(-)}$. For $R$ with fewer elements, those exponentials degenerate; with the minimum size requirement on $R$, we have monadicity for the self adjoint functor $R^{(-)}$.

[4] We have described call/cc and abort formally in §2.3, p46. We recall them here briefly, and in an informal way, so that the reader can continue with the upcoming explanation after this short refreshment. It is also hoped that readers who have not read through the formal description can still get a reasonable idea of the two control operators.

[5] We are using the term 'environment' informally. Formally, what we are describing here are *configurations.* We give formal definitions of them in Chapter 2, for different variants of control operators.

> For any $C \in X^{R^{X^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.C(-)$, the two expressions $k \circ \beta$ and $\lambda N^{\in X^{R^X}}.k(Nk)$ are equal.

The more conventional way to say that is, for any $C \in X^{R^{X^{R^X}}}$,

$$\beta\,\lambda k^{\in R^X}.C(k \circ \beta) = \beta\,\lambda k^{\in R^X}.C\,\lambda N^{\in X^{R^X}}.k(Nk)\,.$$

In the colloquial form, if we apply both expressions $k \circ \beta$ and $\lambda N^{\in X^{R^X}}.k(Nk)$ to some element $N \in X^{R^X}$, then we get $k(\beta N)$ and $k(Nk)$. If we consider $k$ to be some current continuation, and consider $\beta$ as call/cc, then by comparing $k(\beta N)$ and $k(Nk)$, the implicit call/cc-homomorphism law hints at the operational description. We say such a hint is rather implicit, because to make an equality, we have to wrap the two expressions into some context, i.e., $\beta\,\lambda k.C(-)$. We shall not look into the contextual abort-homomorphism law here, as that is similar.

Are such hints adequate? Maybe surprisingly, they are. There are instances of control algebras, where the operations $\beta$ and $\alpha$ can be defined explicitly; moreover, those explicit definitions rephrase the operational description quite faithfully. Such instances are the *intrinsic control algebras* (Example, p66). With the set $R$ containing at least two elements (as we have assumed), the self-adjoint functor $R^{(-)}$ is monadic (a known result, not proved in this chapter); it follows that any control algebra is isomorphic to a (specific) intrinsic control algebra. In other words, any control algebra may be seen as an intrinsic one, and in that sense, $\beta$ and $\alpha$ behave as the operational description of call/cc and abort.

We have also motivated the equations in §2.3, in programming language settings.

**Comparing control algebra equations with state algebra ones** I shall make an attempt to convince the reader that, the control algebra equations we have here is a good candidate for 'the definition' of algebras of the control effect, where an operational reading is apparent. This is in much the same way as that, the three-equation definition of a lookup-update algebra may be seen as 'the definition' of algebras of the state effect, where an operational reading is apparent. We compare the two below.

Let us review the state algebras first. We again work in the category of sets and functions, and assume a set $S$ which has at least one element. We shall call $S$ the set of *states*. Then,

> A *state algebra* is a set $X$, together with two functions
>
> $$\xi\colon X^S \to X$$
> $$\zeta\colon S \times X \to X,$$
>
> such that the following three equations are satisfied (Plotkin & Power [53] came with five equations, then simplified by Melliès [73]):
>
> (i) For any $x \in X$,
> $$x = \xi\,\lambda s^{\in S}.\zeta(s,x); \qquad \text{(the unit law)}$$
>
> (ii) For any $(s,f) \in S \times X^S$,
> $$\zeta(s,\xi f) = \zeta(s,fs); \qquad \text{(update-lookup)}$$
>
> (iii) For any $\big(s,(t,x)\big) \in S \times (S \times X)$,
> $$\zeta\big(s,\zeta(t,x)\big) = \zeta(t,x). \qquad \text{(update-update)}$$

We shall call $\xi$ the *lookup* operation, and $\zeta$ the *update* operation.

The terminology 'lookup' and 'update' has an operational origin. Informally, to run a program with the state effect, we shall keep track of two pieces of information: an element taken from $S$, which shall be called the *current state*, and the program itself. The current state may change as we evaluate the program. In this programming language setting,

- The 'lookup' operation is used in a term 'lookup $M$', where the subterm $M$ expects a state as an argument. What 'lookup' does, is to make a copy of the current state, say $s$, and make it available to $M$, by feeding $s$ to it. Hence this operation 'looks up the current state'.[6]

- The 'update' operation is used in a term 'update $s, M$', where $s$ is a state, and $M$ is a subterm. What 'update' does, is to make $s$ the current state (thus altering the state), and continue with $M$.

Informally, we can read the defining equations operationally, as if $\xi$ is 'lookup', and $\zeta$ is 'update'. For example, the *update-lookup* equation reads, an update followed by a lookup, say $\zeta(s, \xi f)$, is the same as one update, but we shall feed the new state $s$ to lookup's operand subterm $f$.[7] (Similarly to the control case, we can also consider *intrinsic state algebras*, whose carrier is in the form $A^S$, where $A$ is any set. It is known that the currying adjunction $S \times (-) \vdash (-)^S$ is monadic; the monadicity implies that every state algebra is isomorphic to an intrinsic one. Hence the informal operational reading is sound.)

How do control algebra equations compare with the state algebra ones? We match up state operations and control operations in the following way:

| | state | control | |
|---|---|---|---|
| lookup | $\xi$ | $\beta$ | call/cc |
| update | $\zeta$ | $\alpha \circ \mathrm{ev}$ | throw (in terms of 'abort') |

Then we list the equations side by side.

$$\begin{array}{rr}
\text{state} & \text{control} \\
x = \xi\, \lambda s\,.\, \zeta(s, x) & x = \beta\, \lambda k\,.\, (\alpha \circ \mathrm{ev})(k, x) \\
\zeta(s, \xi f) = \zeta(s, f s) & \beta\, \lambda k\,.\, C\, \lambda N\,.\, \underline{\mathrm{ev}(k, \beta N)} = \beta\, \lambda k\,.\, C\, \lambda N\,.\, \underline{\mathrm{ev}(k, N k)} \\
\zeta\big(s, \zeta(t, x)\big) = \zeta(t, x) & \beta\, \lambda k\,.\, \psi\, \lambda r\,.\, \underline{\mathrm{ev}\big(k, (\alpha \circ \mathrm{ev})(\mathrm{id}_R, r)\big)} = \beta\, \lambda k\,.\, \psi\, \lambda r\,.\, \underline{\mathrm{ev}(\mathrm{id}_R, r)}
\end{array}$$

We have slightly paraphrased the control algebra equations so that the similarity becomes more obvious. In particular, we have been writing explicit evaluations, say $\mathrm{ev}(k, \beta N)$, rather than writing them implicit as usual, e.g., $k(\beta N)$ in this case. We have also underlined the parts in the control algebra equations that should be compared to the state counterparts. We see that,

(i) The unit laws match up precisely; a state $s$ matches up with a continuation $k$.

[6] Plotkin and Power's original setting is more general, where we may keep track of multiple states, indexed by natural numbers, say. The setting models an ideal computer's memory that consists of multiple cells, each capable of storing a state. Then the 'lookup' operation looks up the current state of a particular cell, whose index shall also be given as an argument to 'lookup'.

[7] The reader may notice a peculiarity about expressions of effect operations: the expressions are composed from inside towards outside, whereas the effect operations happen from outside towards inside. This is the case for effects in general, including state and control.

(ii) The second and third equations also match well, only that the control equations are wrapped in additional contexts, and we shall pay attention to the underlined parts. For the second equations, the state case equation describes an update followed by a lookup, whereas the control case equation describes a throw followed by a call/cc.

(iii) For the third equations, the state case equation describes consecutive updates, whereas the control case equation describes consecutive throws.

Readers may have noticed that in three occasions in the control case (two in the second equation and one in the third), we are reading a throw operation from a mere 'ev' rather than an '$\alpha \circ \mathrm{ev}$'. We have not overlooked the discrepancies; the explanation is as follows. On those occasions, the control algebra that 'ev' results in has the set $R$ as the carrier ($R$ is the parameter set we fix beforehand), and the abort operation in that case is the identity on $R$. Thus '$\alpha \circ \mathrm{ev}$' on those occasions are exactly 'ev'. Now similarity between equations of control algebras and those of state algebras should be clearly visible, and we come to the bottom line:

> If we are willing to accept the three-equation definition of state algebras as 'the definition' of the state effect, then there is a good reason to consider the definition of control algebras here as 'the definition' of the control effect, given that such control algebras equivalently rephrase the Eilenberg-Moore ones.

**An overview of the chapter** I hope the motivation has fuelled the reader with enough drive for the mathematics ahead.

- In Section 2, we define control algebras, and look at the intrinsic ones (we have mentioned them on p61). Intrinsic control algebras serve as important examples of control algebras.
- In Section 3, we consider homomorphisms between control algebras. We are interested in the special cases where the homomorphism ends in the canonical control algebra on $R$ (the result set). It is our proposal that continuations may be interpreted as such homomorphisms. Continuing from our attention to intrinsic control algebras, we look at homomorphisms from them; we would like to show that such homomorphisms are all in a special form, namely appointed evaluations (to be explained there).
- Finally, in Section 4, we prove the isomorphism theorem, i.e., the isomorphism between the category of control algebras and the category of Eilenberg-Moore algebras. The proof goes through several steps, and we shall provide a more detailed overview there.

## 2 The definition of control algebras

Let $R$ be a set. In view of programming language applications, we shall call it the *set of results.* and an element in $R$ may be called a *result*. For the moment we do not make any assumption on $R$. For example, $R$ may be empty. Later we shall assume $R$ to be not too small (having at least two elements; we will explain that), but it does not concern us now.

**Definition 1.** A *control algebra* is a set $X$ together with functions

$$\begin{aligned} \beta &: X^{R^X} \to X \\ \alpha &: R \to X \end{aligned}$$

such that the following three equations hold:

(i) For any $x \in X$,
$$x = \beta\,\lambda k^{\in R^X}.\,\alpha(kx).$$

(ii) For any $C \in X^{R^{X^{R^X}}}$ (read $X^{R^{(X^{R^X})}}$),
$$\beta\,\lambda k^{\in R^X}.C(k \circ \beta) = \beta\,\lambda k^{\in R^X}.C\,\lambda N^{\in X^{R^X}}.\,k(Nk)\,.$$

(iii) For any $\psi \in X^{R^R}$,
$$\beta\,\lambda k^{\in R^X}.\,\psi(k \circ \alpha) = \beta\,\lambda k^{\in R^X}.\,\psi\,\mathrm{id}_R\,.$$

We call $\beta$ the *call/cc* operation, and call $\alpha$ the *abort* operation; the set $X$ is called the carrier as usual. We call equation (i) the *unit law*, call equation (ii) the *contextual call/cc-homomorphism law* ('call/cc' refers to the $\beta$ inside $C$), and call equation (iii) the *contextual abort-homomorphism law* ('abort' refers to $\alpha$).[8]

The names of the equations may seem mysterious for the moment; we can only give a very brief explanation at this point. The unit law here corresponds to the unit law of an Eilenberg-Moore algebra; this shall become clear once we see how to compose $\beta$ and $\alpha$ into a structure map (of the corresponding Eilenberg-Moore algebra; Section 4). Other two laws are each wrapped in a common context: '$\beta\,\lambda k.C(-)$' in equation (ii), and '$\beta\,\lambda k.\psi(-)$' in equation (iii). If we ignore those contexts, then each law expresses a homomorphism condition for the variable $k$ (an element of $R^X$, which may be considered as a function $X \to R$). For example, $k \circ \beta = \lambda N\,.\,k(Nk)$ would express that $k$ is call/cc-homomorphic, whereas $k \circ \alpha = \mathrm{id}_R$ would express that $k$ is abort-homomorphic (we shall explain these conditions in the next section). However, due to the existence of common contexts, such homomorphism statements become contextual, hence the extra modifier.

The current notation may still be uneasy for efficient communication of the equations, either verbally or in writing. We shall introduce some helpers to make it easier.

(i) We first introduce the notation for appointed evaluations; they appear often enough that deserve a specific shorthand. Let $X$ be a set and $x \in X$. Define $\mathrm{ev}_x$ as the following function:
$$\begin{aligned}\mathrm{ev}_x \colon R^X &\to R\\ k &\mapsto kx.\end{aligned}$$

We shall read $\mathrm{ev}_x k$ as 'evaluate $k$ at $x$', or 'feed $x$ to $k$'; and we shall call all such evaluations *appointed evaluations.* For example, using appointed evaluations, the unit law above may be written as
$$x = \beta(\alpha \circ \mathrm{ev}_x).$$

This may be easier to recall for readers not interested in operational reading of the law. For convenience, we have fixed $R$ as the base in the '$\mathrm{ev}_x$' notation. From time to time, it may be useful to have appointed evaluations with other bases. In that case, we will make

[8] It may be helpful to explain our convention in expressions like $\beta\,\lambda k(\ldots)$. When a $\lambda$-expression is an argument to some function, and this argument ends the whole expression, then we tend to omit parentheses around the $\lambda$-expression, since there can be no ambiguity. Thus the unit law may also be written as $x = \beta\big(\lambda k.\alpha(kx)\big)$. It is easy to read expressions written in such convention: once we see $\beta\,\lambda\,\ldots$, we know the rest is a $\lambda$-expression, and it is an argument to $\beta$.

the base explicit by putting it as an extra subscript. For example, let $a$ be an element of some set $A$. Then $\mathrm{ev}_{X,a}$ denotes the function

$$\begin{aligned}\mathrm{ev}_{X,a} \colon X^A &\to X \\ f &\mapsto fa.\end{aligned}$$

We shall also call them appointed evaluations. For example, let $k \in R^X$, then $\mathrm{ev}_{X,k}$ is the function $\lambda N^{\in X^{R^X}}.\,Nk$, and

$$\begin{aligned}\lambda N^{\in X^{R^X}}.\,k(Nk) &= k \circ (\lambda N^{\in X^{R^X}}.\,Nk) \\ &= k \circ \mathrm{ev}_{X,k}\,.\end{aligned}$$

Thus the equation in the contextual call/cc-homomorphism law may be written as

$$\beta\,\lambda k^{\in R^X}.\,C(k \circ \beta) = \beta\,\lambda k^{\in R^X}.\,C(k \circ \mathrm{ev}_{X,k})\,.$$

(ii) The second helper aims to ease communication of the two contextual laws; it does so by separating the common contexts from the equations. The general idea is as follows: suppose that we have some equation

$$Cf = Cg\,;$$

then we may say, *under* $C$, $f$ is the same as $g$; or we may write, *under* $C$,

$$f = g\,.$$

In actual use, we allow $f$ and $g$ to take a parameter which is bound in the context $C$. This is better given by examples. For instance, the equation in the contextual call/cc-homomorphism law may be written as, under $\beta\,\lambda k^{\in R^X}.\,C(-)$,

$$k \circ \beta = k \circ \mathrm{ev}_{X,k}\,.$$

Here the context is $\beta\,\lambda k^{\in R^X}.\,C(-)$, and it binds $k$, which is referred to by the two sides of the equation. Similarly, the equation in the contextual abort-homomorphism law may be written as, under $\beta\,\lambda k^{\in R^X}.\,\psi(-)$,

$$k \circ \alpha = \mathrm{id}_R\,.$$

We shall refer to an equation expressed in this 'under ...' convention the *colloquial form* of the original equation.

Let us list the laws again with helpers applied.

A control algebra $(X, \beta, \alpha)$ satisfies the following equations:

(i) For any $x \in X$,

$$x = \beta(\alpha \circ \mathrm{ev}_x). \qquad \text{(the unit law)}$$

(ii) For any $C \in X^{R^{X^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\,C(-)$,

$$k \circ \beta = k \circ \mathrm{ev}_{X,k}\,. \qquad \text{(the contextual call/cc-homomorphism law)}$$

(iii) For any $\psi \in X^{R^R}$, and under $\beta\, \lambda k^{\in R^X} . \psi(-)$ ,

$$k \circ \alpha = \mathrm{id}_R \, . \qquad \text{(the contextual abort-homomorphism law)}$$

The new formulation should be less challenging to fit into the mind. Although we will not use colloquial forms in calculations, communicating with them shall be more convenient, as they make the structure of the equations apparent.

Let us look at some concrete control algebras. The following control algebras we shall introduce are *intrinsic control algebras*[9]. As explained in the chapter introduction (p61, 'Are such hints adequate?'), in the case that monadicity of the self-adjoint functor $R^{(-)}$ holds, every control algebra is isomorphic to some (specific) intrinsic control algebra. As we usually work with a big-enough $R$ so that the mentioned monadicity holds, we may consider intrinsic control algebras as all control algebras.[10]

**Example.** Let $A$ be a set. Define functions $\beta_A$ , $\alpha_A$ as follows.

$$\begin{aligned} \beta_A \colon (R^A)^{R^{R^A}} &\to R^A \\ N &\mapsto \lambda a^{\in A} . \mathrm{ev}_a(N \, \mathrm{ev}_a) \, , \\ \alpha_A \colon R &\to R^A \\ r &\mapsto \lambda a^{\in A} . r \, . \end{aligned}$$

In the definition of $\alpha_A$, $\lambda a . r \in R^A$ may be seen as the constant function $A \to R$ whose value is always $r$. This may also be written as $\mathrm{const}_r$ .[11] We can verify that

*$\beta_A$ and $\alpha_A$ satisfy control algebra equations on carrier $R^A$.*

These are routine calculations: for the contextual call/cc-homomorphism law, both sides reduce to

$$\lambda a^{\in A} . C \left( \lambda N^{\in (R^A)^{R^{R^A}}} . N \, \mathrm{ev}_a \, a \right) a \in R^A \, ;$$

for the contextual abort-homomorphism law, both sides are equal to $\psi \, \mathrm{id}_R \in R^A$. For any set $A$, we shall call the control algebra $(R^A, \beta_A, \alpha_A)$ the *intrinsic control algebra* in $A$, and simply write it by the carrier $R^A$.

(*End of Example*)

In the Chapter introduction, we have mentioned that the explicit definitions of intrinsic control algebras rephrase the operational description quite faithfully. It needs little extra explanation to see that. We shall explain, that for any $a \in A$, $\mathrm{ev}_a$ is a control algebra homomorphism, from the intrinsic control algebra in $A$, to the canonical control algebra on $R$ (to be explained in the next section); And, by the homomorphism principle ('Continuations as homomorphisms', p44; rephrased as a definition by Definition 3, p72), $\mathrm{ev}_a$ shall also be understood as a continuation

[9] My terminology. Such control algebras (or algebras of the continuation monad) don't seem to receive a name in the literature.

[10] To make it easier to refer to, we shall also call the adjunction formed by the self-adjoint $R^{(-)}$ the *swapping* adjunction. To see the choice of the name, consider a function $f \colon X \to R^Y$. Under the adjunction, the adjunct of $f$ is a function $g \colon Y \to R^X$, and $g(y)(x) = f(x)(y)$; hence the name 'swap'. This convenient name is on a par with the terminology 'currying adjunction' for the adjunction $S \times (-) \dashv (-)^S$.

[11] We suppress mentioning of the domain $A$ in $\mathrm{const}_r$, to make the notation simpler. This should be acceptable as any argument to the constant function is discarded.

from the intrinsic control algebra in $A$. Also, under a mild condition, $\{\mathrm{ev}_a \mid a \in A\}$ are all homomorphisms from the intrinsic control algebra in $A$, to the canonical control algebra on $R$; hence they are all continuations from the intrinsic control algebra.[12] Thus $\beta_A N$ behaves in the way of 'call/cc $N$':

- Suppose the current continuation is $\mathrm{ev}_a$ (the outer occurrence); then $\beta_A$ makes a copy of it, and feeds it to $N$. The result of the computation is obtained by applying the current continuation to that intermediate value.

For the $\alpha_A$ case, we need to be aware of the following fact (also to be explained later): assuming monadicity of the swapping adjunction, for any set $A$, we have the isomorphism

$$A \cong \mathrm{hom}(R^A, R).$$

By $\mathrm{hom}(R^A, R)$, we mean all homomorphisms from the intrinsic control algebra in $A$ to the canonical algebra on $R$. Operationally, the homomorphisms should be understood as continuations from the intrinsic control algebra in $A$. With that explained, we shall see that $\alpha_A r$ behaves as 'abort $r$':

- Suppose the current continuation is a continuation that corresponds to $a \in A$. Assuming the same mild condition as above, we may say the current continuation is $\mathrm{ev}_a$. Then we consider the result of the computation, $\mathrm{ev}_a(\alpha_A r)$.

  $$\mathrm{ev}_a(\alpha_A r) = \alpha_A r a = \mathrm{const}_r a = r\,,$$

  essentially discarding the current continuation.

The existence of examples assures us that the theory of control algebras is a meaningful one. Of course we still need the isomorphism theorem to be sure that the theory is correct. (By the isomorphism theorem, we mean the isomorphism between the category of control algebras and the Eilenberg-Moore category of the continuation monad.) We shall note that intrinsic control algebras are not merely some useful examples of control algebras; with a suitable condition on the result set $R$, they are essentially all control algebras. To be more precise, when $R$ has at least two elements, every control algebra is isomorphic to an intrinsic one. This is given by monadicity of $R^{(-)}$ and the isomorphism theorem we have just mentioned. The fact means, once we establish the monadicity of $R^{(-)}$ for a suitable $R$, intrinsic control algebras may serve as a way to verify candidate equations. For example, suppose we encounter the diagonalisation principle (p87) but we are uncertain about its validity. Then we may check it using intrinsic control algebras (Exercise, p84). As the result is positive, we know that for a not-too-small $R$, the diagonalisation principle holds for any control algebra.

Readers familiar with category theory may see that intrinsic control algebras come from the comparison functor, from the swapping adjunction. Put differently, under the isomorphism theorem (between the category of control algebras and Eilenberg-Moore algebras), an intrinsic control algebra in $A$ corresponds to an Eilenberg-Moore algebra on $R^A$; that Eilenberg-Moore algebra can be obtained by applying the comparison functor to the set $A$. We shall not elaborate on that here, but leave it till we present the definition of the isomorphisms (p76, Exercise).

[12] We shall explain this in the next section. For readers not interested in operational descriptions, they only need to pay attention to homomorphisms.

# 3 Homomorphisms between control algebras

Any study of algebraic structures needs to consider homomorphisms. In our case, we will show that the category of control algebras are isomorphic to the category of Eilenberg-Moore algebras (Section 4); in this aspect, control algebra homomorphisms are indispensable. However, for our mathematical exploration, there is not much we can say about general control algebra homomorphisms. For any algebraic structure defined by a carrier and some operations, the definition of homomorphisms can be obtained mechanically: we consider corresponding operations of domain and codomain algebras, and for each operation symbol, require commuting of an obvious square, formed by those operations and the candidate homomorphism. We shall do so below (Definition 2); and, to make it more useful for our setting (sets and functions), we shall carry out some easy calculation to obtain a more explicit formulation (p69).

Besides general definitions, what shall we pay attention to? We shall note that the result set $R$ carries a canonical control algebra, and pay attention to control algebra homomorphisms to it (Theorem 1, 2). Recall that we propose the 'homomorphism principle' in Chapter 2 ('Continuations as homomorphisms', p44):

> Semantically, continuations shall be defined as homomorphisms to the canonical control algebra on $R$.

We shall formulate these homomorphisms explicitly (Theorem 2). Then, to see concrete examples, we consider such homomorphisms from intrinsic control algebras. We shall see that when $R$ has at least two elements, such homomorphisms are exactly appointed evaluations (Theorem 3, 4, 5). From this incidence, we shall see that in the case of intrinsic control algebras, the homomorphism principle and semantic instances of continuations agree (p72). Given the significance of homomorphisms to the canonical control algebra on $R$, we shall give them a name; in view of the homomorphism principle, we propose to name them 'continuations' (Definition 3).

We first consider general control algebra homomorphisms. Following the general idea in defining homomorphisms, we define homomorphisms between control algebras as follows.

**Definition 2.** Let $(X, \beta, \alpha)$, $(Y, \xi, \zeta)$ be control algebras. We say a function $h\colon X \to Y$ is

(i) *Call/cc-homomorphic*, if diagram (i) commutes;

(ii) *Abort-homomorphic*, if diagram (ii) commutes.

$$\begin{tikzcd}
X^{R^X} \arrow[r, "h^{R^h}"] \arrow[d, "\beta"'] & Y^{R^Y} \arrow[d, "\xi"] \\
X \arrow[r, "h"] & Y
\end{tikzcd}$$

(i)

$$\begin{tikzcd}
& R \arrow[dl, "\alpha"'] \arrow[dr, "\zeta"] & \\
X \arrow[rr, "h"] & & Y
\end{tikzcd}$$

(ii)

We say $h$ is *homomorphic*, or a *homomorphism* from $(X, \beta, \alpha)$ to $(Y, \xi, \zeta)$, if it is both call/cc- and abort-homomorphic.

Since we work in the category of sets and functions, we can give a more concrete formulation of the function $h^{R^h} : X^{R^X} \to Y^{R^Y}$ (the top morphism in diagram (i)). For any $N \in X^{R^X}$, we may consider it as a function $R^X \to X$. Then $h^{R^h}N$ is an element of $Y^{R^Y}$, which may be considered as a function $R^Y \to Y$. The expression $h^{R^h}N$ can be equivalently written as

$$\lambda j^{\in R^Y}.(h \circ N)(j \circ h).$$

(The $h$ in $h \circ N$ is the one at the base, whereas the $h$ in $j \circ h$ is the one in the exponent.[13]) To see this, we calculate

$$\begin{aligned} h^{R^h} N &= h \circ (N \circ R^h) \\ &= h \circ \lambda j^{\in R^Y} . N(R^h j) \\ &= h \circ \lambda j^{\in R^Y} . N(j \circ h) \\ &= \lambda j^{\in R^Y} . (h \circ N)(j \circ h). \end{aligned}$$

(The calculation is nothing more than the definition of the post-composition $g^{(-)}$, the pre-composition $(-)^f$ and, for the second and the last step, the $\lambda$-notation. If the reader finds the steps on pre- and post-compositions unclear, then the exercise on p5 may be useful.) With the concrete formulation of $h^{R^h}$, we may rephrase the definition of homomorphisms as follows.

> A *control algebra homomorphism* from $(X, \beta, \alpha)$ to $(Y, \xi, \zeta)$ is a function $h \colon X \to Y$ such that
>
> (i) For any $N \in X^{R^X}$,
>
> $$h(\beta N) = \xi \lambda j^{\in R^Y} . (h \circ N)(j \circ h) \, ; \qquad \text{(call/cc-homomorphic)}$$
>
> (ii) For any $r \in R$,
>
> $$h(\alpha r) = \zeta r \, . \qquad \text{(abort-homomorphic)}$$

As said in the section overview, we next introduce the canonical control algebra on $R$, and consider homomorphisms to it.

**Theorem 1.** *The result set $R$ together with functions*

$$\begin{aligned} \mathrm{ev}_{\mathrm{id}_R} \colon R^{R^R} &\to R \\ \mathrm{id}_R \colon R &\to R \end{aligned}$$

*form a control algebra.* □

We shall call it the *canonical control algebra* on $R$, or simply the *result set algebra* (so that we do not repeat $R$ too often in certain contexts). We may simply refer to the control algebra by $R$, if it is clear that we are referring to a control algebra.

In the above concrete formulation of homomorphisms, we take the control algebra $(Y, \xi, \zeta)$ to be the result set algebra, i.e., $(R, \mathrm{ev}_{\mathrm{id}_R}, \mathrm{id}_R)$, then we obtain the following formulation of homomorphisms in this special case.

**Theorem 2.** *Let $(X, \beta, \alpha)$ be a control algebra. A function $k \colon X \to R$ is a homomorphism from that control algebra to the result set algebra if and only if*

(i) *For any $N \in X^{R^X}$,*

$$k(\beta N) = k(Nk),$$

(ii) *And, for any $r \in R$,*

$$k(\alpha r) = r \, .$$

[13] Generally, we may say that $g^{R^f} N$ 'post-composes $g$ to $N$, and applies that to the argument pre-composed with $f$'. We omit the precise set-up of the statement and the calculation, but the calculation would be quite similar to that of $h^{R^h} N$ below.

*Proof.* This is a routine calculation. We calculate that

$$\begin{aligned}\xi\,\lambda j^{\in R^Y}.\,(k\circ N)(j\circ k) &= \mathrm{ev}_{\mathrm{id}_R}\,\lambda j^{\in R^R}.\,(k\circ N)(j\circ k)\\ &= (k\circ N)k = k(Nk).\end{aligned}\qquad\square$$

The two equations in the condition can be equivalently formulated in a point-free style, i.e., $k\colon X\to R$ is a homomorphism if

$$\begin{cases} k\circ\beta = k\circ\mathrm{ev}_{X,k}\\ k\circ\alpha = \mathrm{id}_R\,. \end{cases}$$

(The definition of $\mathrm{ev}_{X,k}$ is explained in helper (i), in simplification of control algebra equations, p64f.) We can compare the formulation in Theorem 2 with the two contextual homomorphism laws in the original definition (Definition 1, p63f.), and compare the point-free style formulation above with the simplified contextual homomorphism laws (p65). It shall be apparent that those laws are indeed homomorphism laws put into contexts, hence we name them 'contextual call/cc- and abort-homomorphism laws'.

As in the explanation of control algebras, we turn to examples after considering general definitions. In particular, we continue with the interest in intrinsic control algebras, and look at homomorphisms from them to the result set algebra.

**Theorem 3.** *Let $A$ be a set. For any $a\in A$, the appointed evaluation*

$$\mathrm{ev}_a\colon R^A\to R$$

*is a homomorphism from the intrinsic control algebra in $A$ to the result set algebra.*

*Proof.* Let $a\in A$, and recall the definition of the intrinsic control algebra in $A$, namely $(R^A,\beta_A,\alpha_A)$ (Example, p66). By routine calculations we verify that

$$\begin{aligned}\mathrm{ev}_a\circ\beta_A &= \mathrm{ev}_a\circ\mathrm{ev}_{R^A,\mathrm{ev}_a}\\ \mathrm{ev}_a\circ\alpha_A &= \mathrm{id}_R\,.\end{aligned}$$

For the first equation, for any $N\in(R^A)^{R^{R^A}}$, functions on both sides maps $N$ to $N\,\mathrm{ev}_a\,a$. □

The converse also holds, if we assume that $R$ has at least two elements, or the set $A$ has at least one element. We are interested in the former condition; however, our plan is to first verify the latter, then point out that, when $R$ has two elements but $A$ is empty, there is no homomorphisms as stated. To show the converse when $A$ has an element, we need to argue indirectly.[14]

**Theorem 4.** *Let $A$ be a set with at least one element. Any homomorphism from the intrinsic control algebra in $A$ to the result set algebra is an appointed evaluation, i.e.,* $\mathrm{ev}_a\colon R^A\to R$ *for some $a\in A$.*

*Proof.* A homomorphism as stated is a function $R^A\to R$. Suppose some function $k\colon R^A\to R$ is not an appointed evaluation; we shall show that it is not a homomorphism from the intrinsic control algebra in $A$, to the result set algebra. As $A$ has at least one element, we may take some

[14] In view of potential categorical generalisation, I am more interested in proofs that are calculational in nature. Hence I try to mark a classical proof if I am aware of it (although I don't aim to be comprehensive). For the purpose of developing a semantic theory, a place where classical proofs seem unavoidable is a sign that we may need a definition.

$a \in A$ (arbitrarily). By definition, a function $f\colon R^A \to R$ is an appointed evaluation if and only if, for some $a_0 \in A$,

$$f c = c a_0 \text{ for any } c \in R^A.$$

Since $k$ is not an appointed evaluation, then for the previously chosen $a \in A$, we may take some $d \in R^A$, such that

$$k d \neq d a \in R\,.$$

We take a note of $d$ and $a$ for later use.

Suppose in addition, $k$ is abort-homomorphic; we shall show that $k$ cannot be call/cc-homomorphic. That is, we claim that there is an $N \in (R^A)^{R^{R^A}}$, such that

$$k(\beta_A N) \neq k(N k)\,.$$

($\beta_A$ is the call/cc operation of the intrinsic control algebra in $A$.) With that claim in mind, we define a functional $N\colon R^{R^A} \to R^A$ as follows, using excluded middle. $N$ takes functions $R^A \to R$ as input; depending on whether the input is an appointed evaluation, the functional gives different values in $R^A$, which are carefully crafted with the $a \in A$ and $d \in R^A$ we have previously chosen:

$$\begin{aligned} N\colon R^{R^A} &\to R^A \\ h &\mapsto \begin{cases} \lambda z^{\in A}.\, d a & \text{if } h\colon R^A \to R \text{ is an appointed evaluation,} \\ d & \text{otherwise.} \end{cases} \end{aligned}$$

Then we have

$$\begin{cases} N\,\mathrm{ev}_b\, b = d a \quad \text{for any } b \in A, & (\mathrm{ev}_b \text{ is an appointed evaluation}) \\ N k = d. & (\text{whereas } k \text{ is not}) \end{cases}$$

Using the first case, we calculate

$$\begin{aligned} k(\beta_A N) &= k\, \lambda b^{\in A}.\, \mathrm{ev}_b(N\,\mathrm{ev}_b) && \text{(a)} \\ &= k\, \lambda b^{\in A}.\, d a && (N\,\mathrm{ev}_b\, b = d a) \\ &= d a\,. && \text{(b)} \end{aligned}$$

Step (a) applies the definition of $\beta_A$; Step (b) applies the abort-homomorphism condition for $k$: for any result $r \in R$ ($da$ in this case),

$$k(\alpha_A r) = r \quad \text{i.e.,} \quad k(\lambda b^{\in A}.\, r) = r\,.$$

On the other hand, we have the second case $N k = d$. Thus

$$\begin{cases} k(\beta_A N) = d a \\ k(N k) = k d\,. \end{cases}$$

By the choice of $a$ and $d$, $k d \neq d a$, so

$$k(\beta_A N) \neq k(N k),$$

i.e., $k$ is not call/cc-homomorphic. We conclude that if $k$ is not an appointed evaluation, then it cannot be a homomorphism from the intrinsic control algebra in $A$, to the result set algebra. □

Now let us add the corner case. Suppose $R$ has at least two elements, but $A$ is empty; then any function $k\colon R^A \to R$ cannot be homomorphic. This is because in this case, $R^A$ becomes a singleton, and $k \in R^{R^A} \cong R^1$ may be considered as a function $1 \to R$; such a function singles out one particular element in $R$. In that case, the abort-homomorphism condition

$$k\,\lambda a^{\in A}.r = r \quad \text{for any } r \in R$$

can never be satisfied: the left hand side can only take one fixed value in $R$, whereas $r$ on the right hand side may vary. With the above argument added, we have the following conditional converse to Theorem 3, as stated earlier.

**Theorem 5.** *Let $R$ be a set with at least two elements. Any homomorphism from an intrinsic control algebra to the result set algebra is an appointed evaluation.*

Why have we put some effort into these homomorphisms? Apparent reasons are that these homomorphisms are related to intrinsic control algebras, our main examples of control algebras; that these homomorphisms are concrete, where we can write down expressions for them; and that such expressions are rather special, i.e., they are appointed evaluations. One other reason is, we can see how the 'homomorphism principle' fits in the case of intrinsic control algebras (the principle is introduced in 'Continuations as homomorphisms', p44ff.). Since intrinsic control algebras are representative (in the informal sense), looking at these homomorphisms shall add some weight to the homomorphism principle in general.[15]

Recall that the homomorphism principle states that, semantically, a continuation should be identified as a homomorphism, from a control algebra to the result set algebra. In our case here, the domain control algebras are the intrinsic ones. If we write $\hom(R^A, R)$ for homomorphisms from the intrinsic control algebra in $A$ to the result set algebra, then our analysis around the last few theorems show that, for a big enough $R$, we have the identity

$$\hom(R^A, R) = \{\mathrm{ev}_a \mid a \in A\}$$

i.e., between the homomorphisms we are interested in, and the relevant appointed evaluations. On one hand, the homomorphism principle states that continuations from $R^A$ should be the hom to the left, whereas on the other hand, our operational reading of intrinsic control algebra operations suggests, that continuations from $R^A$ should be the appointed evaluations to the right (see p66 for the operational reading). Thus the homomorphism principle and the operational reading do agree in this case.

Regardless of my justification to the homomorphism principle, the set of all homomorphisms from a control algebra to the result set algebra is an important one. In particular, these sets give the inverse to the comparison functor, in the argument of monadicity of $R^{(-)}$ (we shall see that in the monadicity chapter). For that reason, homomorphisms to the result set algebra deserve a name. In view of the homomorphism principle, I propose to call them 'continuations'.

**Definition 3.** A *continuation* is a control algebra homomorphism to the result set algebra.

Historically, continuations originated as a semantic notion (Strachey & Wadsworth 1974 [7], see Reynolds' essay [32]), though the terminology was (and still often is) used in a more

[15] Ultimately, the mathematical justification should be, we interpret continuations defined in Chapter 2 with these homomorphisms, and prove adequacy for the languages there. However, as I have gained experience overtime, for example in the call-by-push-value language of Levy, I have convinced myself the principle is reasonable; and since there are other aspects of the theory that are of higher priority than carrying out that argument in all the details, I have not walked through the argument myself. As suggested between the lines, I expect no surprise here.

permissible way.[16] Syntactic definitions of continuations (like those in Chapter 2) came later (e.g., Felleisen & Friedman 1986 [16]). For that reason, I feel that we do not need extra modifiers before 'continuation' in the above definition. In case we need to distinguish the two kinds of continuations, we may qualify them by 'syntactic' or 'semantic' respectively.

Although I find the terminology here well-supported, readers should note that it deviates from the literature: although some writers use the terminology in a more permissive way (as mentioned in footnote 16), others may use it in a more specific way. For example, in Levy's call-by-push-value monograph [56], continuations are defined as homomorphisms from a free algebra.

# 4 The isomorphism theorem

We shall establish the isomorphism theorem, i.e.,

> *The category of control algebras is isomorphic to the Eilenberg-Moore category of the continuation monad.* (p81)

It is by this theorem that we know we get control algebras right. In order to stay on the main narrative, we choose to leave a gap in this section (the special contextual abort-homomorphism law, p78); it will be filled-in in the next section. To establish the isomorphism, we need constructions in both directions:

$$\begin{tikzcd}\begin{array}{c}\text{the category of}\\ \text{control algebras}\\ (X,\beta,\alpha)\end{array} \arrow[r, shift left, "\text{compose}"] & \begin{array}{c}\text{the category of}\\ \text{algebras of the}\\ \text{continuation monad}\\ \text{e.g., } (X,\theta)\,.\end{array} \arrow[l, shift left, "\text{decompose}"]\end{tikzcd}$$

We shall emphasise that constructions in either direction preserve the carrier; in other words, the constructions are on structures only:[17]

- From a control algebra, using the operations $\beta$, $\alpha$, we can obtain the structure map $\theta$ of the corresponding monad algebra. We call this construction the *compose* construction, i.e., composing $\beta$ and $\alpha$ to obtain $\theta$ (Definition 4(i), p75).
- Conversely, from an algebra of the continuation monad, using the structure map $\theta$, we can construction the two operations of the corresponding control algebra, say $\beta$ and $\alpha$. We call the two constructions in this direction the *decompose* constructions, i.e., decompose $\theta$ to $\beta$ and $\alpha$ (Definition 4(ii)).

A notable fact about the isomorphisms is that they are simple constructions that can be formulated in elementary operations (by that we mean function compositions, natural transformations that can be written as $\lambda$-expressions). As such, their definitions do not make use of equations, of control algebras or algebras of the continuation monad. In other words, they can be defined between unrestricted structures, i.e., *magmas.* To be precise, we say

- A *control magma* is a set $X$ together with two functions

$$\beta\colon X^{R^X} \to X, \quad \alpha\colon R \to X.$$

[16] Whereas we are more explicit, and restrict to homomorphisms from a control algebra $(X,\beta,\alpha)$ to the result set algebra, writers usually just say 'a continuation from $X$ to $R$', etc.. It should be obvious that I am certainly not complaining about the historical use.

[17] Some may say the isomorphism is between concrete categories.

We shall call them the pseudo call/cc operation and the pseudo abort operation for convenience. The modifier 'pseudo' is only added to remind ourselves that the functions may not satisfy the control algebra laws.

- A *magma of the continuation monad* is a set $X$ together with a function
$$\theta \colon R^{R^X} \to X.$$
We shall call it the pseudo structure map.

Our plan is as follows.

$$\begin{tikzcd}
(4) & \mathrm{CtAlg} \arrow[r, shift left, "(2)"] \arrow[d, phantom, sloped, "\subseteq"] & \mathrm{Alg} \arrow[l, shift left, "(3)"] \arrow[d, phantom, sloped, "\subseteq"] & (5) \\
& \mathrm{CtMag} \arrow[r, shift left, "\mathrm{comps}"] & \mathrm{Mag} \arrow[l, shift left, "\mathrm{decomp}"] & \\
& {} \arrow[r, phantom, "(1)"] & {} &
\end{tikzcd}$$

(1) Define the compose and decompose constructions for magmas (Definition 4). In the diagram, 'CtMag' is the category of control magmas, and 'Mag' is the category of magmas of the continuation monad. (It is easy to see that they are categories; we will also explain that on p76.)

(2) Restrict the compose construction to algebras (Theorem 7, p78). We shall assume the special contextual abort-homomorphism law (Theorem, p78).

(3) Restrict the decompose construction to algebras (Theorem 9, p79).

(4) Show that the compose-decompose round trip on control algebras is an identity (Theorem 10, p80).

(5) Establish the isomorphism theorem, by showing that the decompose-compose round trip on algebras of the continuation monad is also an identity (p81).

We shall define the constructions below. Note that control magmas form a category that includes the category of control algebras, and similarly for magmas of the continuation monad; also, the constructions we shall define are actually functors. Nevertheless, for the definition below, we do not need to consider the categorical structure. We may simply consider control magmas as a class, and so for magmas of the continuation monad; the constructions are defined as class functions between them. After that, we shall add the categorical structure, and arrive at the following picture:

$$\begin{tikzcd}
\mathrm{CtMag} \arrow[r, shift left, "\mathrm{comps}"] & \mathrm{Mag} \arrow[l, shift left, "\mathrm{decomp}"]
\end{tikzcd}$$

i.e., the category of control magmas ('CtMag'), the category of magmas of the continuation monad ('Mag'), and two functors between them (the compose construction 'comps', and the decompose construction 'decomp'). After that, we will restrict the constructions to algebras.

**Definition 4.** The *compose* and *decompose* constructions are mappings between control magmas and magmas of the continuation monad, defined as follows.

(i) (Control magmas → magmas of the continuation monad) Let $(X, \beta, \alpha)$ be a control magma, i.e., a set $X$ together with functions

$$\beta \colon X^{R^X} \to X, \quad \alpha \colon R \to X.$$

We can define a function

$$\beta \circ \alpha^{R^X} = \beta\big(\alpha \circ (-)\big) \colon R^{R^X} \to X \tag{3.1}$$

($\alpha^{R^X} : R^{R^X} \to X^{R^X}$ is the function that post-composes $\alpha$); for any $\phi \in R^{R^X}$, the function produces

$$(\beta \circ \alpha^{R^X})\phi = \beta(\alpha \circ \phi) = \beta\, \lambda k^{\in R^X}.\, \alpha(\phi k)\,.$$

Then $(X, \beta \circ \alpha^{R^X})$ is a magma of the continuation monad. We call this construction the *compose* construction.

(ii) (Magmas of the continuation monad → control magmas) Conversely, let $(X, \theta)$ be a magma of the continuation monad, i.e., a set $X$ together with a function

$$\theta \colon R^{R^X} \to X.$$

We can define two functions,

$$\begin{array}{lrl} \theta\, \lambda k^{\in R^X}.\, k\big((-)k\big) \colon X^{R^X} \to X & \text{i.e.} & N \mapsto \theta\, \lambda k^{\in R^X}.\, k(Nk) \\ \theta\, \lambda k^{\in R^X}.\, (-) \colon \qquad\qquad R \to X & \text{i.e.} & r \mapsto \theta\, \lambda k^{\in R^X}.\, r\,. \end{array}$$

Then $X$ together with the above two functions form a control magma. We call this construction the *decompose* construction.

(*End of definition*)

In formulating the decompose construction, we stretch the notation a little too much, and the formulation may seem more complicated than it actually is. For the first function (the pseudo call/cc operation), we may separate the following function to make the formulation easier.

For any set $X$, we define a function

$$\begin{aligned} \mathrm{drinker}_X \colon X^{R^X} &\to R^{R^X} \\ N &\mapsto \lambda k^{\in R^X}.\, k(Nk)\,. \end{aligned}$$

(This is a component of the monad morphism from the selection monad to the continuation monad, see [71]; the name ‘drinker’ is taken from the ‘drinker sentence’ in logic.)

With the drinker function, the first function may be written as $\theta \circ \mathrm{drinker}_X$. In the second function of the decompose construction (the pseudo abort operation), $\lambda k.r$ is the constant function $\mathrm{const}_r$ (see p63, the example for its definition). Thus we can write the second function as $\theta \circ \mathrm{const}$.[18] With the new notations, we may formulate the decompose construction as follows:

[18] By convention, we usually write $\mathrm{const}_r$ rather than $\mathrm{const}(r)$

(ii) For any magma $(X, \theta)$ of the continuation monad,

$$(X,\, \theta \circ \mathrm{drinker}_X\,,\, \theta \circ \mathrm{const}) \tag{3.2}$$

is a control magma. We call the construction the *decompose* construction.

Now that the constructions are clear, impatient readers may be ready to prove the isomorphism theorem themselves;[19] as mentioned in the section introduction, before they start, they may want to look at the statement of the *special contextual abort-homomorphism law* (the unnumbered theorem on p78), or the *diagonalisation principle* (p87).

This is also a good time that we fulfil a promise made earlier. Towards the end of Section 2, we mention that intrinsic control algebras can be obtained from the comparison functor (p67). We can now verify that. We give it as an exercise.

**Exercise.** Intrinsic control algebras may be obtained from the comparison functor, of the self-adjoint $R^{(-)}$, as follows.

(i) Recall the general definition of the comparison functor. That is, given an adjunction with the right adjoint $U$, show that for any object $A$ in the left category (the domain of $U$), $(UA, U\varepsilon_A)$ is an Eilenberg-Moore algebra of the induced monad. Here $\varepsilon_A$ is the counit of the adjunction at the object $A$.

(ii) Instantiate the general definition to the self-adjoint $R^{(-)}$. Note that the left category is the opposite category. What is the counit here?

(iii) Apply the decompose construction defined above to the algebra obtained from (ii). Verify that it is the intrinsic control algebra in $A$, where $A$ is the object in the left category that we have started with.

As we have said before defining the constructions, control magmas form a category, so do magmas of the continuation monad. Morphisms in each category are homomorphisms between the magmas; we shall say a few words here. We note that for algebraic structures, the definition of homomorphisms only make use of the operations, but not equations;[20] for that reason, the definition of homomorphisms between algebras are actually that between magmas. Thus we already have the definition of homomorphisms between control magmas, and between magmas of the continuation monad; among those homomorphisms between magmas, there are identities, and homomorphisms compose. That makes control magmas a category, as well as magmas of the continuation monad. Then we consider the two constructions. Routine calculations will show that each construction preserves homomorphisms; so both constructions extend to functors between the magma categories. Thus we arrive at the diagram mentioned previously:

$$\begin{tikzcd}\mathrm{CtMag} \arrow[r, shift left, "\mathrm{comps}"] & \mathrm{Mag} \arrow[l, shift left, "\mathrm{decomp}"]\end{tikzcd}$$

Our next step is to restrict the constructions to algebras (Theorem 7, Theorem 9), so that we arrive at the following diagram:

$$\begin{tikzcd}\mathrm{CtAlg} \arrow[r, shift left, "\mathrm{comps}"] & \mathrm{Alg.} \arrow[l, shift left, "\mathrm{decomp}"]\end{tikzcd}$$

[19] With pen and paper, or their favourite proof assistant.

[20] One may say homomorphisms are only about the structures, but not properties on them.

Here ‘CtAlg’ is the category of control algebras, and ‘Alg’ is the category of algebras of the continuation monad. After that, we shall show that the two round-trips are each an identity on the respective algebras, so that we have the isomorphism. Thus we need to carry out four arguments, i.e., two restrictions and proofs of the two identities. All of them are given as calculations, but the calculation for the first is more subtle (‘given a control algebra, the compose construction produces an algebra of the continuation monad’). We shall point out that subtlety first (Theorem, p78), but take it for granted; then we carry out all four arguments and prove the isomorphism theorem. When all is done, we come back to concentrate on the subtlety. That way we stay oriented when working towards the main proof, and better concentrate on the subtlety later.

We now work towards the main proof. We first consider restricting the compose construction to algebras, i.e., we shall show that given a control algebra, the compose construction produces an algebra of the continuation monad. Recall the three defining laws of a control algebra $(X, \beta, \alpha)$ (p65):

*The unit law* For any $x \in X$,

$$x = \beta(\alpha \circ \mathrm{ev}_x).$$

*The contextual call/cc-homomorphism law* For any $C \in X^{R^{X^{R^X}}}$, and under $\beta\, \lambda k^{\in R^X}. C(-)$,

$$k \circ \beta = k \circ \mathrm{ev}_{X,k}\,.$$

*The contextual abort-homomorphism law* For any $\psi \in X^{R^R}$, and under $\beta\, \lambda k^{\in R^X}. \psi(-)\,$,

$$k \circ \alpha = \mathrm{id}_R\,.$$

We shall also recall the defining laws of an algebra of the continuation monad. Let $(X, \theta)$ be such an algebra ($\theta$ is a function $R^{R^X} \to X$); then, the *unit law* is, for any $x \in X$,

$$x = \theta\, \mathrm{ev}_x\,,$$

and the *multiplication law* is, for any $\Phi \in R^{R^{R^{R^X}}}$, and under $\theta\, \lambda k^{\in R^X}. \Phi(-)$,

$$k \circ \theta = \mathrm{ev}_k.$$

To verify that the compose construction produces continuation monad algebras, we need to establish the unit law and the multiplication law of a monad algebra. The unit law is trivial: suppose $\theta = \beta\big(\alpha \circ (-)\big)$ is the function given by the compose construction, then the monad unit law

$$x = \theta\, \mathrm{ev}_x \quad \text{is exactly} \quad x = \beta(\alpha \circ \mathrm{ev}_x),$$

the unit law of the control algebra. The multiplication law needs more effort. The non-trivial step is to establish the following equation for control algebras.

**The special contextual abort-homomorphism law.** Let $(X, \beta, \alpha)$ be a control magma. We say it satisfies the *special contextual abort-homomorphism law*, if for any $D \in X^{R^{R^{R^X}}}$, and under $\beta\, \lambda k^{\in R^X}. D\big((-) \circ \mathrm{ev}_k\big)$,

$$k \circ \alpha = \mathrm{id}_R.$$

Here $\mathrm{ev}_k$ is an appointed evaluation $R^{R^X} \to R$, as defined before (p64, the helper (i)). It takes some effort to establish this law; we shall take it for granted for now, and come back after we finish explaining the isomorphism theorem. The only difference between this special variant and the original contextual abort-homomorphism law is in the context. In the original law (p66, (iii)), the context is $\beta\,\lambda k^{\in R^X}.\,\psi(-)$, with $\psi$ being any element of $X^{R^R}$; here, $\psi(-)$ becomes $D\big((-)\circ \mathrm{ev}_k\big)$. At first glance, this new variant may be mistaken as an instance of the original law, but it is not. The issue is the occurrence of $k$ in $D\big((-)\circ \mathrm{ev}_k\big)$. Suppose we attempt to instantiate the original contextual abort-homomorphism law, in order to obtain the special variant; the first thing we shall do is to define $\psi$. We immediately run into trouble here: if we let it be $D\big((-)\circ \mathrm{ev}_k\big)$, then we need to refer to $k$, which is not available until we announce the context $\beta\,\lambda k \ldots$. Note that we are not saying that the special variant does not follow from the control algebra equations. It does, but that takes some work. We shall take it for granted for this section.

**Theorem.** *The special contextual abort-homomorphism law holds for any control algebra.*

We continue with the 'control algebras $\to$ monad algebras' argument.

**Lemma 6.** *Suppose a control magma satisfies*

(i) *the contextual call/cc-homomorphism law, and*

(ii) *the special contextual abort-homomorphism law.*

*Then the compose construction produces a magma of the continuation monad that satisfies the multiplication law.*

*Proof.* Let $(X, \beta, \alpha)$ be such a control magma, and $\theta\colon R^{R^X} \to X$ be the function given by the compose construction. For any $\Phi \in R^{R^{R^{R^X}}}$,

$$\begin{aligned}
\theta\,\lambda k^{\in R^X}.\,\Phi(k\circ\theta) &= \beta\,\lambda k^{\in R^X}.\,\alpha\big(\Phi\,\lambda\phi^{\in R^{R^X}}.\,(k\circ\beta)(\alpha\circ\phi)\big) \\
&= \beta\,\lambda k^{\in R^X}.\,\alpha\Big(\Phi\,\lambda\phi^{\in R^{R^X}}.\,(k\circ \mathrm{ev}_{X,k})(\alpha\circ\phi)\Big) && \text{(a)} \\
&= \beta\,\lambda k^{\in R^X}.\,\alpha\Big(\Phi\,\big(k\circ\alpha\circ \mathrm{ev}_k\big)\Big) \\
&= \beta\,\lambda k^{\in R^X}.\,\alpha\big(\Phi\,\mathrm{ev}_k\big) && \text{(b)} \\
&= \theta\,\lambda k^{\in R^X}.\,\Phi\,\mathrm{ev}_k\,.
\end{aligned}$$

Step (a) applies the contextual call/cc-homomorphism law; the following step is trivial calculation of the $\lambda\phi$ expression. Step (b) applies the specialised contextual abort-homomorphism law, taking the variable $D = \alpha\circ\Phi$. The resulting equality is the multiplication law of a continuation algebra. □

Combine the above lemma and the previous argument about the unit law (p77, before introducing the special contextual abort-homomorphism law), then we may restrict the compose construction as needed:

**Theorem 7.** *The compose construction restricts to a functor* CtAlg $\to$ Alg. □

It is more straightforward in the opposite direction; we prove it below. Once that is done, we will have the diagram

$$\begin{tikzcd} \mathrm{CtAlg} \arrow[r, shift left, "\mathrm{comps}"] & \mathrm{Alg}\,. \arrow[l, shift left, "\mathrm{decomp}"] \end{tikzcd}$$

Before looking into the proof, it is useful to be aware of a more convenient form of the multiplication law, of algebras of the continuation monad. We shall call it the **parameterised multiplication law**. The point of this variant is that the variable $\Phi$ may take a parameter $k$ that is bound in the context. See the statement below.

**Lemma 8.** *Let $(X, \theta)$ be an algebra of the continuation monad. For any $\Phi_j \in R^{R^{R^{R^X}}}$, where $j$ is a parameter taken from $R^X$, and under $\theta\,\lambda k^{\in R^X}.\,\Phi_k(-)$,*

$$k \circ \theta = \mathrm{ev}_k.$$

*Proof.* Let $\Phi_j \in R^{R^{R^{R^X}}}$ be an element as stated. We have the following calculation:

$$\begin{aligned}
\theta\,\lambda k^{\in R^X}.\,\Phi_k(k \circ \theta) &= \theta\,\lambda k^{\in R^X}.\,\Phi_{k\circ\theta\circ\mathrm{ev}}(k \circ \theta) && \text{(a)}\\
&= \theta\,\lambda k^{\in R^X}.\,\Phi_{\mathrm{ev}_k\circ\mathrm{ev}}\,\mathrm{ev}_k && \text{(b)}\\
&= \theta\,\lambda k^{\in R^X}.\,\Phi_k\,\mathrm{ev}_k\,.
\end{aligned}$$

Step (a) applies the unit law of the continuation monad algebra. Step (b) applies the multiplication law. □

**Theorem 9.** *The decompose construction restricts to a functor* Alg → CtAlg.

*Proof.* As we have argued earlier, the construction preserves homomorphisms (p76, before the CtMag ⇄ Mag diagram); so we only need to argue that, given an algebra of the continuation monad, the decompose construction produces a control algebra. Let $(X, \theta)$ be an algebra of the continuation monad, and $(X, \beta, \alpha)$ be the control magma given by the decompose construction, i.e.,

$$\beta = \theta \circ \mathrm{drinker}_X \quad \text{and} \quad \alpha = \theta \circ \mathrm{const}$$

(see p75). We need to verify the unit law of a control algebra, the contextual call/cc-homomorphism law and the contextual abort-homomorphism law. For the unit law, we have the following calculation.

$$\begin{aligned}
\beta(\alpha \circ \mathrm{ev}_x) &= (\theta \circ \mathrm{drinker}_X)(\theta \circ \mathrm{const} \circ \mathrm{ev}_x)\\
&= \theta\,\lambda k^{R^X}.\,(k \circ \theta \circ \mathrm{const} \circ \mathrm{ev}_x)k && \text{(a)}\\
&= \theta\,\lambda k^{R^X}.\,(k \circ \theta)\,\mathrm{const}_{kx}\\
&= \theta\,\lambda k^{R^X}.\,\mathrm{ev}_k\,\mathrm{const}_{kx} && \text{(b)}\\
&= \theta\,\mathrm{ev}_x\\
&= x.
\end{aligned}$$

Step (a) is the definition of the drinker function (p75); step (b) is the parameterised multiplication law of the algebra $(X, \theta)$. The last step is the unit law of $(X, \theta)$.

For the contextual call/cc-homomorphism law, we have the following calculation. For any $C \in X^{R^{X^{R^X}}}$,

$$\begin{aligned}
\beta\,\lambda k^{\in R^X}.\,C(k \circ \beta) &= \theta\,\lambda k^{\in R^X}.\,(k \circ C)(k \circ \theta \circ \mathrm{drinker}_X) && \text{(a)}\\
&= \theta\,\lambda k^{\in R^X}.\,(k \circ C)(\mathrm{ev}_k \circ \mathrm{drinker}_X) && \text{(b)}\\
&= \theta\,\lambda k^{\in R^X}.\,(k \circ C)(k \circ \mathrm{ev}_{X,k})
\end{aligned}$$

$$= \beta\,\lambda k^{\in R^X}.\, C(k \circ \mathrm{ev}_{X,k})\,. \tag{c}$$

Steps (a) and (c) apply the definition of $\beta$. Step (b) applies the parameterised multiplication law. The calculation for the contextual abort-homomorphism law is similar. □

With Theorems 7 and 9, we show that the compose and decompose constructions restrict to algebras. Thus we have the following diagram

$$\begin{tikzcd}\mathrm{CtAlg} \arrow[r, shift left, "\text{comps}"] & \mathrm{Alg} \arrow[l, shift left, "\text{decomp}"]\end{tikzcd}\,.$$

To establish the isomorphism theorem, we only need to show that the two round trips of the constructions are both identities. We first look at the control algebra side.

**Theorem 10.** *Given a control algebra, if we carry out the compose construction, then decompose the resulted algebra of the continuation monad, then we get back the same control algebra. In other words, we have equations:*

$$\beta = \theta \circ \mathrm{drinker}_X \tag{3.3}$$

$$\alpha = \theta \circ \mathrm{const} \tag{3.4}$$

*where* $\theta = \beta \circ \alpha^{R^X} = \beta\big(\alpha \circ (-)\big) \colon R^{R^X} \to X$.

*Proof.* We have the following calculation for the first equation (the equation on $\beta$). For any $N \in X^{R^X}$,

$$\begin{aligned}
\beta N &= \beta(\alpha \circ \mathrm{ev}_{\beta N}) && \text{(a)}\\
&= \beta\,\lambda k^{\in R^X}.\,(\alpha \circ k \circ \beta)N\\
&= \beta\,\lambda k^{\in R^X}.\,(\alpha \circ k \circ \mathrm{ev}_{X,k})N && \text{(b)}\\
&= \beta\big(\alpha \circ \lambda k^{\in R^X}.\,(k \circ \mathrm{ev}_{X,k})N\big)\\
&= \beta(\alpha \circ \mathrm{drinker}_X N).
\end{aligned}$$

Step (a) applies the unit law of a control algebra, where the variable $x$ is taken to be $\beta N$. Step (b) applies the contextual call/cc-homomorphism law; the variable $\Phi$ is taken to be $\big(\alpha \circ (-)\big)N$. In the last step, the expression $\alpha \circ \mathrm{drinker}_X N$ should read $\alpha \circ (\mathrm{drinker}_X N)$; by convention, we assume the composition operation $(\circ)$ has low precedence. To verify the equation on $\alpha$, we can adapt the above calculation; we change $\beta N$ to $\alpha r$ (for any $r \in R$), and apply the contextual abort-homomorphism law at step (b). □

The two equations in the above theorem are important in an algebraic understanding of the operations. When we study an algebraic theory, we often consider different combinations of the operations. For example, if we consider symmetries of geometric shapes, we may consider combinations such as 'first a reflection, than a rotation', or the other way around; we are interested in how such combinations relate to each other. We also have combinations here, where the operations are call/cc ($\beta$) and abort ($\alpha$), rather than reflections or rotations. The difference here is that there is more flexibility in the combination of operations. For example, let us consider putting an $\alpha$ before $\beta$. One such combination is $\beta \circ \alpha^{R^X} \colon R^{R^X} \to X$, which is the compose construction. However, there is one more such combination, namely

$$\beta \circ X^{R^\alpha} = \beta\,\lambda k^{\in R^X}.\,(-)(k \circ \alpha) \colon X^{R^R} \to X.$$

The detail of this combination is not important here; our point is that there are two places to choose if we would like to put an $\alpha$ before $\beta$. In other words, if we consider putting operations before $\beta$, then the resulting combinations are not like words (not in a linear shape), but like trees.[21] We can write down an inductive definition of all combinations of $\beta$ and $\alpha$, each of which is a function to $X$ (the detail is regrettably omitted here); among those combinations, we consider $\beta \circ \alpha^{R^X}$ as the *normal form.* This is because, it can be shown that any combination can be written as $\beta \circ \alpha^{R^X} \circ t_X$, where $t$ is a natural transformation, and $t_X$ is its component at the set $X$ (the proof is also omitted here, regrettably). Then, the two equations (3.3), (3.4) in the above theorem are instances of this normalisation. In particular, in (3.3), the natural transformation is $\mathrm{drinker}_X \colon X^{R^X} \to R^{R^X}$, where as in (3.4), that natural transformation is $\mathrm{const} \colon R \to R^{R^X}$. Thus we shall call equations (3.3), (3.4) the *normalisation of call/cc*, and the *normalisation of abort*, respectively.

**The isomorphism theorem.** *We have an isomorphism between the category of control algebras, and the category of algebras of the continuation monad; the isomorphism is given by the compose and decompose constructions.*

*Proof.* After the previous theorem, we only need to show that the other round trip, on algebras of the continuation monad, is also an identity. That is, given an algebra of the continuation monad, if we decompose it into a control algebra, and compose the operations again according to the compose construction, then we get back the same algebra of the continuation monad. To verify that, suppose $(X, \theta)$ is an algebra of the continuation monad, and let $\beta$, $\alpha$ be the operations decomposed from $\theta$ (i.e., $\beta = \theta \circ \mathrm{drinker}_X$ and $\alpha = \theta \circ \mathrm{const}$). Then we have the following calculation: for any $\phi \in R^{R^X}$,

$$\begin{aligned}
\beta(\alpha \circ \phi) &= (\theta \circ \mathrm{drinker}_X)(\theta \circ \mathrm{const} \circ \phi) \\
&= \theta\, \lambda k^{\in R^X}.\, (k \circ \theta)\, \mathrm{const}_{\phi k} && \text{(a)} \\
&= \theta\, \lambda k^{\in R^X}.\, \mathrm{ev}_k\, \mathrm{const}_{\phi k} && \text{(b)} \\
&= \theta\, \phi\,.
\end{aligned}$$

Step (a) applies the definition of $\beta$ and $\alpha$. Step (b) applies the parameterised multiplication law. Thus, the round trip on algebras of the continuation monad is also an identity, and we have the stated isomorphism between the two categories. □

We have now established the isomorphism theorem, assuming the special contextual abort-homomorphism law.

# 5 The diagonalisation principle

We left a gap in the proof of the isomorphism theorem, which is the special contextual abort-homomorphism law. We shall fill the gap here. The stepping stone is a law which we call the

[21] As a personal notation, I write the two combinations as $\beta\frac{1}{\alpha}$ and $\beta\frac{\alpha}{1}$, respectively. The rationale is that the set $R$ is fixed and, since the control algebra is chosen, so is the carrier $X$; thus we don't need to write them explicitly, only to distinguish the two places we would like to put $\alpha$. The idea of my notation is to straighten the exponential notation, i.e., $\overset{\overset{X}{R}}{\alpha}$ in place of $\alpha^{R^X}$; then I borrow the fraction notation to take away $R$ and $X$ (replaced by the division line and 1), writing $\frac{1}{\alpha}$. Finally, I omit the composition symbol $\circ$ between $\beta$ and the fraction, ending up in $\beta\frac{1}{\alpha}$. Admittedly, this is not a perfect notation as there is no division here; nevertheless, the up-side is that the fraction notation allows us to write combinations like $\beta\frac{\beta\frac{1}{\alpha}}{\alpha}$.

diagonalisation principle. We shall motivate the law by considering how the special contextual abort-homomorphism law relates to the original contextual abort-homomorphism law. Impatient readers may skip it and jump to the main development (p84).

We first show that the special contextual abort-homomorphism law implies the original one. Recall the *special contextual abort-homomorphism law* (stated on p77):

> Let $(X, \beta, \alpha)$ be a control magma. We say it satisfies the special contextual abort-homomorphism law, if for any $D \in X^{R^{R^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\, D\big((-) \circ \mathrm{ev}_k\big)$,
>
> $$k \circ \alpha = \mathrm{id}_R\,.$$

(A control magma consists of a carrier and two operations as in control algebras, but without requiring any equations. See p73 in the previous section for a definition.)

**Theorem 11.** *If a control magma satisfies the unit law and the contextual call/cc-homomorphism law, then the special contextual abort-homomorphism law implies the original one.*

*Proof.* Let $(X, \beta, \alpha)$ be a control magma that satisfies the unit law, the contextual call/cc-homomorphism law, and the special contextual abort-homomorphism law. We need to show that the original contextual abort-homomorphism law holds, i.e., for any $\psi \in X^{R^R}$, and under $\beta\,\lambda k^{\in R^X}.\, \psi(-)$,

$$k \circ \alpha = \mathrm{id}_R\,.$$

We see that the equalities within contexts are the same for the two laws (both being $k \circ \alpha = \mathrm{id}_R$); thus for the implication to hold, we only need to consider the contexts. We shall show that for any $\psi \in X^{R^R}$, we can find some $D_\psi \in X^{R^{R^{R^X}}}$, such that the context

$$\beta\,\lambda k^{\in R^X}.\, D_\psi\big((-) \circ \mathrm{ev}_k\big) = \beta\,\lambda k^{\in R^X}.\, \psi(-). \tag{$*$}$$

The right hand side is the context we need in the original contextual abort-homomorphism law. We claim that the $D_\psi$ we need may be defined by

$$D_\psi = \lambda K^{\in R^{R^{R^X}}}.\, \psi(K \circ \mathrm{const}).$$

Here 'const' is the constant function $R \to R^{R^X}$, $\mathrm{const}_r = \lambda k^{\in R^X}.\, r$. We note that for any $k \in R^X$, the expression '$\mathrm{ev}_k \circ \mathrm{const}$' is the identity on $R$. Then, for any $k \in X^R$, and $j \in R^R$,

$$\begin{aligned} D_\psi(j \circ \mathrm{ev}_k) &= \psi\big(j \circ \mathrm{ev}_k \circ \mathrm{const}\big) \\ &= \psi j\,. \end{aligned}$$

We see that it gives us the equality $(*)$. □

Let us consider the converse, i.e., given any control algebra, we consider whether the special contextual abort-homomorphism law holds. In order to show this converse, we need to construct a suitable $\psi_D \in X^{R^R}$ from any $D \in X^{R^{R^{R^X}}}$, such that when we apply the original contextual abort-homomorphism law, the context

$$\beta\,\lambda k^{\in R^X}.\, \psi_D(-) = \beta\,\lambda k^{\in R^X}.\, D\big((-) \circ \mathrm{ev}_k\big)$$

which is the context used in the special contextual abort-homomorphism law. A priori, the variable $\psi_D$ should not depend on $k$, which is bound in the context '$\beta\,\lambda k \ldots$'; an obvious attempt is likely to result in the following law, which is a compromise of what we need. We shall analyse it and see the need for the diagonalisation principle.

**Theorem 12.** *Let $(X, \beta, \alpha)$ be a control algebra. For any $j \in R^X$, $D \in X^{R^{R^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\, D\big((-) \circ \mathrm{ev}_j\big)$,*

$$k \circ \alpha = \mathrm{id}_R\,.$$

*Proof.* We define, for any $j \in R^X$ and $D \in X^{R^{R^{R^X}}}$,

$$\psi_{D,j} = \lambda h^{\in R^R}.\, D(h \circ \mathrm{ev}_j) \in X^{R^R}\,.$$

Then the context in the original contextual abort-homomorphism law is

$$\beta\,\lambda k^{\in R^X}.\, \psi_{D,j}(-) = \beta\,\lambda k^{\in R^X}.\, D\big((-) \circ \mathrm{ev}_j\big)$$

as needed. □

Maybe surprisingly, the special contextual abort-homomorphism law does follow from this compromised version. That is, $j$ can be taken to be $k$, which is bound in the context. We may ask ourselves, can this be a general phenomenon? What is the most general form of the implication we can possibly ask for? If we give it a try, then we may come up with the following statement. Consider any control algebra, with carrier set $X$ and the call/cc operation $\beta$. It would be good if the following holds:

> *Let $\gamma$, $\omega$ be elements of $(X^{R^X})^{R^X}$, i.e., functionals that take two arguments in $R^X$, consecutively. If for all $j \in R^X$, and under $\beta\,\lambda k^{\in R^X}.\,(-)jk$,*
>
> $$\gamma = \omega,$$
>
> *then under $\beta\,\lambda k^{\in R^X}.\,(-)kk$,*
>
> $$\gamma = \omega.$$

We shall call it the **diagonalisation principle**.[22] It should be obvious how we intend to use the law: let

$$\begin{aligned} \gamma jk &= D(k \circ \alpha \circ \mathrm{ev}_j) \\ \omega jk &= D\,\mathrm{ev}_j\,, \end{aligned}$$

then we have the statement that the compromised version of the special contextual abort-homomorphism law implies the uncompromised one.

There is a different way to read the diagonalisation principle. Here is an equivalent formulation of the law:

> *Let $\gamma_j$, $\omega_j$ be elements of $X^{R^X}$, where the parameter $j$ is taken from $R^X$. If for all $j \in R^X$, and under $\beta\,\lambda k^{\in R^X}.\,(-)k$,*
>
> $$\gamma_j = \omega_j\,,$$
>
> *then under the same context,*
>
> $$\gamma_k = \omega_k\,.$$

[22] I choose this name because it is a diagonalisation process on the two arguments $j$, $k$, of the functionals $\gamma$ and $\omega$. If we make a table with values of $\gamma jk$, where the column header enumerates $j$ and the row header enumerates $k$, then $\gamma kk$ are values at the diagonal.

In other words, the law effectively allows variables $\gamma$, $\omega$ to take a parameter $k$ that is bound in the context. Once we have the law, we can also give an alternative proof of the parameterised multiplication law (p79). We will look into that later.

Readers may notice that the abort operation does not appear in the diagonalisation principle. That is a valuable observation, although we shall only give a very brief explanation here. In fact, the diagonalisation principle already holds in a smaller algebraic structure included in a control algebra, namely an *algebra of the selection monad*. The selection monad is a remarkable monad whose functor is $X \mapsto X^{R^X}$ on objects. We refer to Escardó & Oliva [71] for details; but readers may already note the two occurrences of $X$ in the output value of the functor, which is the source of its unusual character. It is explained in Escardó & Oliva that the selection monad is related to the continuation monad by a monad morphism; the monad morphism is the drinker function.[23] An algebra of the selection monad consists of a carrier $X$ together with a single operation $\beta\colon X^{R^X} \to X$, the *call/cc* operation (my terminology). We shall call the unit law of such an algebra the *unit law of call/cc* (we will give the formulation later); the multiplication law has more details than we need here, and we omit its formulation. One can show that for any control algebra $(X, \beta, \alpha)$, the carrier $X$ together with the call/cc operation $\beta$ form an algebra of the selection monad; we can also show that the diagonalisation principle holds for any algebra of the selection monad.[24] Thus the diagonalisation principle may be seen, as a law originated from algebras of the selection monad, and control algebras inherit it. Nevertheless, looking into the selection monad would be a diversion (albeit an impressive one), and we do not chase it further here.

The diagonalisation principle may seem too good to be true. We may consider the following exercise, which may serve as an assurance, and as a break to this long motivation.

**Exercise.** Recall that the intrinsic control algebra in a set $A$ has carrier $R^A$, and the call/cc operation $\beta_A$ is defined as

$$\beta_A N = \lambda a^{\in A}.\, \mathrm{ev}_a(N\,\mathrm{ev}_a) \quad \text{where } N \in (R^A)^{R^{R^A}}.$$

Show that intrinsic control algebras satisfy the diagonalisation principle. That is, let $\gamma$, $\omega$ be functionals in $\big((R^A)^{R^{R^A}}\big)^{R^{R^A}}$; if for all $j \in R^{R^A}$, and under the context

$$\beta_A\, \lambda k^{\in R^{R^A}}.\, (-)jk \;=\; \lambda a^{\in A}.\, \mathrm{ev}_a\big((-)j\,\mathrm{ev}_a\big),$$

we have $\gamma = \omega$, then under the context

$$\beta_A\, \lambda k^{\in R^{R^A}}.\, (-)kk \;=\; \lambda a^{\in A}.\, \mathrm{ev}_a\big((-)\,\mathrm{ev}_a\,\mathrm{ev}_a\big),$$

we also have $\gamma = \omega$.

In the rest we prove the diagonalisation principle, then use it to prove the special contextual abort-homomorphism law (Theorem 18). It takes three steps to establish the diagonalisation principle:

1. Establish the *call/cc unit law* (Theorem 13; we have mentioned it when explaining algebras of the selection monad, before the exercise).

2. Use it to establish the *consecutive call/cc law* (Theorem 15).

[23] Escardó & Oliva works in cartesian-closed categories. We shall stick to the category of sets here.

[24] Use Theorem 16. The consecutive call/cc law we need follows from the multiplication law, of an algebra of the selection monad.

3. Establish the diagonalisation principle as a consequence of the consecutive call/cc law (Theorem 16).

We start from the call/cc unit law. For the statement of this law and some following laws, it is convenient to have a name for the following structure: a set $X$ together with a function $\beta\colon X^{R^X} \to X$, but without any requirement on the equations that $\beta$ shall satisfy. We shall call such a structure a *call/cc magma*.[25] The formulation of the call/cc unit law is as follows.

**The call/cc unit law.** Let $(X, \beta)$ be a call/cc magma. We say the *call/cc unit law* holds for the magma, if for any $x \in X$,

$$x = \beta\,\mathrm{const}_{X,x}\,,$$

where $\mathrm{const}_{X,x}$ is the constant function $R^X \to X$ defined as $\mathrm{const}_{X,x} k = x$.

We say that a control magma $(X, \beta, \alpha)$ satisfies the call/cc unit law if the law holds for $(X, \beta)$. (In the future, if we formulate a law for call/cc magmas, then we say a control magma satisfies the law if the call/cc magma in it does.) To show that control algebras satisfy the call/cc unit law, it is helpful to explicitly formulate naturality of 'ev', the evaluation function. The naturality diagram of the function is

$$\begin{tikzcd}
Y \arrow[r, "\mathrm{ev}"] \arrow[d, "f"'] & R^{R^Y} \arrow[d, "R^{R^f}"] \\
X \arrow[r, "\mathrm{ev}"] & R^{R^X}.
\end{tikzcd}$$

(We have chosen to leave the parameter set $X$ implicit in the notation 'ev', because we would like to reserve the subscript for the base set, i.e., $\mathrm{ev}_X$ is a function $X^A \to X$ for some parameter set $A$.[26]) To say that the diagram commutes is the same as to state the following: for any $y \in Y$,

$$\mathrm{ev}_{fy} = \lambda k^{\in R^X}.\,\mathrm{ev}_y(k \circ f)\,.$$

It only takes a mental calculation to see that the equation does hold.

**Theorem 13.** *If a control magma satisfies the unit law of a control algebra and the contextual call/cc-homomorphism law, then it satisfies the call/cc unit law.*

*Proof.* Let $(X, \beta, \alpha)$ be a control magma as stated, and let $\theta = \beta \circ \alpha^{R^X}$ (i.e., the function given by the compose construction). We have the following calculation.

$$\begin{aligned}
\beta\,\mathrm{const}_{X,x} &= \theta\,\mathrm{ev}_{\beta\,\mathrm{const}_{X,x}} && \text{(a)}\\
&= \theta\,\lambda k^{\in R^X}.\,\mathrm{ev}_{\mathrm{const}_{X,x}}(k \circ \beta) && \text{(b)}\\
&= \theta\,\lambda k^{\in R^X}.\,\mathrm{ev}_{\mathrm{const}_{X,x}}(k \circ \mathrm{ev}_{X,k}) && \text{(c)}\\
&= \theta\,\mathrm{ev}_x \\
&= x\,. && \text{(d)}
\end{aligned}$$

Steps (a), (d) apply the unit law of a control algebra. Step (b) applies naturality of ev. Step (c) applies the contextual call/cc-homomorphism law. □

[25] An alternative name would be a magma of the selection monad. Since we do not study the selection monad here, I have opted for the name above.

[26] We may use superscripts for the parameter set, but we need to make sure that we remember the meanings of superscripts and subscripts, and that we can distinguish the exponential operation from other uses of the superscript. For example, if we use the superscript for the parameter set, then we need a way to determine whether $\mathrm{ev}^X$ means the evaluation function $X \to R^{R^X}$ (where $X$ is the parameter set), or the post-compose function $A^X \to (R^{R^A})^X$, $f \mapsto \mathrm{ev} \circ f$ for some parameter set $A$ (where $X$ is the exponent).

Thus the call/cc unit law holds for any control algebra.

Our next step is the consecutive call/cc law. However, before looking into it, it is useful to be aware of the following convenient form of the contextual call/cc-homomorphism law. It is similar to the parameterised multiplication law (Lemma 8), which is a convenient form of the original multiplication law. Again, the point of the convenient variant is that the variable $C$ may take a parameter $k$ which is bound in the context. See the formulation below.

**The parameterised contextual call/cc-homomorphism law.** Let $(X,\beta)$ be a call/cc magma. We say that $(X,\beta)$ satisfies the *parameterised contextual call/cc-homomorphism law*, if for any $C_j \in X^{R^{X^{R^X}}}$, where $j$ is a parameter taken from $R^X$, and under $\beta\,\lambda k^{\in R^X}.\,C_k(-)$,

$$k \circ \beta = k \circ \mathrm{ev}_{X,k}\,.$$

**Lemma 14.** *If a call/cc magma satisfies the call/cc unit law and the contextual call/cc-homomorphism law, then it satisfies the parameterised contextual call/cc-homomorphism law.*

*Proof.* Let $(X,\beta)$ be a control magma as stated in the assumption. Let $C_j \in X^{R^{X^{R^X}}}$ with parameter $j \in R^X$. We have the following calculation.

$$\begin{aligned}\beta\,\lambda k^{\in R^X}.\,C_k(k\circ\beta) &= \beta\,\lambda k^{\in R^X}.\,C_{k\circ\beta\circ\mathrm{const}}(k\circ\beta) &\text{(a)}\\ &= \beta\,\lambda k^{\in R^X}.\,C_{k\circ\mathrm{ev}_{X,k}\circ\mathrm{const}}(k\circ\mathrm{ev}_{X,k}) &\text{(b)}\\ &= \beta\,\lambda k^{\in R^X}.\,C_k(k\circ\mathrm{ev}_{X,k})\,.\end{aligned}$$

Step (a) applies the call/cc unit law. Step (b) applies the contextual call/cc-homomorphism law. □

From the previous theorem, any control algebra satisfies the call/cc unit law; thus by the lemma, the parameterised contextual call/cc law holds for any control algebra. We shall use this convenient variant when we establish the consecutive call/cc law below.

**The consecutive call/cc law.** Let $(X,\beta)$ be a call/cc magma. We say the *consecutive call/cc law* holds for $(X,\beta)$, if for any $\gamma \in (X^{R^X})^{R^X}$,

$$\beta\,\lambda j^{\in R^X}.\,\beta\,\lambda k^{\in R^X}.\,\gamma j k = \beta\,\lambda j^{\in R^X}.\,\gamma j j\,.$$

As before, we shall say that a control magma satisfies the consecutive call/cc law if the call/cc magma in it does.

**Theorem 15.** *If a control magma satisfies the unit law of a control algebra and the contextual call/cc-homomorphism law, then it satisfies the consecutive call/cc law.*

*Proof.* Let $(X,\beta,\alpha)$ be a control magma as stated in the assumption. We need to show, for any $\gamma \in (X^{R^X})^{R^X}$,

$$\beta\,\lambda j^{\in R^X}.\,\beta\,\lambda k^{\in R^X}.\,\gamma j k = \beta\,\lambda j^{\in R^X}.\,\gamma j j\,.$$

The left hand side may also be written as $\beta(\beta\circ\gamma)$; this alternative form is more convenient for the calculation below. By the previous theorem, we know the control magma satisfies the call/cc unit law; then by the above lemma, the control magma satisfies the parameterised contextual call/cc law. Then we have the following calculation

$$\beta(\beta\circ\gamma) = \beta\,\lambda k^{\in R^X}.\,(\alpha\circ k\circ\beta\circ\gamma)k \qquad \text{(a)}$$

$$
\begin{aligned}
&= \beta\,\lambda k^{\in R^X}.\,(\alpha \circ k \circ \mathrm{ev}_{X,k} \circ \gamma)k && \text{(b)}\\
&= \beta\,\lambda k^{\in R^X}.\,(\alpha \circ k \circ \lambda j^{\in R^X}.\,\gamma j j)k\\
&= \beta\,\lambda j^{\in R^X}.\,\gamma j j\,. && \text{(c)}
\end{aligned}
$$

Steps (a), (c) are normalisations of call/cc; the normalisation law uses both laws mentioned in the assumption. In particular, we use the the following form of the normalisation:

$$\beta N = \beta\,\lambda k.(\alpha \circ k \circ N)k.$$

See the equation (3.3), p80 for the original formulation. Step (b) applies the parameterised contextual call/cc homomorphism law. □

The consecutive call/cc law may be compared to the consecutive lookup law in the state case. Let $S$ be the set of states, and let $(X, \xi, \zeta)$ be a state algebra ($\xi\colon X^S \to X$ is the lookup operation). Then the consecutive lookup law is stated as follows: for any $\gamma \in (X^S)^S$,

$$\xi\,\lambda s^{\in S}.\,\xi\,\lambda t^{\in S}.\,\gamma s t = \xi\,\lambda s^{\in S}.\,\gamma s s.$$

The operational idea of the consecutive lookup law is the following: the state does not change spontaneously; the results of two consecutive lookups must be the same. The operational idea of the consecutive call/cc law is similar: the current continuation may be considered as a state, and this state does not change between consecutive call/cc's.[27]

For interested readers: for an algebra of the selection monad, the consecutive call/cc law follows from the multiplication law of the algebra (with the help of the call/cc unit law, i.e., the unit law of the algebra).

To be precise, let us state the diagonalisation principle formally.

**The diagonalisation principle.** Let $(X, \beta)$ be a call/cc magma. We say it satisfies the *diagonalisation principle* if the following implication holds: For any $\gamma$, $\omega \in (X^{R^X})^{R^X}$, if for all $j \in R^X$, and under $\beta\,\lambda k^{\in R^X}.\,(-)jk$,

$$\gamma = \omega,$$

then under $\beta\,\lambda k^{\in R^X}.\,(-)kk$,

$$\gamma = \omega.$$

**Theorem 16.** *If a call/cc magma satisfies the consecutive call/cc law, then it satisfies the diagonalisation principle.*

*Proof.* Let $(X, \beta)$ be a call/cc magma as stated in the assumption. Suppose that for all $j \in R^X$,

$$\beta\,\lambda k^{\in R^X}.\,\gamma j k = \beta\,\lambda k^{\in R^X}.\,\omega j k\,.$$

This is equivalent to

$$\beta \circ \gamma = \beta \circ \omega\,,$$

and it follows that

$$\beta(\beta \circ \gamma) = \beta(\beta \circ \omega)\,.$$

Apply the consecutive call/cc law on both sides, then we have

$$\beta\,\lambda k^{\in R^X}.\,\gamma k k = \beta\,\lambda k^{\in R^X}.\,\omega k k\,.$$ □

[27] We need to be careful here. The current continuation does change as the program evaluates; nevertheless, it remains the same between two consecutive call/cc's.

By Theorem 15, the consecutive call/cc law holds for any control algebra, then by the previous theorem, it follows that the diagonalisation principle also holds.

**Theorem 17.** *The diagonalisation principle holds for any control algebra.* □

**Corollary 18.** *The special contextual abort-homomorphism law holds for any control algebra.*

*Proof.* We only need to walk through the argument in the motivation again (p83, following Theorem 12). Recall that we have the compromised version of the special contextual abort-homomorphism law: Let $(X, \beta, \alpha)$ be a control algebra; for any $j \in R^X$, $D \in X^{R^{R^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\,D\big((-) \circ \mathrm{ev}_j\big)$,

$$k \circ \alpha = \mathrm{id}_R\,.$$

The equation part is equivalent to, under $\beta\,\lambda k^{\in R^X}.\,(-)$,

$$D(k \circ \alpha \circ \mathrm{ev}_j) = D(\mathrm{id}_R \circ \mathrm{ev}_j)\,.$$

We apply the diagonalisation principle; in particular, we take

$$\begin{aligned} \gamma j k &= D(k \circ \alpha \circ \mathrm{ev}_j) \\ \omega j k &= D\,\mathrm{ev}_j \quad \text{(discards } k\text{)}\,. \end{aligned}$$

Then we obtain the special contextual abort-homomorphism law. □

We have filled the gap in the proof of the isomorphism theorem. Finally, we note that the diagonalisation principle may be used to derive the parameterised laws. We leave it as an exercise.

**Exercise.** Using the diagonalisation principle, show that for any control algebra, the following parameterised laws hold.

(i) The parameterised multiplication law (of the corresponding algebra of the continuation monad);

(ii) The parameterised contextual call/cc-homomorphism law, and

(iii) The parameterised contextual abort-homomorphism law (formulated below).

**The parameterised contextual abort-homomorphism law.** Let $(X, \beta, \alpha)$ be a control magma. We say that it satisfies the *parameterised contextual abort-homomorphism law*, if for any $\psi_j \in X^{R^R}$, where $j$ is a parameter taken from $R^X$, and under $\beta\,\lambda k^{\in R^X}.\,\psi_k(-)$,

$$k \circ \alpha = \mathrm{id}_R\,.$$

# A list of equations

We have seen many control algebra laws in this chapter. We compile them into a list below. Many equations are presented in the colloquial form (i.e., the 'under ... ' convention); see p65, 'the second helper' for details.

**Equations of control algebras** The following equations hold for any control algebra $(X, \beta, \alpha)$. If we encounter a candidate equation not in the list, try intrinsic control algebras (Example, p66).

1. *The unit law of a control algebra* (p65) For any $x \in X$,
$$x = \beta(\alpha \circ \mathrm{ev}_x)\,.$$

2. *The contextual call/cc-homomorphism law* (p65) For any $C \in X^{R^{X^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\,C(-)\,$,
$$k \circ \beta = k \circ \mathrm{ev}_{X,k}\,.$$
We may allow $C$ to take a parameter $k \in R^X$ that is bound in the context (p86, the parameterised contextual call/cc-homomorphism law).

3. *The contextual abort-homomorphism law* (p66) For any $\psi \in X^{R^R}$, and under $\beta\,\lambda k^{\in R^X}.\,\psi(-)\,$,
$$k \circ \alpha = \mathrm{id}_R\,.$$
We may allow $\psi$ to take a parameter $k \in R^X$ that is bound in the context (p88, the parameterised contextual abort-homomorphism law).

4. *The special contextual abort-homomorphism law* (p77, p88) For any $D \in X^{R^{R^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\,D\big((-) \circ \mathrm{ev}_k\big)$,
$$k \circ \alpha = \mathrm{id}_R.$$

5. *Normalisation of call/cc* (p80, Eq. 3.3) For any $N \in X^{R^X}$,
$$\beta N = \beta\,\lambda k^{\,\in R^X}.\alpha\big(k(Nk)\big)\,.$$
(Or equivalently, $\beta = \theta \circ \mathrm{drinker}_X$, where $\theta = \beta \circ \alpha^{R^X}$. See the referenced page for details.)

6. *Normalisation of abort* (p80, Eq. 3.4) For any result $r \in R$,
$$\alpha r = \beta\,\lambda k^{\,\in R^X}.\,\alpha r\,.$$
(Or equivalently, $\beta = \theta \circ \mathrm{const}$, where $\theta = \beta \circ \alpha^{R^X}$. See the referenced page for details.)

7. *The call/cc unit law* (p85) For any $x \in X$,
$$x = \beta\,\mathrm{const}_x\,.$$

8. *The consecutive call/cc law* (p86) For any $\gamma \in (X^{R^X})^{R^X}$,
$$\beta\,\lambda j^{\in R^X}.\,\beta\,\lambda k^{\in R^X}.\,\gamma j k = \beta\,\lambda j^{\,\in R^X}.\,\gamma j j\,.$$

9. *The diagonalisation principle* (p87) For any $\gamma,\ \omega \in (X^{R^X})^{R^X}$, if for all $j \in R^X$, and under $\beta\,\lambda k^{\,\in R^X}.\,(-)jk$,
$$\gamma = \omega,$$
then under $\beta\,\lambda k^{\,\in R^X}.\,(-)kk$,
$$\gamma = \omega.$$

**Equations of algebras of the continuation monad** The following laws hold for any algebra of the continuation monad, $(X, \theta)$.

1. *The unit law* (a defining law) For any $x \in X$,

$$x = \theta\, \mathrm{ev}_x\,.$$

2. *The multiplication law* (a defining law) For any $\Phi \in R^{R^{R^{R^X}}}$, and under $\theta\, \lambda k^{\in R^X}.\, \Phi(-)$,

$$k \circ \theta = \mathrm{ev}_k\,.$$

   We may allow $\Phi$ to take a parameter $k \in R^X$, which is bound in the context (p79, the parameterised multiplication law).

# Chapter 4

# Back to Beck: continuations and determined elements

Beck's monadicity theorem concerns whether a given adjunction is equivalent to the Eilenberg-Moore one. Of course, by the Eilenberg-Moore adjunction, we mean the one that shares the same base category with the given adjunction (i.e., the domain category of the left adjoint), and compose to the same monad as the given adjunction. A typical diagram is as follows, where $F \dashv U$ is the given adjunction:

$$\begin{tikzcd}
 & \mathcal{D} \arrow[dl, "U"'] \arrow[d, "K \vdash Q"] \\
\mathcal{C} \arrow[ur, bend left, "F"] \arrow[ur, phantom, "\perp"] \arrow[dr, bend left, "F^T"] \arrow[dr, phantom, "\perp"] & \\
 & \mathcal{D}^T \arrow[ul, "U^T"] \arrow[u, dashed, bend right]
\end{tikzcd}$$

In the diagram, $T$ is the monad derived from the given adjunction, i.e., $T = UF$. It is a general construction that we always have the comparison functor $K$ from the left category of the given adjunction (the codomain of the left adjoint) to the Eilenberg-Moore category (the category of monad algebras). The given adjunction is said to be *monadic*, or we may also say the right adjoint $U$ is monadic, if the comparison functor $K$ is an equivalence. In Beck's treatment [3], the first step towards monadicity is to construct a left adjoint to $K$ (Theorem 1, op. cit.). We have named it $Q$ (for 'quotient') in the diagram.

The construction of $Q$ is as follows. Let $(X, \theta)$ be an algebra of the derived monad $T$. In the left category, we consider the parallel pair in the left half of the following diagram:

$$\begin{tikzcd}
FUFX \arrow[r, shift left, "F\theta"] \arrow[r, shift right, "\varepsilon_{FX}"'] & FX \arrow[r, dashed, "\pi"] & Q(X, \theta).
\end{tikzcd} \tag{4.1}$$

If for all algebras $(X, \theta)$, we have a chosen coequaliser $\big(Q(X, \theta), \pi\big)$, then those coequalisers give the definition of $Q$, i.e., a left adjoint to $K$. (See Beck's presentation.) The rest of the monadicity theorem then deals with how to make the adjunction $Q \dashv K$ into an adjoint equivalence. In view of the importance of the above pair, we shall name it *Beck's pair*.

Beck's monadicity theorem is well-known since early on, and it is considered as a basic fact among category theorists. However, when the theorem is used to develop general category theory, researchers tend to be satisfied with establishing the equivalence (or sometimes isomorphism), and usually need not consider details of relevant data. For example, it is often fine to leave the coequaliser $Q(X, \theta)$ unspecified, as they may be given by the structure of the left category (say when the left category has all colimits). Eventually that becomes an accustomed way of using the theorem.[1]

As there is close connection between programming language semantics and category theory, monadicity theorems are also well-known in the semantics community. For example, it is well-known that the self-adjoint $R^{(-)}$ and the currying adjunction $S \times (-) \dashv (-)^S$ are monadic under suitable conditions on $R$ and $S$ (e.g., see Hyland et al. [67] for $R^{(-)}$ case and Métayer [61] and Mesablishvili [66] for the state case). However, even though in some of the cases the intended application is clear, the usage of monadicity theorems stays unchanged as in developing general theories; very little attention is given to relevant data. I feel that there is a gap between the way monadicity theorem is used in semantics, and the concrete applications. That leads me to consciously look for potential notions in applications that may serve as data in the monadicity theorem. I share the results below: the set of continuations from an algebra of the continuation monad, and the set of determined elements in an algebra of the state monad; they are coequalisers of Beck's pair in the corresponding setting (Corollary 2, p98; Theorem 5, p102). Between the two, I see the control case by myself, whereas the state case is documented by Métayer (and I have tried to put some effort into the presentation). The viewpoint of copoints and points are my own (around Theorem 3, p99 for the control case, and Theorem 8, p104 for the state case).

# 1 Beck's pair and the instances in control and state

In describing an adjunction, it is convenient to have the term *left category*, which is defined to be the codomain of the left adjoint. That is, for an adjunction

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{D} \arrow[l, "U"] \end{tikzcd}$$

the left category is $\mathcal{D}$. The rationale is that, if we write down the defining isomorphism of the adjunction, namely

$$\mathcal{D}(FX, A) \cong \mathcal{C}(X, UA)$$

then the left category appears on the same side as the left adjoint. We may similarly use 'right category' for the codomain of the right adjoint, but in our setting, we usually just call it the base category.

In the following we will explain Beck's pair, its coequaliser, and instances of them in the cases of control and state. Without further comments, when we explain Beck's pair and its coequaliser, we usually work with a general base category; whereas when we explain the instances, we usually work with the base category Set.

**Definition 1.** Let $F \dashv U$ be an adjunction with counit $\varepsilon$. For any algebra $(X, \theta)$ of the derived monad, *Beck's pair* at the algebra is the following parallel pair in the left category:

$$\begin{tikzcd} FUFX \arrow[r, shift left, "F\theta"] \arrow[r, shift right, "\varepsilon_{FX}"'] & FX. \end{tikzcd}$$

Beck's pair may also be presented in a more symmetric way. In the notation of the definition, let $T$ be the monad derived from the adjunction. Then the Eilenberg-Moore adjunction may be written as $F^T \dashv U^T$, and the counit may be written as $\varepsilon^T$. Here $F^T$ is the free functor and $U^T$

[1] Although I am going to say that knowing the details of relevant data can sometimes be useful, I do understand the reason of the custom. For the development of a general theory, it is often not worth to look into all details and keep track of them. We only bring them up when that is worth the effort.

is the forgetful functor; at any algebra $(X,\theta)$ of the derived monad, $\varepsilon^T_{(X,\theta)}$ is the structure map $\theta$, considered as a homomorphism from the free algebra in $X$ to the algebra $(X,\theta)$. Then, among functors from the Eilenberg-Moore category to the left category, we have $FTU^T$ and $FU^T$, and we have the following parallel pair of natural transformations:

$$\begin{tikzcd} FTU^T \arrow[r, shift left, "FU^T\varepsilon^T"] \arrow[r, shift right, "\varepsilon FU^T"'] & FU^T. \end{tikzcd}$$

Beck's pair at an algebra $(X,\theta)$ is a component of the above pair, namely the component at that algebra.[2]

**Example 1.** Let us consider Beck's pair in the control case. For the self-adjoint $R^{(-)}$, the left category is the opposite category (of Set in our case). Thus we shall consider the opposite Beck's pair in Set. In this case, both $F$ and $U$ are the functor $R^{(-)}$, and the counit $\varepsilon_X$ is the evaluation function $\mathrm{ev}\colon X \to R^{R^X}$ (formulated in Set rather than in the opposite category). Thus the opposite Beck's pair is the parallel pair

$$\begin{tikzcd} R^X \arrow[r, shift left, "R^\theta"] \arrow[r, shift right, "\mathrm{ev}"'] & R^{R^{R^X}}. \end{tikzcd}$$

**Example 2.** We consider Beck's pair in the state case. Let $S$ be a the set of states, i.e., the parameter set in the currying adjunction $S \times (-) \dashv (-)^S$. For the currying adjunction, the left category is the same as the base category Set. The counit $\varepsilon_X$ is the evaluation function $S \times X^S \to X$, which we shall denote by 'e'. (We shall omit the parameter $X$ in the notation 'e'; when the exponent is $S$, we also omit it, otherwise we write the exponent as the subscript, i.e., $\mathrm{e}_A$ is the evaluation function $A \times X^A \to X$ for some $X$. We shall make that $X$ clear in the context.) For an algebra of the state monad, say $(X,\theta)$, Beck's pair is the parallel pair

$$\begin{tikzcd} S \times (S \times X)^S \arrow[r, shift left, "S\times\theta"] \arrow[r, shift right, "\mathrm{e}"'] & S \times X. \end{tikzcd}$$

In both cases, from an operational point of view, the structure map $\theta$ is not the most natural operation in an algebra. For the control case, the operationally more natural operations are call/cc and abort; for the state case, they are lookup and update. We may adjust the formulation of Beck's pair accordingly, using the isomorphism theorem in each case (Theorem, p4 and the isomorphism theorem, p81).

[2] The symmetry may be better seen in a larger diagram

$$\begin{tikzcd}[column sep=large] \cdots & FTTU^T \arrow[r, shift left=8, "FTU^T\varepsilon^T"] \arrow[r, shift right=8, "\varepsilon FTU^T"'] \arrow[r, "F\mu U^T" description] & FTU^T \arrow[l, shift right=4, "FT\eta U^T" description] \arrow[l, shift left=4, "F\eta TU^T" description] \arrow[r, shift left=4, "FU^T\varepsilon^T"] \arrow[r, shift right=4, "\varepsilon FU^T"'] & FU^T \arrow[l, "F\eta U^T" description] \end{tikzcd}$$

where $\eta$, $\mu$ are the unit and multiplication of the derived monad $T$. The diagram may be further simplified and presented as

$$\begin{tikzcd}[column sep=large] \cdots & {[2]} \arrow[r, shift left=8, "d_0"] \arrow[r, shift right=8, "d_2"'] \arrow[r, "d_1" description] & {[1]} \arrow[l, shift right=4, "s_0" description] \arrow[l, shift left=4, "s_1" description] \arrow[r, shift left=4, "d_0"] \arrow[r, shift right=4, "d_1"'] & {[0]}. \arrow[l, "s_0" description] \end{tikzcd}$$

The last diagram is the opposite of the simplicial category $\Delta$. (We are using the $\Delta$ notation as in Mac Lane & Moerdijk [30], which is the opposite of May's [4].) Beck's pair corresponds to the pair $d_0, d_1\colon [0] \rightrightarrows [1]$ in that diagram. We omit the details of the simplicial category here, as well as how exactly the two diagrams correspond.

We get back to the examples below. In the examples, we will consider the following composition of natural transformations

$$\varepsilon F \circ Ff \colon FH \to F.$$

Here $F$ is the left adjoint of a fixed adjunction, where the right adjoint we shall name by $U$, and $\varepsilon$ is the counit; $H$ is an endofunctor on the base category, and $f$ is a natural transformation $H \to UF$. It is a basic property that the composition is the natural adjunct of $f$. The meaning of the last sentence is the following: let $\mathcal{C}$ be the base category, and $\mathcal{D}$ be the left category, then under the natural bijection of the adjunction

$$\mathcal{C}(Y, UFX) \xrightarrow{\cong} \mathcal{D}(FY, FX) \quad \text{natural in } Y,\ X,$$

every component of $f$, say $f_X \colon HX \to UFX$, corresponds to the composition $\varepsilon_{FX} \circ Ff_X$ (we take $Y = HX$ in the bijection); also, naturality is preserved. We shall not explain the details here, as the statements can be easily checked by routine calculations. Readers who find the above description too abstract need not be put off; in each examples we can carry out concrete calculation of $\varepsilon_{FX} \circ Ff_X$, which may be conceptually easier.

We shall now get back to the examples, and consider Beck's pair for the operationally more natural variants of the algebras, i.e., control algebras and state algebras.

**Example 1** (continued)**.** We first consider Beck's pair for control algebras. By the isomorphism theorem, we may consider corresponding algebras of the continuation monad. Let $(X, \theta)$ be an algebra of the continuation monad. Recall that by the decompose construction (p75), the corresponding call/cc and abort operations are given as

$$\beta = \theta \circ \text{drinker}$$
$$\alpha = \theta \circ \text{const}.$$

Then we may adjust the formulation of Beck's pair as follows:

$$\begin{tikzcd}
 & & R^{X^{R^X}} \\
R^X \arrow[r, shift left, "R^\theta"] \arrow[r, shift right, "\text{ev}"'] & R^{R^{R^X}} \arrow[ur, "R^{\text{drinker}}"] \arrow[dr, "R^{\text{const}}"'] & \\
 & & R^R .
\end{tikzcd}$$

That is, we post-compose $R^{\text{drinker}}$ and $R^{\text{const}}$ to the pair, making it a span of parallel pairs. For $R^\theta$, the post-composition results in $R^\beta$, $R^\alpha$. We then consider the post-composition to ev. We recall that 'ev' here is the counit $\varepsilon_{R^X}$ if we work in the opposite category. Also, the natural transformations we would like to post-compose, namely $R^{\text{drinker}}$ and $R^{\text{const}}$, are the results of applying the left adjoint $R^{(-)}$ to the corresponding natural transformations, if we work in the opposite category. Thus we are composing morphisms in the form

$$\varepsilon_{FX} \circ Ff_X \colon FHX \to FUFX,$$

and the basic property mentioned above applies. (The endofunctor $H$ is $X^{R^X}$ in $X$ for 'drinker', and the constant $R$ for const.) The adjunction here is the self-adjoint $R^{(-)}$, or we may call it the swapping adjunction. The natural bijection of the adjunction is the following operation: from a function $f \colon Y \to R^X$ that takes two arguments consecutively, produces an alternative form of the function, $f' \colon X \to R^Y$, where the order of the two arguments are swapped, i.e.,

$$f'yx = fxy.$$

The bijection is the same in both directions. We shall fix the prime sign ($'$) as the notation for this natural bijection.[3] Then, by the basic property, we have

$$R^{\text{drinker}} \circ \text{ev} = \text{drinker}'$$
$$R^{\text{const}} \circ \text{ev} = \text{const}'.$$

Readers who prefer concrete calculations may try the following. Let $A$ be any set, and $f$ be a function $A \to R^{R^X}$, we can show that, for any $k \in R^X$ and $a \in A$,

$$(R^f \circ \text{ev})ka = fak.$$

The equalities about 'drinker', const and ev result in the following diagram, which is Beck's pair for a control algebra $(X, \beta, \alpha)$:

$$\begin{tikzcd}
 & R^{X^{R^X}} \\
R^X \arrow[ur, shift left, "R^\beta"] \arrow[ur, shift right, "\text{drinker}'"'] \arrow[dr, shift left, "R^\alpha"] \arrow[dr, shift right, "\text{const}'"'] & \\
 & R^R .
\end{tikzcd}$$

(We have kept the spanning pairs on the right side of $R^X$, because we would like to reserve the left side for the equaliser. We will describe that later.)

(*End of Example 1*)

**Example 2** (continued)**.** Let us consider Beck's pair for a state algebra $(X, \xi, \zeta)$. Here $\xi$ is the lookup operation, on $X^S$, and $\zeta$ is the update operation, on $S \times X$. We start with an algebra of the state monad, say $(X, \theta)$. There are also compose and decompose construction in the state case, which are similar to the control case. We define a function 'presv' (for 'preserve') as follows:

$$\text{presv}\colon X^S \to (S \times X)^S, \quad N \mapsto \lambda s.\langle s, Ns\rangle.$$

The intended meaning is that the function preserves the input state. We also write 'const' for the constant function $S \times X \to (S \times X)^S$.[4] Then the decompose construction is defined as

$$\xi = \theta \circ \text{presv} \tag{4.2}$$
$$\zeta = \theta \circ \text{const}. \tag{4.3}$$

Then we can obtain Beck's pair for the corresponding state algebra as follows, by pre-composing a cospan of $S \times$ presv, $S \times$ const to the parallel pair:

$$\begin{tikzcd}
S \times X^S \arrow[dr, "S\times\text{presv}"] & & \\
 & S \times (S \times X)^S \arrow[r, shift left, "S\times\theta"] \arrow[r, shift right, "\text{e}"'] & S \times X \\
S \times (S \times X) \arrow[ur, "S\times\text{const}"'] & &
\end{tikzcd}$$

For $S \times \theta$ in the pair, the pre-composition results in $S \times \xi$ and $S \times \zeta$. We then consider the pre-composition to e. It shall be obvious that we are again composing morphisms in the form

$$\varepsilon_{FX} \circ Ff_X \colon FHX \to FUFX,$$

[3] A rather convenient aspect of the notation is that we have $f'' = f$ by definition.

[4] Our convention for omission in the state case has the same rationale as in the control case; but since the state case is about power (i.e., $(-)^S$), whereas the control case is about exponential (i.e., $R^{(-)}$), the actual rules are different. For the constant function, we shall always omit the base set ($S \times X$ in this case); when the exponent is $S$, the set of states, we also omit that, otherwise we write the exponent into the subscript.

and the basic property mentioned above applies. (This time the endofunctor $H$ is $X^S$ for 'presv', and $S \times X$ for const, both in the variable set $X$.) In this case, we are interested in the following natural bijection, as given by the adjunction:

$$\mathrm{Set}\big(A,\, (S \times X)^S\big) \xrightarrow{\cong} \mathrm{Set}(S \times A,\, S \times X)\,.$$

It is the uncurry operation. For any function $f\colon A \to (S \times X)^S$, we shall write $f^-$ for the uncurried form, i.e., for any $s \in S$ and $a \in A$,

$$f^-(s, a) = f a s\,.$$

Using the notation, we may write the pre-composition as

$$\begin{aligned} \mathrm{e} \circ (S \times \mathrm{presv}) &= \mathrm{presv}^- \\ \mathrm{e} \circ (S \times \mathrm{const}) &= \mathrm{const}^-. \end{aligned}$$

Thus we obtain Beck's pair for a state algebra $(X, \xi, \zeta)$ as the following cospan of parallel pairs:

$$\begin{tikzcd} S \times X^S \arrow[dr, shift left, "S\times\xi"] \arrow[dr, shift right, "\mathrm{presv}^-"'] & \\ & S \times X\,. \\ S \times (S \times X) \arrow[ur, shift left, "S\times\zeta"] \arrow[ur, shift right, "\mathrm{const}^-"'] & \end{tikzcd}$$

(*End of Example 2*)

# 2 Continuations, determined elements, and the coequaliser of Beck's pair

Let $f$ be a function $X \to Y$. We have the functional

$$R^{R^f} : R^{R^X} \to R^{R^Y},$$

whose explicit formulation may be calculated as follows. Recall that the functional $R^f\colon R^Y \to R^X$ is the operation 'precompose with $f$' (see Exercise, p5 for some explanation). Then for any $N \in R^{R^X}$, we calculate that

$$\begin{aligned} R^{R^f} N &= N \circ R^f \\ &= \lambda j^{\in R^Y}.\,(N \circ R^f) j \\ &= \lambda k^{\in R^Y}.\,N(j \circ f). \end{aligned}$$

We may again consider $R^{R^{(-)}}$ as a functional. We define

$$\mathbf{t}_Y = R^{R^{(-)}} : Y^X \to (R^{R^Y})^{R^{R^X}}\,;$$

that is, for any $f \in Y^X$,

$$\mathbf{t}_Y f = R^{R^f} = \lambda N^{\in R^{R^X}}.\lambda j^{\in R^Y}.\,N(j \circ f). \tag{4.4}$$

We shall call this functional the *internal functor* (of the continuation monad) in $Y$. [5] As a convention, when the base set $Y$ is the result set $R$, we may leave it out in the notation, i.e., $\mathbf{t}$ is a functional $R^{R^X} \to R^{R^R}$ for some parameter set $X$. This is consistent with the convention we have been using, say, in the notation 'ev'.

**Theorem 1.** *Let $(X, \theta)$ be an algebra of the continuation monad. Continuations from the algebra form the equaliser of the parallel pair*

$$R^{\theta},\, ({\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}} \circ \mathbf{t})\colon R^X \rightrightarrows R^{R^{R^X}}.$$

*Proof.* By definition, continuations from $(X, \theta)$ are homomorphisms $(X, \theta)$ to the result set algebra $R$. That is, they are function $k\colon X \to R$ such that the following diagram commutes:

$$\begin{tikzcd}
R^X \arrow[r, "R^{R^k}"] \arrow[d, "\theta"'] & R^{R^R} \arrow[d, "\mathrm{ev}_{\mathrm{id}_R}"] \\
X \arrow[r, "k"] & R.
\end{tikzcd}$$

The diagram may be written as an equation

$$k \circ \theta = \mathrm{ev}_{\mathrm{id}_R} \circ R^{R^k}.$$

We may rewrite the equation as follows. The left hand side may be reformulated as $R^{\theta}k$, i.e., precomposing $k$ with $\theta$. The right hand side may be reformulated as 'post-composing $R^{R^k}$ with $\mathrm{ev}_{\mathrm{id}_R}$', i.e.,

$${\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}}(R^{R^k}) = {\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}}(\mathbf{t}k) = ({\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}} \circ \mathbf{t})k.$$

Therefore the set of continuations may be expressed as

$$\{k \in R^X \mid R^{\theta}k = ({\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}} \circ \mathbf{t})k\}.$$

That is exactly the equaliser of the parallel pair

$$R^{\theta},\, ({\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}} \circ \mathbf{t})\colon R^X \rightrightarrows R^{R^{R^X}}. \qquad \square$$

For any algebra of the continuation monad, say $(X, \theta)$, let us write $\mathrm{ctn}(X, \theta)$ for the set of continuations from the algebra. Then the above theorem states that the diagram

$$\begin{tikzcd}
\mathrm{ctn}(X,\theta) \arrow[r] & R^X \arrow[rr, "R^{\theta}"] \arrow[dr, "\mathbf{t}"'] & & R^{R^{R^X}} \\
& & (R^{R^R})^{R^{R^X}} \arrow[ur, "{\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}}"'] &
\end{tikzcd}$$

[5] I am thinking in enriched category theory here for the terminology. By an internal functor, I mean the natural transformation that comes with an enriched functor. Here Set is self-enriched, and the exponentials are internal homs. This notion of internal functors does not seem to receive a name, although it certainly deserves one; for example, the notion is unnamed in the classical text by Kelly [63]. Given that we have three essentially equivalent ways to formulate enrichment, i.e., the enrichment itself, by tensors, and by cotensors, I feel that it would be good that the equivalents of the internal functors all bear the same base name, and distinguished by some modifiers. For example, I am tempted to call the notion of internal functors in the three settings by *functorial strengths*, *tensorial strengths* (as conventional), and *cotensorial strengths*, respectively. Also, as before, we leave the parameter set $X$ implicit in the notation, and reserve the subscript for the base set ($Y$ in the case of the definition). See footnote 26, p85 for the rationale.

is a coequaliser diagram. However, a more transparent way to read the theorem, is that the coequaliser diagram is the internal description of continuations from the algebra $(X,\theta)$, in the sense that such a continuation is a homomorphism to the result set algebra. The term ‘internal’ is taken from enriched category theory. Here we consider Set as a self-enriched category, and consider sets

$$R^X, \quad (R^{R^R})^{R^{R^X}} \quad \text{and} \quad R^{R^{R^X}}$$

as the internal presentations of the homsets

$$\mathrm{Set}(X,R), \quad \mathrm{Set}(R^{R^X}, R^{R^R}) \quad \text{and} \quad \mathrm{Set}(R^{R^X}, R)$$

respectively. Thus the morphism $\mathrm{ctn}(X,\theta) \to R^X$ in the equaliser shall be understood as a forgetful map, which forgets that the continuations are homomorphic. We shall show that the above coequaliser diagram is exactly the coequaliser diagram of Beck’s pair at the algebra $(X,\theta)$. The formulation of Beck’s pair only refers to data related to the adjunction and an algebra; however, in this case, the coequaliser diagram suggests we work in the mindset of enriched category theory.[6]

From here it only takes a small step to arrive at Beck’s pair. We note that the composition ${\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}} \circ \mathbf{t}$ in the equaliser diagram may be reduced to the evaluation function $\mathrm{ev}\colon R^X \to R^{R^{R^X}}$. We may see that by the following calculation. For any $k \in R^X$,

$$\begin{aligned}
({\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}} \circ \mathbf{t})\,k &= {\mathrm{ev}_{\mathrm{id}_R}}^{R^{R^X}}(\mathbf{t}k) \\
&= \mathrm{ev}_{\mathrm{id}_R} \circ \mathbf{t}k \\
&= \lambda N^{\in R^{R^X}}.\,(\mathrm{ev}_{\mathrm{id}_R} \circ \mathbf{t}k)N \\
&= \lambda N.\,\mathbf{t}kN\,\mathrm{id}_R & (*) \\
&= \lambda N. Nk \\
&= \mathrm{ev}_k\,.
\end{aligned}$$

Step $(*)$ applies the definition of $\mathbf{t}$, eq. (4.4). With the equality, we may redraw the equaliser diagram as

$$\mathrm{ctn}(X,\theta) \longrightarrow R^X \underset{\mathrm{ev}}{\overset{R^\theta}{\rightrightarrows}} R^{R^{R^X}}\,. \tag{4.5}$$

That is exactly the coequaliser diagram of the corresponding Beck’s pair, in the opposite category. Let us phrase it as a corollary.

**Corollary 2.** *For any algebra of the continuation monad, continuations from that algebra form the coequaliser of the corresponding Beck’s pair.* □

Let us wind back to the homomorphism diagram

$$\begin{tikzcd}
R^X \arrow[r, "R^{R^k}"] \arrow[d, "\theta"'] & R^{R^R} \arrow[d, "\mathrm{ev}_{\mathrm{id}_R}"] \\
X \arrow[r, "k"] & R\,.
\end{tikzcd}$$

There is a different reading of the diagram. We can show that, in the category of algebras of the continuation monad, the result set algebra is initial. Thus the diagram describes a global copoint: the continuation $k$ picks a global copoint from the algebra $(X,\theta)$, in the category of algebras.

[6] Obviously this is due to the adjoint $R^{(-)}$, which is about exponentials. Nevertheless, it still caught me with surprise when I first saw it.

**Theorem 3.** *The result set algebra is initial in the Eilenberg-Moore category of the continuation monad. In particular, for an algebra $(X, \theta)$ of the continuation monad, the unique homomorphism from the result set algebra is the abort operation in the corresponding control algebra, i.e.,*

$$\alpha = \theta \circ \mathrm{const},$$

*as given by the decompose construction* (p75).

*Proof.* We first show the necessity, i.e., if there is ever a homomorphism from the result set algebra to an algebra of the continuation monad, then it can only be the abort operation of the corresponding control algebra. By the isomorphism theorem (p81), a function is homomorphic between algebras of the continuation monad if and only if it is homomorphic between the corresponding control algebras. In the case here, if a function $f\colon R \to X$ is homomorphic between the monad algebras, then it is necessary that it is abort-homomorphic between the corresponding control algebras. By definition, the latter means the diagram

$$\begin{tikzcd}
& R \arrow[dl, "\mathrm{id}"'] \arrow[dr, "\alpha"] & \\
R \arrow[rr, "f"] & & X
\end{tikzcd}$$

has to commute. It follows that $f = \alpha$. Thus if there is ever a homomorphism from the result set to the algebra $(X, \theta)$, then it has to be the abort operation $\alpha$. We now show that $\alpha$ is homomorphic. This homomorphism requirement can be given as the diagram

$$\begin{tikzcd}
R^{R^R} \arrow[r, "R^{R^\alpha}"] \arrow[d, "\mathrm{ev}_{\mathrm{id}_R}"'] & R^{R^X} \arrow[d, "\theta"] \\
R \arrow[r, "\alpha"] & X\,.
\end{tikzcd}$$

By normalisation of abort (p80), we may write the lower-left leg composition as

$$\theta \circ \mathrm{const} \circ \mathrm{ev}_{\mathrm{id}_R}.$$

By normalisation of abort and the multiplication law, we may right the upper-right leg composition as

$$\theta \circ R^{\mathrm{const}'}.$$

We only need to show that the factors before $\theta$, namely $\mathrm{const} \circ \mathrm{ev}_{\mathrm{id}_R}$ and $R^{\mathrm{const}'}$, are equal. We calculate that, for any $N \in R^{R^R}$ and $k \in R^X$,

$$\begin{aligned}
(\mathrm{const} \circ \mathrm{ev}_{\mathrm{id}_R})Nk &= \mathrm{const}(\mathrm{ev}_{\mathrm{id}_R} N)k \\
&= \mathrm{const}_{N\mathrm{id}_R} k \\
&= N\mathrm{id}_R
\end{aligned}$$

and

$$\begin{aligned}
R^{\mathrm{const}'}Nk &= (N \circ \mathrm{const}')k \\
&= N(\mathrm{const}'k) \\
&= N\mathrm{id}_R.
\end{aligned}$$

Thus $\mathrm{const} \circ \mathrm{ev}_{\mathrm{id}_R}$ and $R^{\mathrm{const}'}$ are equal, so are their pre-compositions to $\theta$, and the homomorphism diagram commutes. Therefore the abort operation $\alpha$ is a homomorphism from the result set to the algebra $(X, \theta)$. □

How about the state case? In the state case, the notion that plays the same role as continuations are determined elements. We shall assume the set of states, $S$, has at least one element. However, we won't use that assumption until we prove that the set of determined elements is the coequaliser of Beck's pair (Theorem 5, p102).

**Definition 2.** Let $(X, \theta)$ be an algebra of the state monad. A *determined element* of the algebra is an element $x \in X$ such that for any state $s \in S$,

$$\zeta(s, x) = x.$$

Here $\zeta$ is the update operation of the corresponding state algebra, defined as $\zeta = \theta \circ \mathrm{const}\colon S \times X \to X$, as given by the decompose construction (Eq. (4.3), p95). We shall write $\mathrm{de}(X, \theta)$ for the set of determined elements of that algebra.

Operationally, a determined element is an element that won't be changed when preceding with updates. By the *consecutive update* law

$$\zeta\big(s, \zeta(t, x)\big) = \zeta(t, x) \quad \text{for any } s,\, t \in S \text{ and } x \in X,$$

any updating element $\zeta(t, x)$ is determined. In fact, they are all determined elements, as we shall briefly describe below.[7]

We first describe the set of determined elements by a universal property. In order to do that, it is convenient to have the following variant of an equaliser.

**Definition 3.** Let $F$ be a functor. We shall call its domain the base category, and its codomain the target category. Let $X$ be an object in the base category, and, in the target category, let $A$ be an object, and $a, b\colon FX \rightrightarrows A$ be a parallel pair of morphisms.

(i) In the base category, let $Y$ be an object and $f\colon Y \to X$ be a morphism. We say that $(Y, f)$ *equalises* the pair $a$, $b$ under $F$ (or $F$-*equalises* the pair), if $Ff$ equalises them in the target category (see the diagram below);

(ii) In the base category, let $E$ be an object and $e\colon E \to X$ be a morphism, we say $(E, e)$ is an *equaliser* of the pair $a$, $b$ under $F$ (or an $F$-*equaliser* of the pair), if it is terminal among $(Y, f)$ that equalises the pair.

A diagram that depicts the definition is as follows, in the target category:

$$\begin{tikzcd}
FY \arrow[rr, "Ff"] \arrow[dr, dashed, "Fu"'] & & FX \arrow[r, shift left, "a"] \arrow[r, shift right, "b"'] & A \\
& FE \arrow[ur, "Fe"'] & &
\end{tikzcd}$$

(The unusual positioning of the equaliser and the equalising morphism is on purpose. That arrangement fits better with our later diagram in the state case.)

We will look into the set of determined elements as an example; but before that, it is useful to note the following equivalent description, when $F$ is a left adjoint. In that case, the definition can be phrased succinctly by equalisers in the usual sense. Suppose $F$ has a right adjoint, say $U$, and let $\varphi$ be its defining natural bijection, from the left category to the base category to be specific. Then, using the same notation as in the definition,

[7] The consecutive update law can be easily proved, using normalisation of update, and the multiplication law of an algebra of the state monad. Obviously this is aided by the isomorphism theorem in the state case.

(i) $(Y, f)$ equalises a parallel pair $a$, $b$ under $F$ if and only if it equalises the parallel pair under the natural bijection, i.e., $\varphi a, \varphi b \colon X \rightrightarrows UA$, in the usual sense (see the diagram below).

(ii) $(E, e)$ is an equaliser of the pair under $F$ if and only if it is an equaliser of the parallel pair under the natural bijection, in the usual sense.

A diagram that depicts the equivalent formulation is as follows, in the base category:

$$\begin{tikzcd}
Y \arrow[rr, "f"] \arrow[dr, dashed, "u"'] & & X \arrow[r, shift left, "\varphi a"] \arrow[r, shift right, "\varphi b"'] & UA \\
& E \arrow[ur, "e"'] & &
\end{tikzcd}$$

We now describe the set of determined elements as a universal object. Let us write ‘pr’ for the projection $S \times X \to X$ (as usual, we omit the parameter set $X$ in the notation). Let $(X, \theta)$ be an algebra of the state monad. We have in mind the following guiding diagram, which will be justified below:

$$\begin{tikzcd}
S \times (S \times X) \arrow[rr, "S \times \zeta"] \arrow[dr, dashed, "S \times \pi"'] & & S \times X \arrow[r, shift left, "\zeta"] \arrow[r, shift right, "\mathrm{pr}"'] & X. \\
& S \times \mathrm{de}(X, \theta) \arrow[ur, "S \times i"'] & &
\end{tikzcd} \tag{4.6}$$

Here $\zeta$ is the update operation of the corresponding state algebra, and $i$ is the inclusion $\mathrm{de}(X, \theta) \subseteq X$.

**Theorem 4.** *Let $(X, \theta)$ be an algebra of the state monad. Then $\mathrm{de}(X, \theta)$, the set of the determined elements,* (*together with the inclusion to $X$,*) *is the equaliser of the pair*

$$S \times X \overset{\zeta}{\underset{\mathrm{pr}}{\rightrightarrows}} X$$

*under the left adjoint $S \times (-)$. As usual, $\zeta$ is the update operation of the corresponding state algebra.*

*Proof.* By definition, the set $\mathrm{de}(X, \theta)$ is

$$\begin{aligned} \{x \in X \mid \zeta(s, x) = \mathrm{pr}(s, x) \\ \text{for any } s \in S\}. \end{aligned}$$

Then the universal property is obvious. □

**Theorem.** *The update operation $\zeta$ equalises the parallel pair $\zeta, \mathrm{pr} \colon S \times X \rightrightarrows X$ under the left adjoint $S \times (-)$.*

(Confer the guiding diagram before the previous theorem.)

*Proof.* This is exactly the consecutive update law: for any $s$, $t \in S$ and $x \in X$,

$$\zeta\bigl(s, \zeta(t, x)\bigr) = \zeta(t, x). \qquad \square$$

By the previous theorems, we have obtained the guiding diagram (4.6); given that the set of determined elements is the equaliser under $S \times (-)$, there is a unique function from $S \times X$ to that set, and by $\pi$ we refer to that mediating function (following Beck in his monadicity theorem, which is for adjunctions in general). The explicit description of $\pi$ is simple: it is the same function as $\zeta$, except for a different codomain (or we may say $\pi$ has the same graph as $\zeta$).

Now let us curry the guiding diagram, to obtain

$$\begin{tikzcd}
S \times X \arrow[rr, "\zeta"] \arrow[dr, dashed, "\pi"'] & & X \arrow[r, shift left, "\zeta^+"] \arrow[r, shift right, "\mathrm{pr}^+"'] & X^S. \\
& \mathrm{de}(X, \theta) \arrow[ur, "i"'] & &
\end{tikzcd}$$

Here $\zeta^+$ and $\mathrm{pr}^+$ are the curried form of the corresponding functions. By our general comments after the definition of equaliser under a functor, we know that this is an equaliser diagram of $\mathrm{de}(X, \theta)$, together with its universal property applied to the update operation $\zeta$. At this point we can finally exhibit the full picture. It is a diagram in Set (both the base category and the left category), obtained by pre-composing with the corresponding Beck's pair (Example 2, p93):

$$\begin{tikzcd}
S \times (S \times X)^S \arrow[r, shift left, "S\times\theta"] \arrow[r, shift right, "\mathrm{e}"'] & S \times X \arrow[rr, "\zeta"] \arrow[dr, "\pi"'] & & X \arrow[r, shift left, "\zeta^+"] \arrow[r, shift right, "\mathrm{pr}^+"'] & X^S. \\
& & \mathrm{de}(X, \theta) \arrow[ur, "i"'] & &
\end{tikzcd} \tag{4.7}$$

As readers may expect, the only missing fact from the diagram is the coequaliser of Beck's pair, as stated in the following theorem. (This is documented in Métayar [61].)

**Theorem 5.** *Let $(X, \theta)$ be an algebra of the state monad, and $\pi\colon S \times X \to \mathrm{de}(X, \theta)$ as the function defined as above. Then $\mathrm{de}(X, \theta)$, the set of determined elements, together with $\pi$, form a coequaliser of the corresponding Beck's pair.*

Before proving the theorem, we need two lemmas.

**Lemma 6.** *The update operation $\zeta$ of the state algebra coequalises the corresponding Beck's pair.*

(See the above full picture for a diagram.)

*Proof.* Normalise the update operation, then apply the multiplication law. (Compare this with the proof that the abort operation of a control algebra is a homomorphism from the result set algebra to that control algebra. It is the latter part of the proof to Theorem 3, 'the result set algebra is initial', p99.) □

For any $s_0 \in S$, we denote by '$\mathrm{tag}\, s_0$' the tagging function

$$X \to S \times X, \quad x \mapsto (s_0, x).$$

As a benign abuse of notation, we shall also denote by '$\mathrm{tag}\, s_0$' the composition

$$\mathrm{de}(X, \theta) \xrightarrow[\text{(inclusion)}]{i} X \xrightarrow{\mathrm{tag}\, s_0} S \times X.$$

**Lemma 7.** *Let $s_0 \in S$. We have the following split coequaliser diagram:*

$$\begin{tikzcd}
S \times (S \times X)^S \arrow[r, shift left, "S\times\theta"] \arrow[r, shift right, "\mathrm{e}"'] & S \times X \arrow[l, bend left=40, "\mathrm{tag}\, s_0 \circ \mathrm{const}"] \arrow[dr, "\pi"] & \\
& & \mathrm{de}(X, \theta). \arrow[ul, bend left=40, "\mathrm{tag}\, s_0"]
\end{tikzcd}$$

*In the diagram, we have placed sections (in curved arrows) next to their corresponding retracts. The set that is not presented in the composition* $\mathrm{tag}\, s_0 \circ \mathrm{const}$ *is* $(S \times X)^S$.

*Proof.* Since $\pi$ and $\zeta$ are essentially the same function, by the previous lemma, we know that $\pi$ also coequalises Beck's pair. Then it remains to show that

(i) $\operatorname{tag} s_0$ is a section to $\pi$;

(ii) $\operatorname{tag} s_0 \circ \mathrm{const}$ is a section to e, and

(iii) The following diagram commutes:

$$\begin{tikzcd}
S \times (S \times X)^S \arrow[r, "S\times\theta"] & S \times X \\
S \times X \arrow[u, "\operatorname{tag} s_0 \circ \mathrm{const}"] \arrow[r, "\pi"] & \mathrm{de}(X, \theta). \arrow[u, "\operatorname{tag} s_0"']
\end{tikzcd}$$

These are all routine calculation. For the commuting square in (iii), we need to note the decomposing construction (Eq. (4.3), p95). □

We forget about the splitting structure, then obtain Theorem 5, i.e., for an algebra of the state monad, the set of determined elements is the coequaliser of Beck's pair. Early in the explanation, we mention that every updating element $\zeta(s, x)$ is a determined element (p100, 'the consecutive update law'). We can now complete the other half of the statement. By being the morphism in the coequaliser, $\pi\colon S \times X \to \mathrm{de}(X, \theta)$ is surjective, thus updating elements are all determined elements. (This is also in Métayer's note.)

Finally, we explain the viewpoint that determined elements are global points of the algebra. We note that the parallel pair

$$\begin{tikzcd}
X \arrow[r, shift left, "\zeta^{+}"] \arrow[r, shift right, "\mathrm{pr}^{+}"'] & X^S.
\end{tikzcd}$$

is isomorphic to the parallel pair

$$\begin{tikzcd}
& (S \times X)^{S\times 1} \arrow[dr, "\zeta^{S\times 1}"] & \\
X^1 \arrow[ur, "\mathbf{t}"] \arrow[rr, "X^{\langle\rangle}"] & & X^{S\times 1}.
\end{tikzcd} \tag{4.8}$$

Here 1 is the terminal object, i.e., the singleton set; $\langle\rangle$ is the unique morphism to 1, i.e., the empty tuple; and $\mathbf{t}$ is the internal functor of $S \times (-)$, defined as

$$\lambda u \,.\, x \mapsto \lambda\langle s, u\rangle . \langle s, x\rangle\,.$$

The equaliser diagram of the latter parallel pair is the internal formulation of the set, consisting of all global elements $x\colon 1 \to X$ such that the following diagram commutes:

$$\begin{tikzcd}
S \times 1 \arrow[r, "S\times x"] \arrow[d, "\langle\rangle"'] & S \times X \arrow[d, "\zeta"] \\
1 \arrow[r, "x"] & X.
\end{tikzcd}$$

We note that 1 is the terminal algebra with the evident operations (all being the unique morphism $\langle\rangle$). Thus the commuting of the above diagram means $x$ is an update homomorphism from the terminal algebra to the algebra $(X, \theta)$. On the other hand, the corresponding lookup homomorphism diagram is

$$\begin{tikzcd}
1^S \arrow[r, "x^S"] \arrow[d, "\langle\rangle"'] & X^S \arrow[d, "\xi"] \\
1 \arrow[r, "x"] & X.
\end{tikzcd}$$

The diagram commutes for any global point $x$. Thus a coequaliser of the parallel pair 4.8 is the set that consists of all global points $x$ in Set, which is a homomorphism from the terminal algebra to the algebra $(X, \theta)$. In other words, such a set is the set of global points of the algebra $(X, \theta)$ in the category of algebras. We have proved the following theorem:

**Theorem 8.** *For any algebra of the state monad, determined elements are in bijection with global points of the algebra in the category of algebras.* □

# Chapter 5

# Conclusion

## 1 A summary of the thesis

We shall first review the main results of the thesis. We have begun the work by raising the following research problems:

**Research Problem 1.** Find suitable equations for control operators *call/cc* and *abort*. Such equations should characterise the control effect presented as the continuation monad, much like how equations of state algebras characterise the state effect.

**Research Problem 2.** Study monadicity in the cases of state and control. We shall

(i) Identify common structure of the monadicity arguments.

(ii) Pay attention to data. More specifically, we consider Beck's pair (to be explained below), and possible operational interpretation of it.

We have solved research problem 1, and made some progress on research problem 2. For research problem 1, our answer is the definition of a control algebra (Definition 1, p63; also see p65). The definition is justified by the isomorphism theorem (p81). The significance of the definition is that, it is formulated using natural control operators, namely call/cc and abort, while it is equivalent to algebras of the continuation monad. Control algebras play the same role for the continuation monad as state algebras do for the state monad. Historically, Plotkin & Power discover the first formulation of state algebra due to their emphasis on operationally natural operations [53]. Here we do the same for the continuation monad, although less emphasis is put on Lawvere theories compared to their approach. (This is partly due to technical reasons; see the Lawvere theory section (§3) in the introduction chapter for more details.)

Apart from the definition of control algebras and the isomorphism, I would like to mention the homomorphism viewpoint of continuations as a piece of contribution. This viewpoint is suggested in the two contextual homomorphism laws in the definition of a control algebra. Using it to interpret programming language structures works well. (We have not shown such examples in this work, but readers may try to interpret Levy's call-by-push-value with letstk and changestk; in particular, the type of a control identifier may be interpreted as the set of continuations from the corresponding algebra.) As readers can easily see, I consider this viewpoint important, thus phrasing it as a principle, and eventually define continuations this way (Definition 3, p72).

The second research problem is more open in nature. What we have done here is to identify the set of continuations as a signature object in terms of monadicity. By that we mean, for an algebra of the continuation monad, the coequaliser of Beck's pair is exactly the set of continuation from that algebra. Similarly, in the state case, the signature object in terms of monadicity is the set of determined elements. This is already noticed by other researchers (Métayer [61] for example, although not giving such elements a name). These signature objects encourage us to pay attention to data involved in categorical statement, especially in application. Besides, we also notice the global copoint description of continuations and the global point description of determined elements. The similarity between the two descriptions is evident. It may be seen as an instance of commonality of state and control, from the monadicity perspective. This is one attempt to look for commonality between the two effects; one eventual goal of the effort is to gather enough ingredients to understand algebras defined by control operators letstk and changestk (or similarly, catch and throw).

In the global copoint and global point description, relevant diagrams are read as internal descriptions of some homomorphism condition. I think this is a notable fact. It suggests, generalisation of the description may need to use enrichment.

Let us collect the important results below. At this point, we may have more freedom in the formulation and explanation, which may not be available earlier.

**Definition.** A *control algebra* is a set $X$ together with functions

$$\begin{aligned} \beta\colon X^{R^X} &\to X \\ \alpha\colon R &\to X \end{aligned}$$

such that three equations, namely the unit law, the contextual call/cc-homomorphism law, and the contextual abort-homomorphism law hold. The precise formulation of the laws are as follows:

(i) For any $x \in X$,

$$x = \beta(\alpha \circ \mathrm{ev}_x). \qquad \text{(the unit law)}$$

(ii) For any $C \in X^{R^{X^{R^X}}}$, and under $\beta\,\lambda k^{\in R^X}.\,C(-)$,

$$k \circ \beta = k \circ \mathrm{ev}_{X,k}\,. \qquad \text{(the contextual call/cc-homomorphism law)}$$

(iii) For any $\psi \in X^{R^R}$, and under $\beta\,\lambda k^{\in R^X}.\,\psi(-)$,

$$k \circ \alpha = \mathrm{id}_R\,. \qquad \text{(the contextual abort-homomorphism law)}$$

In the definition, the operation $\beta$ corresponds to the call/cc operator, whereas the operation $\alpha$ corresponds to the abort operator. The unit law here is exactly the unit law of an algebra of the continuation monad; this can be readily seen from the compose construction, which gives the structure map $\theta = \beta\big(\alpha \circ (-)\big)$ from the two operations. In the other two laws, $\beta\,\lambda k.\,C(-)$ and $\beta\,\lambda k.\,\psi(-)$ are contexts of the equations; it means when interpreting the equality in a law, we shall first wrap both sides of the equation with the corresponding context. (See Definition 1, p63 for the full forms.) The key parts in those two laws are about the homomorphism condition. In particular, if the contexts were removed, then the equalities state the following homomorphism diagrams:

$$\begin{tikzcd}
X^{R^X} \arrow[r, "k^{R^k}"] \arrow[d, "\beta"'] & R^{R^R} \arrow[d, "\mathrm{ev}_{\mathrm{id}_R}"] \\
X \arrow[r, "k"] & R
\end{tikzcd}
\qquad
\begin{tikzcd}
& R \arrow[dl, "\alpha"'] \arrow[dr, "\mathrm{id}"] & \\
X \arrow[rr, "k"] & & R
\end{tikzcd}$$

The diagrams require that $k$ be both call/cc-homomorphic and abort-homomorphic. However, we do have contexts in the laws, thus the way to read them is that under those contexts (which declare $k$), $k$ may be seen as call/cc- and abort-homomorphic. That is how the names of the laws come to be.

We note that there are following similarities between equations of a control algebra and of a state algebra: they each have a unit law that is the same as that of the corresponding monad algebra; in the other two laws, one is about the copying operation, i.e., call/cc and lookup, and the other is about the discarding operation, i.e., abort and update. However, what is interesing is the difference: for a control algebra, it is the call/cc operation that appears in both non-unit laws, whereas for a state algebra, the common operation is update. Now with the experience in contexts, we may look at the defining equations of a state algebra in a different way. Recall that the update-lookup and the consecutive update laws are as follows:

$$\zeta(s, \xi f) = \zeta(s, fs)$$
$$\zeta\big(s, \zeta(t, x)\big) = \zeta(t, x).$$

We may rephrase them in the following way:

(i) For any $s \in S$, and under $\zeta(s, -)$,
$$\xi = \mathrm{ev}_s.$$

(ii) When restricted to elements $\zeta(t, x)$ (i.e., the updating elements),
$$\zeta = \mathrm{pr}.$$

Isn't that close to how we would explain to a colleague, say in a short walk? In words, (i) expresses that after an update, a lookup feeds the new state to its subterm; (ii) expresses that, prior to an update, any update works as if it discards the accompanying state (i.e., the update becomes useless). This new formulation is in the same spirit as the two contextual homomorphism laws, in the colloquial form. In the control case, we describe the $k$ in $\beta\lambda k$: it behaves like a continuation; in the state case, we describe the update operation: what happens if we lookup afterwards, and what happens if we update before.

One remarkable aspect of the control algebra definition is that it carves out the set of continuations from a rather coarse description of it ($R^X$). This feature makes it particularly interesting.

The definition of control algebras is justified by the isomorphism theorem (p81):

**The isomorphism theorem.** *The category of control algebras, and the Eilenberg-Moore category of the continuation monad, are isomorphic; the isomorphism is given by the compose and decompose constructions* (Definition 4, p74).

In its proof, the step that once posed most difficulty was the diagonalisation principle (p87):

**The diagonalisation principle.** Let $(X, \beta)$ be a call/cc magma. It satisfies the diagonalisation principle if the following implication holds: For any $\gamma$, $\omega \in (X^{R^X})^{R^X}$, if for all $j \in R^X$, and under $\beta\lambda k^{\in R^X}.(-)jk$,

$$\gamma = \omega,$$

then under $\beta\lambda k^{\in R^X}.(-)kk$,

$$\gamma = \omega.$$

However, as the quote says, ‘In mathematics, it is either unknown or trivial.’ I probably will not make a fuss about it now, but I still think it is a key step in the proof of the isomorphism theorem. Also, readers should keep in mind that they can use the two contextual homomorphism laws in quite a liberal way; namely, the open variables of the laws can take a parameter $k$ that is bound in the context (the parameterised laws: p79, p86, p88).

Our results related to monadicity are the following.

**Theorem** (Corollary 2, p98)**.** *For any algebra of the control monad, the set of continuations is the coequaliser of the corresponding Beck's pair.*

**Theorem** (p98)**.** *A continuation from an algebra of the control monad is a global copoint of the algebra.*

**Theorem** (p104)**.** *A determined element of an algebra of the state monad is a global point of the algebra.*

There is a minor aspect that I would like to mention in passing. When $R$ has at least two elements, the exponential functor $R^{(-)}$ is monadic. We have

$$\mathrm{ctn} R^A \cong A \quad \text{and} \quad R^{\mathrm{ctn}(X,\theta)} \cong (X,\theta)$$

for any set $A$ and any algebra of the continuation monad, $(X,\theta)$. Here we write $R^A$ for the (algebra of the continuation monad that corresponds to the) intrinsic control algebra in $A$, and $\mathrm{ctn}(X,\theta)$ is the set of continuations from $(X,\theta)$. By the two isomorphisms, it seems justified to understand $\mathrm{ctn}(-)$ as *logarithm.* It would be good if we can write ‘log’ for that, and the isomorphisms

$$\log R^A \cong A \quad \text{and} \quad R^{\log(X,\theta)} \cong (X,\theta)$$

looks so much more natural. Similarly, when $S$ has at least one element, the power functor $(-)^S$ is monadic, and we have

$$\mathrm{de} A^S \cong A \quad \text{and} \quad \big(\mathrm{de}(X,\theta)\big)^S \cong (X,\theta)$$

for any set $A$ and any algebra of the state monad, $(X,\theta)$. Here $A^S$ is the algebra given by the comparison functor from $(-)^S$, which we may as well call it the intrinsic state algebra in $A$; $\mathrm{de}(X,\theta)$ is the set of determined elements of the algebra $(X,\theta)$. It seems justified to understand $\mathrm{de}(-)$ as the $S$-th *root.* It would be so much more natural if we can write

$$\sqrt[S]{A^S} \cong A \quad \text{and} \quad \sqrt[S]{(X,\theta)}^{\,S} \cong (X,\theta).$$

## 2 Future work

**Algebras of letstk and changestk** As we have mentioned in the introduction chapter, we consider the definition of control algebras as a stepping stone towards our eventual goal, which is to define the algebra of the letstk and changestk operations (in Levy's setting), or the catch and throw operations (in Crolard's setting). As explained in the introduction, the difficulty lies in the fact that a definition of such algebras shall never mention the result set $R$. The consequence is that equations that take place of the contextual abort-homomorphism law must inevitably draw in other such algebras. That means we may have to define all such algebras together, rather than defining each one separately. Here is my first attempt.

**Definition.** The category of stateful control algebras consists of

(i) A cartesian-close category $\mathcal{C}$,

(ii) Two functors

$$H\colon \mathcal{C}\times\mathcal{C}\to\mathcal{C} \quad E\colon \mathcal{C}\times\mathcal{C}^{\mathrm{op}}\to\mathcal{C}$$

(iii) For any $Y$, $X$ in $\mathcal{C}$, a morphism

$$\zeta_{Y,X}\colon EYY\to X$$

(iv) Pairs $(X,\beta)$ where $X$ is an object in $\mathcal{C}$, and $\beta$ is a morphism

$$HXX\to X.$$

The pairs should satisfy suitable equations involving the set $X$, the morphism $\beta$ and morphisms $\zeta_{Y,X}$.

We think $H$ as the functor $HXX = X^{R^X}$, and $E$ as the functor $EXX = X\times R^X$. A solution to the problem should open a door to the modelling of more subtle control behaviours.

**Properties of call/cc and abort** During the investigation of control algebras, I get the impression that they are of considerable depth, and there seems many remarkable properties yet to be documented. For example, here is one:

**Theorem.** *Take any control algebra, say with carrier $X$ and the call/cc operation $\beta$. For any $N$, $M\in X^{R^X}$, if they are identical on continuations from the control algebra, then $\beta N=\beta M$.*

Apart from those properties, there are some simple definitions and facts that I have to left out here due to a lack of time. Among those are combinations of call/cc and abort, and a normalisation theorem. As we mention in passing, combinations of call/cc and abort are unlike combinations in the state case; combinations of call/cc and abort are tree-like, rather than word-like. Nevertheless, we can still prove a normalisation theorem as in the state case. The theorem states that any combination of call/cc and abort can be normalised into the one given by the compose construction, and in a unique way. This should be an universal method in showing that some equation about call/cc and abort does hold. Other than that, the normalisation theorem is also interesting from the perspective of algebraic theory in general.

**Generalising control algebras to categories other than** Set We have developed the theory of control algebras in Set. It can be readily seen that most of the calculations only need cartesian-closedness to work out. We hope that control algebras can be generalised to other categories, say, the category of cpo's, so that recursion can also be included.

# Bibliography

[1] F. William Lawvere. “Functorial semantics of algebraic theories and some algebraic problems in the context of functorial semantics of algebraic theories”. Republished in Reprints in Theory and Applications of Categories, No. 5 (2004) pp1–121. PhD thesis. Columbia University, 1963 (cit. on pp. 12, 13).

[2] Peter J. Landin. “A generalization of jumps and labels”. In: *UNIVAC systems programming research report (unpublished)* (Aug. 29, 1965). Published in Higher-Order and Symbolic Computation, 11, 125–143, 1998 (cit. on p. 24).

[3] Jon Beck. “Triples, algebras and cohomology”. Available as *Reprints in Theory and Applications of Categories* 2 (2003). PhD thesis. 1967 (cit. on pp. 8, 91).

[4] J. Peter May. *Simplicial objects in algebraic topology.* The University of Chicago Press, 1967 (cit. on p. 93).

[5] John C. Reynolds. “Definitional interpreters for higher-order programming languages”. In: *Proceedings of the ACM National Conference.* Republished in Higher-Order and Symbolic Computation, 11, 363–397 (1998). 1972, pp. 717–740 (cit. on p. 24).

[6] Robert Paré. “Colimits in topoi”. In: *Bull. Amer. Math. Soc.* 80 (1974), pp. 556–561 (cit. on p. 7).

[7] Christopher Strachey and Christopher P. Wadsworth. *Continuations, a mathematical semantics for handling full jumps.* Technical Monograph PRG-11. Oxford University Computing Laboratory, 1974 (cit. on pp. 20, 23, 72).

[8] Gordon D. Plotkin. “Call-by-name, call-by-value and the λ-calculus”. In: *Theoretical Computer Science* 1 (1975) (cit. on p. 27).

[9] Ernest G. Manes. *Algebraic theories.* Springer-Verlag, 1976 (cit. on p. 6).

[10] Robert Milne and Christopher Strachey. *A Theory of Programming Semantics.* Chapman and Hall (Part A), Wiley (Part B), 1976 (cit. on p. 23).

[11] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics.* The MIT Press, 1977 (cit. on p. 23).

[12] Michael J. C. Gordon. *The Denotational Description of Programming Languages.* Springer, 1979 (cit. on p. 23).

[13] Guy L. Steele Jr. *Common Lisp: The Language.* Digital Press, 1984 (cit. on p. 24).

[14] Hal Abelson, Norman Adams, David Bartley, Gary Brooks, William Clinger (ed.), Dan Friedman, Robert Halstead, Chris Hanson, Chris Haynes, Eugene Kohlbecker, Don Oxley, Kent Pitman, Jonathan Rees, Bill Rozas, Gerald Jay Sussman, and Mitchell Wand. *The Revised Revised Report on Scheme, or An UnCommon Lisp.* AI Memo No. 848. 1985 (cit. on pp. 24, 47).

[15] Michael Barr and Charles Wells. *Toposes, Triples and Theories.* (Available online from the authors, version 1.3, 2019). Springer, 1985 (cit. on pp. 9, 13).

[16] Matthias Felleisen and Daniel P. Friedman. "Control operators, the SECD-machine, and the $\lambda$-calculus". In: *Formal Description of Programming Concepts III.* 1986, pp. 193–217 (cit. on pp. 18, 24, 29, 47, 73).

[17] Matthias Felleisen, Daniel P. Friedman, Eugene Kohlbecker, and Bruce Duba. "Reasoning". In: 1986 (cit. on pp. 18, 29, 47).

[18] David Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, 1986 (cit. on p. 23).

[19] Matthias Felleisen. "Reflections on Landin's **J**-operator: a partly historical note". In: *Computer Languages* 12.3/4 (1987), pp. 127–207 (cit. on p. 24).

[20] Matthias Felleisen, Daniel P. Friedman, Eugene Kohlbecker, and Bruce Duba. "A syntactic theory of sequential control". In: *Theoretical Computer Science* 52 (1987), pp. 205–237 (cit. on pp. 18, 29, 31, 47, 53).

[21] Matthias Felleisen. "The theory and practice of first-class prompts". In: *POPL '88. Proceedings of the 15th ACM Symposium on Principles of Programming Languages.* Jan. 1988 (cit. on pp. 20, 53).

[22] Philip J. Scott and Joachim Lambek. *Introduction to Higher-Order Categorical Logic.* Cambridge University Press, 1988 (cit. on p. 5).

[23] Eugenio Moggi. *An abstract view of programming languages.* Tech. rep. Report ECS-LFCS-90-113. University of Edinburgh, 1989 (cit. on pp. 14, 15).

[24] Olivier Danvy and Andrzej Filinski. "Abstracting control". In: *LFP '90: Proceedings of the 1990 ACM conference on LISP and functional programming.* May 1990 (cit. on pp. 20, 54).

[25] Timothy G. Griffin. "A formulae-as-types notion of control". In: *Conference record of the seventeenth annual ACM symposium on Principles of Programming Languages.* 1990 (cit. on pp. 18, 29, 31, 32, 47, 51, 52).

[26] Dorai Sitaram and Matthias Felleisen. "Control delimiters and their Hierarchies". In: *Lisp and symbolic compuation* 3 (1990), pp. 67–99 (cit. on pp. 20, 54).

[27] Eugenio Moggi. "Notions of computation and monads". In: *Information and computation* 93.1 (1991), pp. 55–92 (cit. on pp. 3, 13, 14).

[28] R. D. Tennent. *Semantics of Programming Languages.* Prentice-Hall, 1991 (cit. on p. 23).

[29] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* The MIT Press, 1992 (cit. on p. 23).

[30] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic.* Springer, 1992 (cit. on p. 93).

[31] Michel Parigot. "$\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction". In: *Proceedings of the International Conference on Logic Programming and Automated Reasoning.* LNCS 624. Springer, 1992, pp. 190–201 (cit. on pp. 18, 34–36, 38).

[32] John C. Reynolds. "The discoveries of continuations". In: *Lisp and Symbolic Computation* 6 (1993), pp. 233–247 (cit. on pp. 24, 72).

[33] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* The MIT Press, 1993 (cit. on p. 23).

[34] Francis Borceux. *Handbook of categorical algebra. 2. Categories and structures.* Cambridge University Press, 1994 (cit. on pp. 9, 12).

[35] Philippe de Groote. “On the relation between the $\lambda\mu$-calculus and the syntatic theory of sequential control”. In: *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning.* 1994 (cit. on pp. 29, 34).

[36] Martin Hofmann and Thomas Streicher. “Continuation models are universal for $\lambda\mu$-calculus”. In: *Proceedings, 12th Symposium on Logic in Computer Science.* 1997 (cit. on pp. 27, 32, 34, 36, 38, 39).

[37] Eugenio Moggi. “Metalanguages and applications”. In: *Semantics and Logics of Computation.* Cambridge University Press, 1997 (cit. on pp. 14, 15).

[38] John Power. “Modularity in denotational semantics”. In: *Thirteenth Annual Conference on Mathematical Foundations of Progamming Semantics, MFPS 1997, Carnegie Mellon University, Pittsburgh, PA, USA, March 23-26, 1997.* Ed. by Stephen D. Brookes and Michael W. Mislove. Vol. 6. Electronic Notes in Theoretical Computer Science. Elsevier, 1997, pp. 293–307. DOI: 10.1016/S1571-0661(05)80153-7 (cit. on p. 15).

[39] Hayo Thielecke. “Categorical structure of continuation passing style”. PhD thesis. University of Edinburgh, 1997 (cit. on pp. 20, 24, 28).

[40] Saunders Mac Lane. *Categories for the Working Mathematician.* 2ed. Springer, 1998 (cit. on pp. 8, 13).

[41] John Power and Giuseppe Rosolini. “A modular approach to denotational semantics”. In: *Automata, Languages and Programming, 25th International Colloquium, ICALP’98, Aalborg, Denmark, July 13-17, 1998, Proceedings.* Ed. by Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel. Vol. 1443. Lecture Notes in Computer Science. Springer, 1998, pp. 351–362. DOI: 10.1007/BFB0055066 (cit. on p. 15).

[42] Thomas Streicher and Bernhard Reus. “Classical Logic, Continuation Semantics and Abstract Machines”. In: *Journal of Functional Programming* 8.6 (1998), pp. 543–572 (cit. on pp. 18, 27, 29, 52).

[43] Hayo Thielecke. “An introduction to Landin’s “A generalization of jumps and labels””. In: *Higher-Order and Symbolic Computation* 11 (1998), pp. 117–123 (cit. on p. 24).

[44] Tristan Crolard. “A confluent $\lambda$-calculus with a catch/throw mechanism”. In: *Journal of Functional Programming* 9.6 (1999), pp. 625–647 (cit. on pp. 3, 18, 20, 29, 34–36, 39, 47, 54, 55).

[45] John Power. “Enriched Lawvere theories”. In: *Theory Appl. Categ.* 6 (1999), pp. 83–93 (cit. on p. 15).

[46] Hayo Thielecke. “Using a continuation twice and its implications for the expressive power of texttttcall/cc”. In: *Higher-Order and Symbolic Computation* 12 (1999), pp. 47–73 (cit. on pp. 24, 32).

[47] Paul Blain Levy. “Call-by-push-value”. PhD thesis. Queen Mary, University of London, 2001 (cit. on pp. 2, 16).

[48] Gordon D. Plotkin and John Power. “Adequacy for algebraic effects”. In: *Foundations of Software Science and Computation Structures, 4th Conference, FOSSACS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001, Proceedings.* Ed. by Furio Honsell and Marino Miculan. Vol. 2030. Lecture Notes in Computer Science. Springer, 2001, pp. 1–24. DOI: 10.1007/3-540-45315-6_1 (cit. on pp. 13, 14).

[49] Gordon D. Plotkin and John Power. "Semantics for algebraic operations". In: *Seventeenth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2001, Aarhus, Denmark, May 23-26, 2001.* Ed. by Stephen D. Brookes and Michael W. Mislove. Vol. 45. Electronic Notes in Theoretical Computer Science. Elsevier, 2001, pp. 332–345. DOI: 10.1016/S1571-0661(04)80970-8 (cit. on pp. 14, 16).

[50] Peter Selinger. "Control categories and dualtiy: on the categorical semantics of the lambda-mu calculus". In: *Mathematical Structures in Computer Science* 11 (2001), pp. 207–260 (cit. on pp. 18, 27, 32, 34–36, 38, 39, 52).

[51] Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium.* Vol. 1. Oxford University Press, 2002 (cit. on p. 8).

[52] Gordon D. Plotkin and John Power. "Computational effects and operations: an overview". In: *Proceedings of the Workshop on Domains VI 2002, Birmingham, UK, September 16-19, 2002.* Ed. by Martín Escardó and Achim Jung. Vol. 73. Electronic Notes in Theoretical Computer Science. Elsevier, 2002, pp. 149–163. DOI: 10.1016/J.ENTCS.2004.08.008 (cit. on pp. 14, 16).

[53] Gordon D. Plotkin and John Power. "Notions of computation cetermine monads". In: *Foundations of Software Science and Computation Structures: 5th International Conference, Proceedings.* Springer, 2002, pp. 342–356 (cit. on pp. 3, 13, 14, 16, 17, 61, 105).

[54] Edmund Robinson. "Variations on algebra: monadicity and generalisations of equational theories". In: *Formal Aspects of Computing* 13.3-5 (2002), pp. 308–326 (cit. on pp. 12, 16, 17).

[55] Paul Blain Levy. "Adjunction models for Call-by-Push-Value with stacks". In: *Electronic Notes in Theoretical Computer Science* 69 (2003) (cit. on p. 16).

[56] Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperitive Synthesis.* Kluwer Academic Publishers, 2003 (cit. on pp. 2, 16, 18, 20, 28, 32, 33, 39, 54, 55, 73).

[57] Gordon D. Plotkin and John Power. "Algebraic operations and generic effects". In: *Appl. Categorical Struct.* 11.1 (2003), pp. 69–94. DOI: 10.1023/A:1023064908962 (cit. on p. 14).

[58] Tristan Crolard. "A formulae-as-types interpretation of subtractive logic". In: *Journal of Logic and Computation* 14.4 (Aug. 2004). Special issue on intuitionistic modal logic and application, pp. 529–570 (cit. on p. 39).

[59] Carsten Führmann and Hayo Thielecke. "On the call-by-value CPS transform and its semantics". In: *Inf. Comput.* 188.2 (2004), pp. 241–283. DOI: 10.1016/J.IC.2003.08.001 (cit. on p. 39).

[60] John MacDonald and Manuela Sobral. "Aspects of monads". In: *Categorical Foundations. Special Topics in Order, Topology, Algebra and Sheaf Theory.* Ed. by Maria Cristina Pedicchio and Walter Tholen. Cambridge University Press, 2004 (cit. on p. 9).

[61] François Métayer. "State monads and their algebras". In: *arXiv e-prints* (2004) (cit. on pp. 8, 10, 92, 102, 106).

[62] Maria Cristina Pedicchio and Fabrizio Rovatti. "Algebraic categories". In: *Categorical Foundations. Special Topics in Order, Topology, Algebra and Sheaf Theory.* Ed. by Maria Cristina Pedicchio and Walter Tholen. Cambridge University Press, 2004 (cit. on pp. 8, 12).

[63] G. Max Kelly. "Basic concepts of enriched category theory". In: *Theory Appl. Categ.* 10 (2005). Reprint of the 1982 original, pp. 1–136 (cit. on p. 97).

[64] Paul Blain Levy. "Adjunction models for call-by-push-value with stacks". In: *Theory Appl. Categ.* 14 (2005), pp. 75–110 (cit. on pp. 16, 45).

[65] Martin Hyland, Gordon D. Plotkin, and John Power. "Combining effects: sum and tensor". In: *Theor. Comput. Sci.* 357.1-3 (2006), pp. 70–99. DOI: 10.1016/J.TCS.2006.03.013 (cit. on pp. 15, 16).

[66] Bachuki Mesablishvili. "Monads of effective descent type and comonadicity". In: *Theory Appl. Categ.* 16.1 (2006), pp. 1–45 (cit. on pp. 7, 9, 92).

[67] Martin Hyland, Paul Blain Levy, Gordon D. Plotkin, and John Power. "Combining algebraic effects with continuations". In: *Theoretical Computer Science* 375.1–3 (2007), pp. 20–40 (cit. on pp. 7, 8, 15–17, 92).

[68] Martin Hyland and John Power. "The category theoretic understanding of universal algebra: Lawvere theories and monads". In: *Computation, meaning, and logic: articles dedicated to Gordon Plotkin.* Vol. 172. Electron. Notes Theor. Comput. Sci. Elsevier, 2007, pp. 437–458 (cit. on pp. 12, 13, 16).

[69] Mauro Jaskelioff. "Modular monad transformers". In: *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings.* Ed. by Giuseppe Castagna. Vol. 5502. Lecture Notes in Computer Science. Springer, 2009, pp. 64–79. DOI: 10.1007/978-3-642-00590-9_6 (cit. on p. 14).

[70] Gordon D. Plotkin and Matija Pretnar. "Handlers of algebraic effects". In: *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22–29, 2009, Proceedings.* Vol. 5502. Lecture Notes in Computer Science. 2009 (cit. on pp. 14, 18, 20).

[71] Martín Escardó and Paulo Oliva. "Selection functions, bar recursion, and backward induction". In: *Mathematical Structures in Computer Science* 20.2 (2010), pp. 127–168 (cit. on pp. 20, 75, 84).

[72] Mauro Jaskelioff and Eugenio Moggi. "Monad transformers as monoid transformers". In: *Theoretical Computer Science* 411.51–52 (2010), pp. 4441–4466 (cit. on pp. 6, 14, 15).

[73] Paul-André Melliès. "Segal condition meets computational effects". In: *Proceedings of 25th Annual IEEE Symposium on Logic in Computer Science.* 2010, pp. 150–159 (cit. on pp. 4, 14, 18, 61).

[74] Emily Riehl. *Categorical theory in context.* Also freely available at emilyriehl.github.io/files/context.pdf. Dover Publications, 2016 (cit. on p. 8).

[75] Paul Blain Levy. "Call-by-push-value". In: *ACM SIGLOG News* 9.2 (Apr. 2022), pp. 7–29 (cit. on pp. 2, 16, 53, 54).

[76] Paul Blain Levy. *Martin-Löf clashes with Griffin, operationally.* https://cs.bham.ac.uk/~pbl/papers/index.html. Manuscript (cit. on p. 24).

[77] Pietro Cenciarelli and Eugenio Moggi. "A syntactic approach to modularity in denotational semantics". In: 1993 (cit. on p. 15).