

An Algebraic Theory of Type-and-Effect Systems

Ohad Kammar



Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2014

Abstract

We present a general semantic account of Gifford-style type-and-effect systems. These type systems provide lightweight static analyses annotating program phrases with the sets of possible computational effects they may cause, such as memory access and modification, exception raising, and non-deterministic choice. The analyses are used, for example, to justify the program transformations typically used in optimising compilers, such as code reordering and inlining. Despite their existence for over two decades, there is no prior comprehensive theory of type-and-effect systems accounting for their syntax and semantics, and justifying their use in effect-dependent program transformation.

We achieve this generality by recourse to the theory of algebraic effects, a development of Moggi’s monadic theory of computational effects that emphasises the operations causing the effects at hand and their equational theory. The key observation is that annotation effects can be identified with the effect operations.

Our first main contribution is the uniform construction of semantic models for type-and-effect analysis by a process we call *conservative restriction*. Our construction requires an algebraic model of the unannotated programming language and a relevant notion of predicate. It then generates a model for Gifford-style type-and-effect analysis. This uniform construction subsumes existing ad-hoc models for type-and-effect systems, and is applicable in all cases in which the semantics can be given via *enriched Lawvere theories*.

Our second main contribution is a demonstration that our theory accounts for the various aspects of Gifford-style effect systems. We begin with a version of Levy’s Call-by-push-value that includes algebraic effects. We add effect annotations, and design a general type-and-effect system for such call-by-push-value variants. The annotated language can be thought of as an intermediate representation used for program optimisation. We relate the unannotated semantics to the conservative restriction semantics, and establish the soundness of program transformations based on this effect analysis. We develop and classify a range of validated transformations, generalising many existing ones and adding some new ones. We also give modularly-checkable sufficient conditions for the validity of these optimisations.

In the final part of this thesis, we demonstrate our theory by analysing a simple example language involving global state with multiple regions, exceptions, and non-determinism. We give decision procedures for the applicability of the various effect-dependent transformations, and establish their soundness and completeness.

Lay summary

We instruct computers about their tasks using detailed textual descriptions, called “programs”, encoded in a precisely defined “programming language”. Another program, the “compiler”, then translates our textual description into a description the machine can execute. The compiler may change, or “optimise”, the translated program in order to achieve better performance, for example to make the program execute faster, or extend battery-life. One example for such “transformations” is the reordering of instructions.

Our programs may cause “computational effects” besides computing their end results, such as display an image on a monitor, respond to a keyboard stroke, and print a document. Some transformations become invalid in the presence of effects, causing tension between the two. For example, consider a program that first displays a message on the screen and then waits for the user to press a key on the keyboard. Our compiler must not reorder these two tasks during optimisation, or else the program will wait for the user to hit the keyboard before it displays the message to do so.

To ensure the optimisation process is correct, some compilers analyse the program and attach to each part a summary of the effects this part may cause. In the above example the part of the program that displays the message will be annotated with the effect “output”, and the part of the program that reads the user’s keystroke will be annotated “input”. This annotation is called a “type-and-effect system”. The compiler’s designers carefully study the programming language at hand and develop conditions that ensure the optimisations are indeed safe. For example, parts annotated with “output” should never be reordered with parts annotated with “input”. Despite their existence for over two decades, there is no prior comprehensive account of type-and-effect systems beyond a case-by-case study. This thesis develops such a general account.

We rely on the theory of “algebraic effects”, which describes many computational effects through equations and operations. This account combines the abstract “input” and “output” effects in a fashion reminiscent of how we combine addition and multiplication in elementary school algebra. Our first main contribution is a general mathematical description of these effect annotations based on such an algebraic description of the programming language at hand. Our second main contribution is a demonstration that our theory accounts for the various aspects of type-and-effect systems: it describes the effect annotations and their meaning; it validates the optimisations; and it justifies that the optimisation process does not change the meaning of the original program. Finally, we demonstrate our theory by analysing a simple example language.

Preface by Andrej Bauer

The following preface was composed using the Up-Goer-Six Text Editor¹ and (mostly) uses the 1,000 most common English words².

Mr. Ohad is about to become a doctor (the computer kind, not the kind that helps sick people). But first he must explain to normal people like you and I what he did. Most soon-to-be doctors find this a very hard thing to do.

Anyway, Mr. Ohad soon-to-be-doctor studied how to tell computers what to do. You may think there is not much to it, but in fact this is quite hard because computers do exactly what they are told, so we need to be very, very careful and precise when we give computers instructions. A good way to do this is to use lots of math, and so Mr. Ohad did it. He used algebra. Yes, everyone hated algebra in school, but Mr. Ohad did not, which is why he is about to become a doctor. Let me tell you what he did.

Instructions for computers are called “programs”. There are many different ways of writing these, and some turn out to be easier to use than others. Since humans are very sloppy, we try to come up with ways of writing programs that make it easy to spot errors and figure out what programs do.

Here is where algebra comes up. Mr. Ohad, on the advice of his mentor Dr. Plotkin, worked on how to turn stuff such as printing, playing music, and generally doing cool things with computers into good old boring algebra. No, he and his advisor are not insane, they are just mathematicians. Mr. Ohad perfected one angle in the lately popular idea that playing music and doing stuff with computers is really a lot like standing in front of a blackboard with a piece of chalk trying to solve an equation. OK, I am being funny. But seriously, this is the sort of thing he did. If you listened to him for about a day (and to your algebra teacher before that for about a year), you would understand why checking that a program works is like checking an equation. Believe it or not, but hitting the Print button in your word processor is actually the same sort of thing as adding two numbers. People like Mr. Ohad understand this, and much more.

I am skipping over funky words such as “effect handler” so I cannot really go into details here. But the gist of the matter is that he used cool math to improve lives of people who have to instruct computers what to do, and he improved your life by helping make sure that computers will do what we want them to do faster and better, and all he asks in return is that when you see him next time you call him Doctor Kammar.

¹<http://splasho.com/upgoer6/>

²The frequency list is taken from:
http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/TV/2006/explanation.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Ohad Kammar)

Acknowledgements

The existence and completion of this thesis is owed to Gordon Plotkin, who has been a model supervisor for the entire duration of my studies. No amount of praise and thanks can express my gratitude to his endless reservoirs of patience upon which he drew while taking the time to discuss my work, share his ideas, and offer useful and insightful criticism of the highest rank. Regardless of physical location and time-zone differences, he was always present with sharp perception, clarity of thought, and good humour to comment, guide, and inspire me. Gordon, thank you.

I am grateful to Nick Benton and Ian Stark for accepting the formidable task of examining this “brick”. Numerous hours of their valuable time have been spent in exploring and understanding its many nooks and crannies, and in my examination.

My path leading to this thesis began with two figures from the Open University of Israel: Vadim Grinshtein and Ehud Lamm. Vadim walked besides me in my first steps in mathematics, and not only taught me about correct proofs, but about the reflection and methodologies involved in producing them. He has since been a mentor, keeping up to date with my progress and never ceasing to share his experience. Ehud has exposed me to the wonderful discipline of programming language theory. Through hours of discussion, he introduced me to the beautiful interplay between theory and practice. Over the years, he maintained his mentoring, and offered advice and support.

Sam Staton played a crucial role in the final step of the production of this thesis. I am grateful for his interest in my work, for bringing me to Cambridge, and for many insightful observations and discussions. I hope we will share many more of the latter in years to come.

I am indebted to Alex Simpson, my secondary supervisor, for keeping up to date with my work and for discussing it. He shared his deep and broad of knowledge and identified new connections in my work. I am also thankful for his kind invitation to present my work at the first European Workshop on Computational Effects in Ljubljana, impacting both my PhD work, and my work on effect handlers.

Neil Ghani, my external supervisor, deserves my praise for watching my progress, supplying technical and career advice, and enabling my escapades west to the Mathematically Structured Programming group in the University of Strathclyde.

I thank the following individuals for their scientific help during my studies. Vincent Danos, for re-introducing me to the wonderful world of combinatorics, measures, and probabilities, and for my first publication. Sam Lindley, for his friendship, endless conversations and discussions, both technical and social, and for orchestrating our

libraries for effect handlers. Bob Atkey, for his many explanations and discussions, and for leading by example. Philip Wadler, for suggesting a PhD course in Edinburgh with Gordon, for patiently listening to my ideas and giving advice and example on how to best present them, and for providing me with countless timely recommendation letters. Peter Sewell and Glynn Winskel, for enabling me to remain in Cambridge and learn about concurrency in both theory and practice. Conor McBride, for narrating the wider context and history behind the science that we do, over a pint, for valuable career advice, and for his wonderful courses on dependently-typed programming. Paul Blain Levy, for his CBPV, and for interesting discussions and suggestions ever since my first steps at the Midlands Graduate School. Marcelo Fiore, for countless discussions, usually over lunch. Paul-André Melliès, for mnemoinds in particular. Andrej Bauer and Matija Pretnar, for following their vision on programming with algebraic effects and handlers and for giving me the opportunity to visit Ljubljana. Bob Harper, for inviting me to Carnegie-Mellon University to present my work, which turned out to be the last time I was to meet John Reynolds, and for a delightful dinner. Danel Ahman, for picking up where I left off and continuing much farther, and for many interesting discussions. Andy Gordon for suggesting openings in Cambridge and for discussions on refinement types. Elham Kashefi, for her infectious enthusiasm for anything quantum. Jeff Egger, for teaching me much category theory. John Power, for patiently answering my many questions. The occupiers of office IF-5.28 for putting up with my frequent visits and joining in on the conversation: Brian Campbell, Illias Garnier, James McKinna, Nicolas Oury, as well as Kenneth MacKenzie. To the members of the Mathematically Structured Programming group for putting up with my interruptive visits: Patricia Johann, Peter Hancock, Clement Fumex, Adam Gundry, Pierre-Evariste Dagand, Lorenzo Malatesta, Guillaume Allais, and Stevan Andjelkovic.

I thank the following colleagues in Edinburgh. Michael David Pedersen, the first person to welcome me to the School of Informatics in Edinburgh, the first person to welcome me to Cambridge, and a wonderful and loyal friend. I hope our paths will remain intertwined for long. Alireza Pourranjbar, for late night discussions about values, ideas, and science. The modelling group, for adopting me in the Wolfson Wing: Jane Hillston, Stephen Gilmore, Allan Clark, Vashti Galpin, and Maria Luisa Guerriero. To my dear office-mates in IF.3-50 for hours of useful distraction: Adam Duguid, Michael Smith, Mirco Tribastone, Nick O'Shea, Marek Kwiatkowski, Krzysztof Gorgolewski, Dimitris Milios, and Chris Banks. To my other fellow students in the Laboratory for Foundations of Computer Science, who helped me hone my research skills:

Ben Kavanagh, Ezra Cooper, Grant Passmore, Willem Heijltjes, Matteo Mio, Lorenzo Clemente, and Patrick Totzke. Björn Franke from the Institute for Computing Systems Architecture for having a view towards theory, and for career advice. Gaya Nadarajan, Silvia Pareti, Chris Fensch, Christophe Dubach, and Hugh Leather for their friendship and cultural activities.

I am grateful to my colleagues in Cambridge. Andy Pitts, Tim Griffin, Mike Gordon, and Jonathan Hayman for their conversations. To my dear office-mates in FS15: Jonas Frey and Marc Lasson, for their scientific, career, emotional, and personal support while writing this thesis, as well as for putting up with my various caprices and untidiness. To the other post-docs and students in the Logic, Semantics, and Programming Languages group, for scientific and social rapport: Pierre Clairambault, Mike Dodds, Kathy Gray, Gabriel Kerneis, Dominic Mulligan, Dominic Orchard, Scott Owens, Thomas Tuerk, Mark Batty, Steffen Lösch, Justus Matthiesen, Kayvan Memarian, and Simon Castellan.

I am grateful to the members of the category theory group in the University of Cambridge Department of Pure Mathematics and Mathematical Statistics for patiently answering my questions and tolerating my interruptions: Julia Goedecke, Filip Bar, Achilleas Kryptis, Sori Lee, Guilherme Lima, Zhen Lin Low, Paige North, Cristina Vasilakopoulou, and Tamara von Glehn.

I have much gratitude for Alexander Rabinovitch for arranging my multiple visits to the Blavatnik School of Computer Science in Tel-Aviv University. The wonderful office-space he supplied me with was the haven I needed to write this thesis.

I express my profound gratitude to the numerous administrators in the University of Edinburgh who enabled my course to take place, with honourable mentions to the following individuals. Magdalena Mazurczak, for doing above and beyond with an ever-present smile. Monika Lekuse, for remembering the people behind the numbers, and for her career advice. Sarah Wyse, for dealing with the various aspects of my submission, and accepting my thesis on Tuesday, 30 June, 2013.

I am indebted to the Edinburgh Hebrew Community for their endless support during my stay in Edinburgh. Stephanie and Josh Brickman, for adopting me into their family, with stimulating conversations, games, and many festive and Shabbat meals at their table. Rabbi David Rose, for being a dear and supporting friend, and for connecting me to the wider context in which we live. Jackie and Raymond Taylor and their sons. And to the Ansell, the Griffins, the Rifkinds, and the Sperbers. The members and leaders of the JSocs in Scotland and Cambridge for many arguments and joys: Li-

uba Adar, Daniel Barnett, Brett Bernstein, David Bobick, Dan Cooper, Sarah Covshoff, Liorah Dekel, Morris Griffin, Ellen Hockley, Samantha Kulok, Ben Leibowitz, Amit Mander, Adam Nuhi, Daniel Pinchassoff, Adina Roth, Reuben Sagar, David Shaw, Reuven Shirazi, and Arilol Weiss. I also thank the Jewish chaplains Rabbi Gary and Susan Wayland, and Rabbi Yisrael and Elisheva Malkiel, and to Rabbi Reuven and Rochel Leigh.

I am also grateful to Ezra and Sam L. for introducing me to the joy of Ultimate Frisbee, which, apart from a few broken fingers, was a constructive distraction from my research. I am thankful to the members of the University of Edinburgh Ultimate Club, Ro Sham Bo, the Edinburgh Ultimate Club, Sneekys, and the Cambridge Ultimate club, Strange Blue, in particular to Oliver Browne, Melanie Craxton, Conor Gaffney, Chris Rawson, Colin Shearer, Nick Skliar-Davies, and Ross Walder.

I am grateful to Hephtzi Plotkin for her wonderful resourcefulness and kindness, providing hospitality and making me feel welcome in the most difficult conditions.

Special thanks are reserved for Lucy Bates, Laurentiu Prodan, and Zac Waldman for valuable feedback on the Lay Summary of this thesis.

The work leading to this thesis was directly supported by a Scottish Informatics and Computer Science Alliance scholarship, a University of Edinburgh School of Informatics studentship, the Isaac Newton Trust starter grant “algebraic theories, computational effects, and concurrency”, and Engineering and Physical Sciences Research Council grant EP/H005633/1.

Finally, I cannot thank my dear family enough for their unfailing support throughout the years, and for being my anchor through difficult times: Mother Esti, Father Yossi, Hagay, Sandy, Netta, Reut, Ariad, and Orel.

Contents

1	Introduction	15
1.1	Effect-dependent optimisation	16
1.2	Denotational semantics for effect systems	18
1.3	An algebraic solution	22
1.4	Our advantage	24
1.5	Thesis structure	25
I	Semantic foundations	29
2	Algebraic operations	31
2.1	Categorical preliminaries	31
2.2	Algebraic operations	33
2.3	Generic effects	37
2.4	CBPV models	44
3	Models	47
3.1	Categorical models	48
3.2	Generic models	50
3.3	Set-theoretic models	53
4	Recursion	61
4.1	Domain-theoretic preliminaries	62
4.2	ω CPO-enriched CBPV models	65
4.3	ω CPO-enriched algebraic operations	73
4.4	Recursion models	76
4.5	Domain-theoretic models	78

5	Locally presentable categories	87
5.1	Directed colimits	87
5.2	Presentable objects	93
5.3	Locally presentable categories	97
5.4	Countably presentable domains	101
6	Lawvere theories	113
6.1	Enriched powers and copowers	113
6.2	Enriched Lawvere theories	131
6.3	Algebraic operations	144
6.4	Algebraic CBPV models	146
7	Algebraic models	149
7.1	Categorical algebraic models	149
7.2	Factorisation systems	151
7.3	Conservative restriction	168
8	Presentation models	179
8.1	Universal algebra and equational logic	179
8.2	Universal algebra and monads	187
8.3	Conservative and surjective translations	190
8.4	Presentation models	193
9	Relational models	199
9.1	Set-theoretic logical relations	199
9.2	Monadic lifting	202
II	Effect-dependent optimisation	213
10	Generic intermediate language	215
10.1	CBPV with algebraic operations	215
10.2	Multi-adjunctive intermediate languages	228
10.3	Model generation	242
11	Optimisations	247
11.1	Validity	248
11.2	Optimisation taxonomy	252

<i>CONTENTS</i>	13
12 Combining effects	257
12.1 Combining theories	258
12.2 Operation-wise validity	266
12.3 Ad-hoc combination	271
13 Use case	287
13.1 Language and semantics	288
13.2 Optimisation validation	290
13.3 Completeness	293
13.4 Mechanised analysis	302
14 Conclusion	323
A Conventions	329
Bibliography	331

Chapter 1

Introduction

It took me all night

To get hold of the right introduction

—Queen

Pure functional programs are amenable to equational reasoning that can be used for program optimisation [dMS95]. Practical programs are *not* pure, involving *computational effects* such as memory and file access. Unfortunately, computational effects violate the validity of equational reasoning. However, not every possible equational transformation is violated by every computational effect. Code fragments restricted to a smaller set of effects may be amenable to some of the equational transformations pure code satisfies. *Gifford-style type-and-effect systems* [LG88] are a light-weight static analysis for annotating each code fragment with a set of effects it may cause. These effect-annotations provide a basis for effect-dependent optimisation [Tol98, BKR98].

This thesis concerns the semantic validity of such effect-dependent optimisations. The *denotational approach* to semantics provides a natural setting for validating such equational optimisations using *denotational equivalence*. Moggi [Mog89, Mog91] established that collections of effects denote *monads*. Based on his observation, Tolmach [Tol98] and Benton et al. [BKR98] independently validate effect-dependent optimisations denotationally. However, their approach suffers from the following drawback, independently posed by Wadler [Wad98]:

“As hypothesised by Moggi and as born out by practice, most computational effects can be viewed as a monad. Does this provide the possibility to formulate a general theory of effects and monads, avoiding the need to create a new effect system for each new effect?”

In this thesis we propose a general theory of type-and-effect systems, and answer Wadler’s question in the affirmative.

1.1 Effect-dependent optimisation

Consider the following illustrative ML-like code:

```

1  let z = 2 in
2  let f = fun b -> if z*z > 1
3                      then x := if b
4                              then z*z
5                              else 4
6                      else x := !y + z*z in
7  let c = getchar () in
8  f (c = 'y')
9  f (c = 'y')
```

This program mixes functional code with *computational effects*: lines 3 and 6 update a global memory location x , line 6 looks-up some global state, and line 7 reads a character from the user.

An *optimising compiler* may transform this program to an equivalent program that is better in some way. The precise nature of the improvement is not the focus of this thesis, and it may include run-time efficiency, space utilisation, code size, or energy consumption. For example, the compiler may transform lines 3–6 to the following code, pre-computing $z*z$ and reusing the result:

```

let u = z*z
if u > 1
then x := if b
           then u
           else 4
else x := !y + u
```

However, the presence of computational effects introduces subtlety to the optimisation process. For example, the following two program phrases are *not* equivalent:

```

x := !x+1
x := !x+1

let u = !x+1 in
x := u
x := u
```

Fortunately, not all computational effects rule out this optimisation. If y is a different global memory location, the following two phrases *are* equivalent:

$x := !y+1$ $x := !y+1$	$\mathbf{let\ } u = !y+1 \mathbf{\ in}$ $x := u$ $x := u$
----------------------------	---

More generally, if the only effects a program phrase may cause are restricted to reading from one memory region ρ_1 and writing to a different memory region ρ_2 , this phrase may be safely reused. Thus, the compiler may first analyse the program to determine which computational effects each phrase causes. For this purpose, in the presence of higher-order functions, the simplest such effect analysis is a *Gifford-style type-and-effect system* [LG88], which assigns to each phrase both a type and a *set* of effects it may cause. Thus, the **then** branch on lines 3–5 will be assigned the type `int` and the effect set $\{\text{read}_{\rho_1}\}$. More formally, Gifford and Lucassen introduced type-and-effect judgements $\Gamma \vdash M : A! \epsilon$, which state that M has type A and effect ϵ in context Γ . For example:

$$b : \text{bool}, x : \text{Loc}_{\rho_1} \vdash x := \mathbf{if\ } b \mathbf{\ then\ } z*z \mathbf{\ else\ } 4 : \text{unit}! \{\text{write}_{\rho_1}\}$$

The typing rules then propagate effects, for example:

$$\frac{\Gamma \vdash M : \text{bool}! \epsilon_1 \quad \Gamma \vdash N : B! \epsilon_2 \quad \Gamma \vdash K : B! \epsilon_3}{\Gamma \vdash \mathbf{if\ } M \mathbf{\ then\ } N \mathbf{\ else\ } K : B! \epsilon_1 \cup \epsilon_2 \cup \epsilon_3}$$

Note that we gloss over how to *slice* the global memory locations into regions, as it is not the focus of this thesis. Henglein et al. [HMN05, Section 3.6] discuss such effect inference, focussing on Tofte and Birkedal’s inference algorithm [TB98, BT01].

Crucially, type-and-effect systems admit higher-order treatment. We refine function types $A \rightarrow B$ with effect annotations $A \xrightarrow{\epsilon} B$ that state that the functions may cause effects in ϵ during their execution. The rules for abstraction and application propagate the effects as follows:

$$\frac{\Gamma, x : A \vdash M : B! \epsilon}{\Gamma \vdash \lambda x : A. M : (A \xrightarrow{\epsilon} B)! \emptyset} \qquad \frac{\Gamma \vdash M : A \xrightarrow{\epsilon} B! \epsilon' \quad \Gamma \vdash N : A! \epsilon''}{\Gamma \vdash MN : B! \epsilon \cup \epsilon' \cup \epsilon''}$$

Thus, the type-and-effect assigned to the function `f` in our sample program is:

$$x : \text{Loc}_{\rho_1}, y : \text{Loc}_{\rho_2}, z : \text{int} \vdash f : (\text{bool} \xrightarrow{\{\text{write}_{\rho_1}, \text{read}_{\rho_2}\}} \text{int})! \emptyset$$

Using such type-and-effect systems (*effect systems* for brevity), we can now precisely describe when it is safe to reuse a piece of code:

$$\Gamma \vdash M : A! \varepsilon \quad \Gamma, x : A, y : A \vdash N : B! \varepsilon' \quad \left(\begin{array}{l} \text{for all } \rho, \{\text{read}_\rho, \text{write}_\rho\} \not\subseteq \varepsilon \\ \varepsilon \subseteq \bigcup_{\rho'} \{\text{read}_{\rho'}, \text{write}_{\rho'}\} \end{array} \right)$$

let $x = M$ in let $y = M$ in N	=	let $x = M$ in let $y = x$ in N
---	---	---

Thus, an optimising compiler should implement a decision procedure guaranteeing the side condition before transforming the source code to an equivalent variant using this equation. Such a compiler may then transform lines 8–9 in the sample program above as follows, using the effect-dependent optimisation in the transition marked (*):

$f(c = 'y')$ $f(c = 'y')$	=	let $u = f(c = 'y')$ in let $v = f(c = 'y')$ in v	(*) \downarrow =	let $u = f(c = 'y')$ in let $v = u$ in v
	=	let $u = f(c = 'y')$ in u	=	$f(c = 'y')$

This thesis addresses three issues arising from this situation. First, we need to guarantee that the side conditions on the optimisations are *sound*: if the side condition holds, then applying the transformation does result in equivalent code. In order to do so, we need to relate the syntactic information included in the effect system to the semantics of the program [Ben96]. Thus, the second issue we address is the semantics of the effect annotated language, and its relationship to the original, unannotated code. Finally, we would like the side condition to be *complete*: if it is safe to apply this optimisation based on the effect annotations, then the side condition is satisfied. For example, the computational effects in our source language may also include *exceptions*. Then, the side condition given above is *not* complete: the optimisation is valid in the presence of exceptions, i.e., ε may contain the ‘throw’ effect.

1.2 Denotational semantics for effect systems

Our goal is to study the validity of equations between annotated program terms. Therefore, the natural approach to the semantics of effect systems is *denotational*. Briefly, in

the denotational approach each program phrase is assigned a mathematical object that captures its meaning by composing the meanings of its sub-phrases. Thus, such semantics for an effect annotated language makes *denotational equality* a natural notion of validity for equations between effect-analysed program phrases.

Historically, the validation of effect-dependent optimisations using the denotational approach only took place following Moggi's unification of effectful denotational semantics using *monads* [Mog89, Mog91], and its popularisation by Wadler [Wad90, Wad92]. With monads in place, the connection between type-and-effect systems and monads was independently published during the same year by Wadler [Wad98], Tolmach [Tol98], and Benton et al. [BKR98].

To describe the connection concretely, consider a language whose effects are interactions with a single global memory cell capable of storing values from some set \mathbb{V} . We may interact with this memory cell in two ways: we either *look-up* the stored value, or *update* it to store a new value. Languages involving such a memory cell are modeled by the *global state* monad $T_{GS}X := (\mathbb{V} \times X)^{\mathbb{V}}$. Thus, a computation is modeled by a function $\lambda v. \langle u_v, x_v \rangle$ mapping an initial cell state v to its new state u_v following the execution, and the return value of the computation x_v . For example, the function $\lambda v. \langle 42, \text{True} \rangle$ models a computation that modifies the cell contents to 42 and returns the boolean value `True`.

One way to model the look-up interaction is by the following family of functions:

$$\begin{aligned} O \llbracket \text{lookup} \rrbracket_X &: (T_{GS}X)^{\mathbb{V}} && \rightarrow T_{GS}X \\ O \llbracket \text{lookup} \rrbracket_X &: \lambda v_1. (\lambda v_2. \langle u_{v_1, v_2}, x_{v_1, v_2} \rangle) && \mapsto \lambda v. \langle u_{v, v}, x_{v, v} \rangle \end{aligned}$$

The domain of the function, $(T_{GS}X)^{\mathbb{V}}$, models a given family of computations parametrised by the value v_1 currently stored in the memory cell. Given such a family, $\kappa = \lambda v_1. (\lambda v_2. \langle u_{v_1, v_2}, x_{v_1, v_2} \rangle)$, the function $O \llbracket \text{lookup} \rrbracket (\kappa)$ looks-up the current value v to obtain the computation $\kappa(v) = \lambda v_2. \langle u_{v, v_2}, x_{v, v_2} \rangle$. As such look-ups do not affect the value stored in the cell, the execution is modeled by applying the selected computation, $\kappa(v)$, to v , resulting in $\kappa(v)(v) = \lambda v. \langle u_{v, v}, x_{v, v} \rangle$.

Similarly, updates are modeled by a family of functions

$$\begin{aligned} O \llbracket \text{update} \rrbracket_X &: T_{GS}X && \rightarrow (T_{GS}X)^{\mathbb{V}} \\ O \llbracket \text{update} \rrbracket_X &: \lambda v. \langle u_v, x_v \rangle && \mapsto \lambda v_0. \lambda v. \langle u_{v_0}, x_{v_0} \rangle \end{aligned}$$

Thus, given a computation $k = \lambda v. \langle u_v, x_v \rangle$ to be executed after the update, and given a value v_0 to update the cell contents to v_0 , we model the update by ignoring, or over-

writing, the currently stored value v , and passing to k the new value v_0 as the currently stored state, resulting in $k(v_0) = \lambda v. \langle u_{v_0}, x_{v_0} \rangle$.

Next, consider a language whose effects are interactions with a single global *read-only* memory cell for \mathbb{V} . In this scenario, the only interaction of interest is a memory look-up. This language is modeled by the *environment* monad $T_{\text{Env}}X := X^{\mathbb{V}}$. The look-up interaction is modeled by the family:

$$\begin{aligned} O[\text{lookup}]_X &: (T_{\text{Env}}X)^{\mathbb{V}} \rightarrow T_{\text{Env}}X \\ O[\text{lookup}]_X &: \lambda v_1. (\lambda v_2. x_{v_1, v_2}) \mapsto \lambda v. x_{v, v} \end{aligned}$$

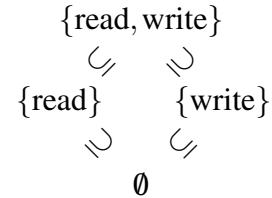
This time there is no need to return a new state for the cell, as the computation may not change it.

Similarly, consider a language whose effects are interactions with a single global *write-only* cell for \mathbb{V} . In this case, the only interaction of interest is a memory update. This language is modeled by the *overwrite* monad $T_{\text{OW}}X := (\mathbb{1} + \mathbb{V}) \times X$, a particular instance of the more general *writer* monad, also called the *complexity* monad, and the *monoid* monad. These computations, in addition to returning an X -value after execution, summarise the change in the memory cell. The left injection, $\iota_1 \star$, models a computation that leaves the cell unchanged. The right injections, $\iota_2 v$, where $v \in \mathbb{V}$, model computations that change the contents of the cell to v . The update interaction is modeled by the family:

$$\begin{aligned} O[\text{update}]_X &: T_{\text{OW}}X \rightarrow (T_{\text{OW}}X)^{\mathbb{V}} \\ O[\text{update}]_A &: \langle \delta, x \rangle \mapsto \begin{cases} \lambda v_0. \langle \iota_2 v_0, x \rangle & \delta = \iota_1 \star \\ \lambda v_0. \langle \iota_2 v, x \rangle & \delta = \iota_2 v \end{cases} \end{aligned}$$

Finally, in this effectful context, a pure language, i.e. a language without side-effects, is modeled by the *identity* monad $T_{\text{id}}X := X$.

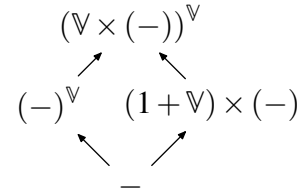
In this setting, Wadler, Tolmach, and Benton made the following observation. In a language consisting of a single global memory cell, there are four possible effect sets: \emptyset , $\{\text{read}\}$, $\{\text{write}\}$, and $\{\text{read}, \text{write}\}$. These sets are ordered by inclusion to form an *effect hierarchy*. For each effect set ε in the effect hierarchy there is a corresponding monad T_ε :



$$T_{\{\text{read}, \text{write}\}} := T_{\text{GS}} \quad T_{\{\text{read}\}} := T_{\text{Env}} \quad T_{\{\text{write}\}} := T_{\text{OW}} \quad T_{\emptyset} := T_{\text{id}}$$

Corresponding to each effect set inclusion, there is an obvious monad morphism. For example, the monad morphism from T_{\emptyset} to $T_{\{\text{update}\}}$ is given by $v \mapsto \langle \iota_1 \star, v \rangle$, which

lifts a pure computation returning v to a computation that does not change the state and returns v . As another example, the monad morphism from $T_{\{\text{lookup}\}}$ to $T_{\{\text{lookup}, \text{update}\}}$ is given by $k \mapsto \lambda v. \langle v, kv \rangle$, lifting a read-only computation k to a computation that inspects the current state, but does not change it. Thus the effect set hierarchy has a corresponding hierarchy of monads and monad morphisms. Consequently, computations only involving the interaction in each effect set ε can be interpreted within the monad T_ε , or lifted to monads above it in the hierarchy.



Wadler’s contribution [Wad98, WT03] is to identify the type-and-effect judgements $M : A! \varepsilon$ in a Gifford-style type-and-effect system for a language with multiple regions with monadic type judgements $M : T_\varepsilon A$ in a multi-monadic language. He then considers the two languages as variants of Moggi’s computational lambda-calculus and the computational meta-language [Mog89]. His technical contribution shows that Moggi’s translation from the former to the latter preserves the types, effects, type-and-effect derivations and inference, and operational semantics of the Gifford-style language. Wadler concludes by noting the monadic hierarchy *sans* the monad morphisms, and conjectures the existence of these morphisms. As Wadler only considers a language whose effects consist of memory allocation and access, he poses the question regarding the existence of a general theory of type-and-effect systems.

Tolmach’s contribution [Tol98] consists of a type-and-effect system for a recursive language with exceptions and state. His effect hierarchy consists of a linearly-ordered partial order of abstract effects (and not effect sets):

$$ID \leq LIFT \leq EXN \leq ST$$

which correspond to the effect sets

$$\emptyset \subseteq \{\text{diverge}\} \subseteq \{\text{diverge}, \text{exception}\} \subseteq \{\text{diverge}, \text{exception}, \text{read}, \text{write}\}$$

where ‘exception’ means the program may throw an exception, and ‘diverge’ means the program may diverge in a loop. Tolmach uses this effect hierarchy to validate some effect-dependent optimisations using his denotational equivalence. Tolmach outlines how his approach generalises to the complete lattice of subsets, or, more generally, to any lattice. However, he reports on the following problem:

“The lack of a generic mechanism for combining monads is rather unfortunate, since it turns the proofs of many transformation laws into lengthy case analyses.”

We divide this problem into two causes. The first is Wadler’s question: it is unclear how to obtain the hierarchy of monads. The second caveat is that the Tolmach-style models require specifying overwhelming amounts of data. In the small global state example we gave earlier we had to deal with four monads, four effect interactions and four monad morphisms. In general, we expect a number of monads, effect interactions, and monad morphisms exponential in the number of effects. Paired, these two caveats greatly restrict our ability to present such models for realistic cases, as we will require lengthy definitions and case-by-case analyses.

Benton et al.’s work in this area began with an optimising compiler from SML to JAVA bytecodes that uses a monadic intermediate language for effect-dependent optimisations [BKR98]. In a continued line of work, Benton et al. [BK99, BKHB06, BB07, BKBH07, BKBH09] validate the optimisations employed by this compiler using Tolmach-style models, independently from Tolmach. The key advantage of the Benton et al. models is their *uniformity*. For every effect set ϵ , they define a monad T_ϵ as a function of ϵ . This uniform definition saves much repetitiveness and tediousness in the account for the language and optimisations. For example, a single argument shows that T_ϵ is a monad, instead of a different argument for each ϵ . However, Wadler’s caveat still applies to Benton et al.’s approach: each change to the overall set of effects requires creating a new effect system with new semantics and new proofs.

Our goal is to address the issues encountered by Wadler, Tolmach, and Benton et al. by developing a *general* theory of type-and-effect systems, accounting for all aspects required by effect-dependent transformations: syntax, type system, effect inference, models, and optimisations and their validation. Moreover, we strive for an *applicable* theory that leads to engineering methodologies for optimising compilers. We thus formulate our thesis:

There exists a general applicable theory of Gifford-style type-and-effect systems.

1.3 An algebraic solution

Our work relies on Plotkin and Power’s algebraic theory of computational effects [PP03, PP02]. Plotkin and Power complement Moggi’s monadic account of computational ef-

fects [Mog91] by revisiting the well-established connection between universal algebra and monads [HP07]. Briefly, each equational theory gives rise to a monad, and most monads used in denotational semantics arise from such theories, with the notable exception of the continuation monad.

For example, the global state monad $(\mathbb{V} \times (-))^{\mathbb{V}}$ for a single memory cell storing values from a finite set $\mathbb{V} = \{v_1, \dots, v_n\}$, $n \geq 2$ arises from the following equational theory [PP02, Mel10]:

$$\text{update}_v(\text{update}_u(\mathbf{x})) = \text{update}_u(\mathbf{x}) \quad (1.1)$$

$$\text{update}_{v_i}(\text{lookup}(\mathbf{x}_1, \dots, \mathbf{x}_n)) = \text{update}_{v_i}(\mathbf{x}_i) \quad (1.2)$$

$$\text{lookup}(\text{update}_{v_1}(\mathbf{x}), \dots, \text{update}_{v_n}(\mathbf{x})) = \mathbf{x} \quad (1.3)$$

These equations have an operational reading as terms for manipulating computation. For example, the left side of Equation (1.1) reads: first update the cell to v , then update it to u , and then proceed with the computation \mathbf{x} . Thus Equation (1.1) states that later updates override earlier updates. As another example, the left side of Equation (1.2) reads: first update the cell to v_i , then look-up the currently stored value, say v_j , and proceed with the computation \mathbf{x}_j . Thus, Equation (1.2) states that we look-up the most recently updated value. Similarly, Equation (1.3) states that we do not keep track of the interaction history if it has no bearing on the rest of the computation.

Plotkin and Power [PP03] define *algebraic operations* as families of functions that arise from syntactic terms such as update and lookup. For example, the look-up and update interactions $O \llbracket \text{lookup} \rrbracket$, $O \llbracket \text{update} \rrbracket$ defined in the previous section are such algebraic operations for the equational theory for global state. The following observation is the fundamental idea underlying this thesis:

The annotation effects inside the effect-sets of Gifford-style type-and-effect systems denote Plotkin and Power's algebraic operations.

Thus, the annotation effects ‘read’ and ‘write’ we used above denote effect operations $O \llbracket \text{lookup} \rrbracket$ and $O \llbracket \text{update} \rrbracket$. Therefore, in the remainder of the thesis we use the same name for both annotations and operations.

This observation naturally leads to an algebraic theory of type-and-effect systems.

1. We begin with an equational theory as semantics to an effectful programming language.
2. The collection of syntactic constructs for algebraic effect operations in our language determines the effect annotations.
3. The effect annotations and their correspondence with the syntactic constructs determine the syntax, type-and-effect system, and effect inference.
4. By restricting the equational theory to the effect annotations in each effect-set, we obtain a hierarchy of monads and monad morphisms compatible with the effect hierarchy. We call this construction the *conservative restriction construction*.
5. We now have both syntax and semantics for formulating and validating effect-dependent transformations.

Note the interplay between *structure* and *property*. The only structure we need to specify is the original, algebraic, semantics of the programming language. The effect system and the valid optimisations are then a property arising out of our structure, which we may choose to investigate or ignore. As a result, our account is deeply coupled with the semantics of the language in question.

1.4 Our advantage

We highlight four key properties of our approach:

rigorous. If annotation effects indeed denote algebraic effect operations, then even a relatively simple language involving ten different effects will require over a thousand effect sets. A non-rigorous treatment is therefore error-prone. Moreover, even if such non-rigorous treatment is subject to thorough testing, it is impossible to establish the completeness of the validity decision procedures. Thus a non-rigorous design of effect-dependent transformations cannot be complete apart from the most trivial cases.

denotational. Monads provide a mechanism to abstract over effects that is not available with axiomatic or operational approaches. Moreover, validating the soundness of high-order program equalities using operational methods is inherently

difficult due to contextual equivalence. In contrast, the axiomatic approach is inappropriate for establishing the completeness of validity decision procedures.

abstract. As we will see in Chapter 12, our abstract and algebraic treatment hides many irrelevant details and facilitates delicate algebraic manipulations of symbols. It is implausible that such subtle arguments can be applied rigorously and without error to large concrete programs.

general. Our generality allows us to present a systematic, rather than ad-hoc, account of the optimisations. For example, as we will see in Chapter 11, this systematic account suggested new effect-dependent optimisations not present in the semantics literature.

1.5 Thesis structure

In order to support our thesis, we establish results in two directions. First, we show that the semantics of effect systems are a property of the algebraic semantics. Then, we show how this semantics accounts for the spectrum of type-and-effect systems and related concepts.

To provide a general theory, we must support a wide selection of effects. Even simply adding recursion to the language forces us to work beyond sets and functions, for example, with an appropriate notion of domains and continuous functions. For this reason, we provide evidence that our account can be formulated in *category theory*, to encompass the various standard denotational semantics for effects. Such a formulation encounters two problems.

- First, not all the notions and tools from equational logic have been fully developed for the level of generality required by our categorical account [Plo06].
- Second, the abstract categorical language hinders the accessibility of our account.

To address the first problem, we keep the development of new mathematical tools to a minimum, and reuse existing accounts as much as possible. Thus, we present only our conservative restriction construction in its full categorical generality. The other results are formulated using sets and functions, but structured in a manner we believe may generalise once the required tools are developed.

To address accessibility, we spread the required background across the thesis and introduce it as needed. We also highlight the parts of the thesis which require a deeper knowledge of category theory.



The sections indicated with a “beware cats” sign assume deeper, but standard, knowledge of category theory. The assumed knowledge includes: (co)limits, adjunctions, (co)continuity, and monads, as covered by Mac Lane [ML98], and symmetric monoidal closed categories, and enrichment, as covered by Kelly [Kel82a].



The “cat-free” parts of this thesis assume only basic categorical notions, which are common knowledge in the semantics and functional programming communities. In particular, we assume familiarity with functors, monads, natural transformations, and monad morphisms. These parts of the thesis are designed to be understood if read sequentially, skipping over the sections marked with “beware cats”. Of course, while the statements are formulated in these more accessible terms, their proofs may rely on categorically involved accounts. The scope of these two modifiers extends to the end of the literary unit they are introduced in, such as Chapter, Section, or Proof.

The second supporting evidence we present for our thesis is an account of the various concepts involved in type-and-effect systems and their optimisations. Due to time and space limitations, we focus on the core concepts required to study effect-dependent optimisations, namely an annotated source language and its models, and the optimisations and their validity. For example, we do *not* study the effect analysis phase that annotates a given source program with the effect sets.

The thesis is structured as follows:

Part I develops the general semantic foundations of type-and-effect systems.

Chapter 2 reviews Plotkin and Power’s notion of algebraic operations.

Chapter 3 defines hierarchical categorical models for type-and-effect systems.

Chapter 4 refines these models to account for languages with recursion.

Chapter 5 reviews the background on locally presentable categories we require.

Chapter 6 reviews Power’s enrichment of equational logic via enriched Lawvere theories.

Chapter 7 specialises the categorical models to models arising from Lawvere theories, and presents our categorical conservative restriction construction.

Chapter 8 presents the conservative restriction construction in terms of equational logic, sets, and functions.

Chapter 9 presents logical relations models for relating the semantics of multiple set-theoretic models.

Part II develops the general theory of type-and-effect systems and their effect-dependent optimisations based on the semantic foundations.

Chapter 10 presents a general type-and-effect intermediate language and its denotational semantics.

Chapter 11 develops the notion of effect-dependent optimisations, its validity conditions, and establishes the soundness and completeness of the conservative restriction construction.

Chapter 12 studies how the validity of optimisations in combined collections of effects follow from their validity in each component collection.

Chapter 13 demonstrates a use case for analysing an imperative, higher-order language.

Chapter 14 concludes and discusses future work.

Appendix A summarises our notational conventions for cross-references.

Part I

Semantic foundations

Chapter 2

Algebraic operations

I'm your operator

—Pink



The key contribution of Plotkin and Power's theory of effects is to incorporate into Moggi's monadic account the language constructs causing the effects, in the form of *algebraic operations*, or, equivalently, in the form of *generic effects*. This chapter defines the semantics of a call-by-push-value (CBPV) language with the effects incorporated directly into the model. The key difference between CBPV and traditional monadic semantics is the use of adjunctions instead of monads. Therefore, the main technical contribution of this chapter is to generalise Plotkin and Power's account of algebraic operations and generic effects [PP01, PP03] from the monadic setting to the adjunctive setting. In all other respects, this chapter contains only background material.

In Section 2.1 we begin with categorical preliminaries concerning resolutions of monads, lifting, and exponentials. Then, in Section 2.2, we define our generalisation of algebraic operations to an arbitrary resolution of a monad. In Section 2.3 we generalise Plotkin and Power's equivalence of algebraic operations with generic effects. This equivalence allows us to further analyse algebraic operations. Finally, in Section 2.4, we define our CBPV models of interest, and incorporate algebraic operations into the meta-theory.

2.1 Categorical preliminaries

Let \mathcal{V} be a category. Recall that a *resolution* of a monad T is an adjunction $F \dashv U : \mathcal{C} \rightarrow \mathcal{V}$ such that $T = U \circ F$ and the monadic unit η coincides with the adjunctive

unit. Recall also that every monad has two canonical resolutions, the *Eilenberg-Moore*, and *Kleisli* resolutions. The *Eilenberg-Moore category* for T , \mathcal{V}^T , consisting of algebras¹ $\underline{B} = \langle |\underline{B}|, \underline{B}[-] : T|\underline{B}| \rightarrow |\underline{B}| \rangle$ for the monad T , and T -algebra homomorphisms $h : \underline{B} \rightarrow \underline{B}'$. The *Eilenberg-Moore resolution* of T , is the resolution $F \dashv U : \mathcal{V}^T \rightarrow \mathcal{V}$ in which the left adjoint maps every object A to its *free algebra* $\langle TA, \mu \rangle$. The *Kleisli category* for T , \mathcal{V}_T consists of the \mathcal{V} -objects and *Kleisli maps* $f : A \rightarrow TB$. The *Kleisli resolution* of T is the resolution $F \dashv U : \mathcal{V}_T \rightarrow \mathcal{V}$ in which the left adjoint maps every object to itself.

Let $F \dashv U$ be any resolution of a strong monad T over a cartesian category \mathcal{V} , let $\theta : FU \rightarrow \text{id}$ be the counit for the resolution, and $f : \Gamma \times X \rightarrow U\underline{B}$ any \mathcal{V} -morphism. The *lifting* of f is the morphism $f^\dagger : \Gamma \times TX \rightarrow U\underline{B}$ given by the composition

$$\Gamma \times TX \xrightarrow{\text{str}} T(\Gamma \times X) \xrightarrow{Tf} TUB \xrightarrow{U\theta} U\underline{B}$$

Let A, B be objects in a cartesian category \mathcal{V} . Recall that an *exponential* of B by A consists of a pair $\langle B^A, \text{eval} \rangle$, where B^A is a \mathcal{V} -object, and $\text{eval} : B^A \times A \rightarrow B$ is a \mathcal{V} -morphism, such that, for any other arrow $f : C \times A \rightarrow B$, there exists a unique arrow $\lambda A.f : C \rightarrow B^A$ for which

$$\begin{array}{ccc} C & & C \times A \\ \lambda A.f \downarrow \text{dashed} & & \downarrow \text{solid} \\ B^A & & B^A \times A \end{array} \quad \begin{array}{ccc} & & C \times A \\ & & \downarrow \text{solid} \\ & & B^A \times A \end{array} \quad \begin{array}{ccc} & & \searrow f \\ & & = \\ & & \downarrow \text{solid} \\ & & B \end{array} \quad \begin{array}{ccc} & & \xrightarrow{\text{eval}} \\ & & B \end{array}$$

Thus exponentials are universal arrows from $- \times A$ to B [ML98, Section III.1]. Some authors use the notation $\text{curry}(f)$ for $\lambda A.f$. Given $f : B \rightarrow C$ with B, C exponentiable by A , we can define a map $f^A : B^A \rightarrow C^A$ by

$$f^A := \lambda A. \left(B^A \times A \xrightarrow{\text{eval}} B \xrightarrow{f} C \right)$$

The notion of a cartesian closed category corresponds exactly to a cartesian category with all exponentials and finite products, where the exponentials and the products are specified.

Let $F \dashv U$ be any resolution of a strong monad T over a cartesian category \mathcal{V} . If A is a \mathcal{V} -object for which all exponentials of the form $(U\underline{B})^A$ exist, we define the arrow $f^{\ddagger A} : \Gamma \times (TX)^A \rightarrow (U\underline{B})^A$ as

$$\lambda A. \left(\Gamma \times (TX)^A \times A \xrightarrow{\Gamma \times \text{eval}} \Gamma \times TX \xrightarrow{f^\dagger} U\underline{B} \right)$$

¹We will justify our choice of notation for the algebra map $\underline{B}[-]$ in Chapter 8.

Example 2-1. Let \mathbb{V} be a set with at least two elements denoting storable values. The global state monad over **Set** for storing \mathbb{V} values is given by:

$$\begin{aligned} TX &:= (\mathbb{V} \times X)^{\mathbb{V}} & \eta : x & \mapsto \lambda v. \langle v, x \rangle \\ Tf &: (\mathbb{V} \times X)^{\mathbb{V}} \rightarrow (\mathbb{V} \times Y)^{\mathbb{V}} & \mu : \lambda v. \langle u_v, k_v \rangle & \mapsto \lambda v. k_v(u_v) \\ Tf &: \lambda v. \langle u_v, x_v \rangle \mapsto \lambda v. \langle u_v, f(x_v) \rangle & \text{str}_{X,Y} : \langle x, \lambda v. \langle u_v, y_v \rangle \rangle & \mapsto \lambda v. \langle u_v, \langle x, y_v \rangle \rangle \end{aligned}$$

An algebra for this monad is a pair $\underline{Y} = \langle |\underline{Y}|, \underline{Y}[-] \rangle$, satisfying:

$$\underline{Y}[\lambda v. \langle v, y \rangle] = y \quad \underline{Y}[\lambda v. \langle u_v, \underline{Y}[\lambda v'. \langle w_{v,v'}, y_{v,v'} \rangle] \rangle] = \underline{Y}[\lambda v. \langle w_{v,u_v}, y_{v,u_v} \rangle]$$

Given any $f : \Gamma \times X \rightarrow |\underline{Y}|$, and a set Z direct calculations show that

$$\begin{aligned} f^\dagger : \langle \gamma, \lambda v. \langle u_v, x_v \rangle \rangle & \mapsto \underline{Y}[\lambda v. \langle u_v, f(\gamma, x_v) \rangle] \\ f^{\ddagger Z} : \langle \gamma, \lambda z. \lambda v. \langle u_{z,v}, x_{z,v} \rangle \rangle & \mapsto \lambda z. \underline{Y}[\lambda v. \langle u_{z,v}, f(\gamma, x_{z,v}) \rangle] \end{aligned} \quad \square$$

2.2 Algebraic operations

Without further ado, we generalise Plotkin and Power's definition of algebraic operations [PP01, PP03].

Definition 2.1 (algebraic operations). *Let $F \dashv U : \mathcal{C} \rightarrow \mathcal{V}$ be a resolution of a strong monad $T = UF$, and A, P be \mathcal{V} -objects such that all exponentials of objects UB by A, P exist. An algebraic operation of type $A \langle P \rangle$ for $F \dashv U$ is an $\text{Ob}(\mathcal{C})$ -indexed family of morphisms $\text{op} : (U-)^A \rightarrow (U-)^P$ satisfying, for all $f : \Gamma \times X \rightarrow UB$:*

$$\begin{array}{ccc} \Gamma \times (TX)^A & \xrightarrow{f^{\ddagger A}} & (UB)^A \\ \Gamma \times \text{op}_{FX} \downarrow & = & \downarrow \text{op}_B \\ \Gamma \times (TX)^P & \xrightarrow{f^{\ddagger P}} & (UB)^P \end{array}$$

We call A the arity of op , and P the parameter type of op .

This notion of algebraic operations is relative to any resolution of a strong monad. In contrast, Plotkin and Power define algebraic operations for the Kleisli and Eilenberg-Moore resolutions only. Thus, this definition is a minor generalisation of their notion. As we work with CBPV, we need non-free algebras, so the Kleisli resolution is not enough for our purposes. For working with *effect handlers*, Pretnar [Pre09] introduces a special case of this definition for a particular syntactic resolution. As we want

further work to account for handlers, we do not wish to tie algebraic operations too closely with the Eilenberg-Moore resolution. Therefore, we formulate our definition in this level of generality, and later instantiate it to the Eilenberg-Moore resolution.

Example 2-2. We present algebraic operations for global memory cell monad from Example 2-1. The look-up operation is given by:

$$\begin{aligned} \text{lookup}_{\underline{Y}} &: |\underline{Y}|^{\vee} \rightarrow |\underline{Y}|^{\mathbb{1}} \\ \text{lookup}_{\underline{Y}} &: \lambda v. y_v \mapsto \lambda \star. \underline{Y} \llbracket \lambda v. \langle v, y_v \rangle \rrbracket \end{aligned}$$

Note that if \underline{X} is a *free* algebra, FX , then lookup is given by:

$$\begin{aligned} \text{lookup}_{FX} &: \lambda v_1. \lambda v_2. \langle u_{v_1, v_2}, x_{v_1, v_2} \rangle \mapsto \lambda \star. TX \llbracket \lambda \tilde{v}. \langle \tilde{v}, \lambda v_2. \langle u_{\tilde{v}, v_2}, x_{\tilde{v}, v_2} \rangle \rangle \rrbracket \\ &= \lambda \star. \mu(\lambda \tilde{v}. \langle \tilde{v}, \lambda v_2. \langle u_{\tilde{v}, v_2}, x_{\tilde{v}, v_2} \rangle \rangle) \\ &= \lambda \star. \lambda \tilde{v}. \langle u_{\tilde{v}, \tilde{v}}, x_{\tilde{v}, \tilde{v}} \rangle \end{aligned}$$

Therefore, the sub-family of these maps indexed by the free algebras FX is the family

$$\begin{aligned} \text{lookup}_{FX} &: (TX)^{\vee} \rightarrow (TX)^{\mathbb{1}} \\ \text{lookup}_{FX} &: \lambda v_1. (\lambda v_2. \langle u_{v_1, v_2}, x_{v_1, v_2} \rangle) \mapsto \lambda v. \langle u_{v, v}, x_{v, v} \rangle \end{aligned}$$

given in the introduction.

The family lookup forms an algebraic operation of type $\vee \langle \mathbb{1} \rangle$ for the Eilenberg-Moore resolution of the global state monad. To see that, take any $f : \Gamma \times X \rightarrow |\underline{Y}|$, and chase an arbitrary element of $\Gamma \times (TX)^{\vee}$ around the following diagram, using the algebra multiplication law.

$$\begin{array}{ccc} & \xrightarrow{f^{\ddagger \vee}} & \\ \langle \gamma, \lambda v_1. \lambda v_2. \langle u_{v_1, v_2}, x_{v_1, v_2} \rangle \rangle & & \lambda v_1. \underline{Y} \llbracket \lambda v_2. \langle u_{v_1, v_2}, f(\gamma, x_{v_1, v_2}) \rangle \rrbracket \\ \downarrow \Gamma \times \text{lookup}_{FX} & & \downarrow \text{lookup}_{\underline{Y}} \\ \langle \gamma, \lambda \star. \langle u_{\tilde{v}, \tilde{v}}, x_{\tilde{v}, \tilde{v}} \rangle \rangle & & \lambda \star. \underline{Y} \llbracket \lambda \tilde{v}. \langle \tilde{v}, \underline{Y} \llbracket \lambda v_2. \langle u_{\tilde{v}, v_2}, f(\gamma, x_{\tilde{v}, v_2}) \rangle \rangle \rrbracket \rrbracket \\ & & \parallel \leftarrow \underline{Y} \text{ is an algebra} \\ & & \lambda \star. \underline{Y} \llbracket \lambda \tilde{v}. \langle u_{\tilde{v}, \tilde{v}}, x_{\tilde{v}, \tilde{v}} \rangle \rrbracket \\ & \xleftarrow{f^{\ddagger \mathbb{1}}} & \end{array}$$

Similarly, the update interaction is given by:

$$\begin{aligned} \text{update}_{\underline{Y}} &: |\underline{Y}|^{\mathbb{1}} \rightarrow |\underline{Y}|^{\mathbb{V}} \\ \text{update}_{\underline{Y}} &: \lambda \star . y \mapsto \lambda v_0 . \underline{Y} \llbracket \lambda v . \langle v_0, y \rangle \rrbracket \end{aligned}$$

Again, note that the sub-family of these maps indexed by the free algebras coincides with the family

$$\begin{aligned} \text{update}_{FX} &: (TX)^{\mathbb{1}} \rightarrow (TX)^{\mathbb{V}} \\ \text{update}_{FX} &: \lambda \star . k \mapsto \lambda v_0 . \lambda v . k(v_0) \end{aligned}$$

given in the introduction. An identical argument shows it is an algebraic operation of type $\mathbb{1} \langle \mathbb{V} \rangle$ for the Eilenberg-Moore resolution of the global state monad. \square

Hyland et al. [HPP06, Section 6] describe means to transform every algebraic operation op for T to an algebraic operation op' for T' which they call *operation transformers*. Hyland et al. showed that operation transformers are in bijection with strong monad morphisms. We therefore take strong monad morphisms as our primary means of transforming operations:

Definition 2.2. *Let $m : T \rightarrow T'$ be a strong monad morphism over a cartesian category \mathcal{V} , and op , op' be algebraic operations of type $A \langle P \rangle$ for any two resolutions $F \dashv U$, $F' \dashv U'$ of T , T' , respectively. We say that m maps op to op' if, for all $X \in \text{Ob}(\mathcal{V})$:*

$$\begin{array}{ccc} (TX)^A & \xrightarrow{\text{op}_{FX}} & (TX)^P \\ m^A \downarrow & = & \downarrow m^P \\ (T'X)^A & \xrightarrow{\text{op}'_{F'X}} & (T'X)^P \end{array}$$

Example 2-3. The environment monad is given by:

$$\begin{aligned} T_{\text{Env}(\mathbb{V})}X &:= X^{\mathbb{V}} & \eta : x & \mapsto \lambda v . x \\ T_{\text{Env}(\mathbb{V})}f &: X^{\mathbb{V}} \rightarrow Y^{\mathbb{V}} & \mu : \lambda v . k_v & \mapsto \lambda v . k_v(u_v) \\ T_{\text{Env}(\mathbb{V})}f &: \lambda v . x_v \mapsto \lambda v . f(x_v) & \text{str}_{X,Y} : \langle x, \lambda v . y_v \rangle & \mapsto \lambda v . \langle x, y_v \rangle \end{aligned}$$

This monad admits an algebraic operation $\text{lookup}^{\text{Env}(\mathbb{V})} : \mathbb{V}$, given by:

$$\begin{aligned} \text{lookup}_{\underline{Y}} &: |\underline{Y}|^{\mathbb{V}} \rightarrow |\underline{Y}|^{\mathbb{1}} \\ \text{lookup}_{\underline{Y}} &: \lambda v . y_v \mapsto \lambda \star . \underline{Y} \llbracket \lambda v . y_v \rrbracket \end{aligned}$$

When instantiated to free algebras, it subsumes the lookup operation from the introduction:

$$\begin{aligned} \text{lookup}_{F_{\text{Env}(\mathbb{V})}X} &: (T_{\text{Env}(\mathbb{V})}X)^{\mathbb{V}} \rightarrow (T_{\text{Env}(\mathbb{V})}X)^{\mathbb{1}} \\ \text{lookup}_{F_{\text{Env}(\mathbb{V})}X} &: \lambda v_1 . (\lambda v_2 . x_{v_1, v_2}) \mapsto \lambda v . x_{v, v} \end{aligned}$$

Let $T_{\text{GS}(\mathbb{V})}$ and $\text{lookup}^{\text{GS}(\mathbb{V})}$ be the global state monad and its associated look-up operation from Examples 2-1 and 2-2. Then the following map is a monad morphism from $T_{\text{Env}(\mathbb{V})}$ to $T_{\text{GS}(\mathbb{V})}$:

$$\begin{aligned} m &: T_{\text{Env}(\mathbb{V})}Y \rightarrow T_{\text{GS}(\mathbb{V})}Y \\ m &: \lambda v. x_v \quad \mapsto \lambda v. \langle v, x_v \rangle \end{aligned}$$

This monad morphism maps $\text{lookup}^{\text{Env}(\mathbb{V})}$ to $\text{lookup}^{\text{GS}(\mathbb{V})}$:

$$\begin{array}{ccc} \lambda v_1. \lambda v_2. x_{v_1, v_2} & \xrightarrow{\text{lookup}^{\text{Env}(\mathbb{V})}} & \lambda \star. \lambda v. x_{v, v} \\ m^{\mathbb{V}} \downarrow & & \downarrow m^{\mathbb{1}} \\ \lambda v_1. \lambda v_2. \langle v_2, x_{v_1, v_2} \rangle & \xrightarrow{\text{lookup}^{\text{GS}(\mathbb{V})}} & \lambda \star. \lambda v. \langle v, x_{v, v} \rangle \end{array}$$

Similarly, the overwrite monad is given by:

$$\begin{aligned} T_{\text{OW}(\mathbb{V})}X &:= (\mathbb{1} + \mathbb{V}) \times X & \eta : x & \mapsto \lambda v. x \\ T_{\text{OW}(\mathbb{V})}f &: (\mathbb{1} + \mathbb{V}) \times X \rightarrow (\mathbb{1} + \mathbb{V}) \times Y & \mu : \langle \mathbf{t}_1 \star, \langle \delta, x \rangle \rangle & \mapsto \langle \delta, x \rangle \\ T_{\text{OW}(\mathbb{V})}f &: \langle \delta, x \rangle \mapsto \langle \delta, f(x_v) \rangle & \langle \delta, \langle \mathbf{t}_1 \star, x \rangle \rangle & \mapsto \langle \delta, x \rangle \\ \text{str}_{X, Y} &: \langle x, \langle \delta, y \rangle \rangle \mapsto \langle \delta, \langle x, y_v \rangle \rangle & \langle \mathbf{t}_2 v_1, \langle \mathbf{t}_2 v_2, x \rangle \rangle & \mapsto \langle \mathbf{t}_2 v_2, x \rangle \end{aligned}$$

This monad admits an algebraic operation $\text{update}^{\text{OW}(\mathbb{V})} : \mathbb{1} \langle \mathbb{V} \rangle$, given by:

$$\begin{aligned} \text{update}_{\underline{Y}} &: |\underline{Y}|^{\mathbb{1}} \rightarrow |\underline{Y}|^{\mathbb{V}} \\ \text{update}_{\underline{Y}} &: \langle \mathbf{t}_1 \star, y \rangle \mapsto \lambda v. \underline{Y} \llbracket \langle v, y \rangle \rrbracket \\ & \quad \langle \mathbf{t}_2 v', y \rangle \mapsto \lambda v. \underline{Y} \llbracket \langle v', y \rangle \rrbracket \end{aligned}$$

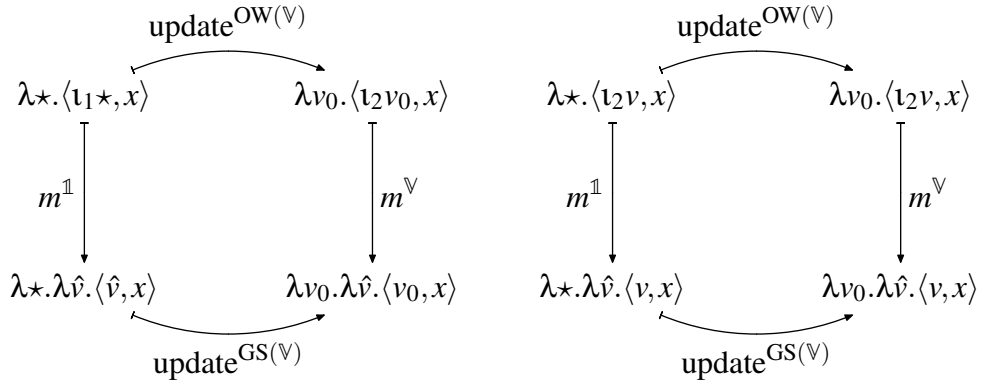
When instantiated to free algebras, it subsumes the update operation from the introduction:

$$\begin{aligned} \text{update}_{F_{\text{OW}(\mathbb{V})}X} &: (T_{\text{OW}(\mathbb{V})}X)^{\mathbb{1}} \rightarrow (T_{\text{OW}(\mathbb{V})}X)^{\mathbb{V}} \\ \text{update}_{F_{\text{OW}(\mathbb{V})}X} &: \lambda \star. \langle \delta, v \rangle \mapsto \begin{cases} \lambda v_0. \langle \mathbf{t}_2 v_0, x \rangle & \delta = \mathbf{t}_1 \star \\ \lambda v_0. \langle \mathbf{t}_2 v, x \rangle & \delta = \mathbf{t}_2 v \end{cases} \end{aligned}$$

Let $\text{update}^{\text{GS}(\mathbb{V})}$ be the update operation for the global state from Example 2-2. Then the following map is a monad morphism from $T_{\text{OW}(\mathbb{V})}$ to $T_{\text{GS}(\mathbb{V})}$:

$$\begin{aligned} m &: T_{\text{OW}(\mathbb{V})}A \rightarrow T_{\text{GS}(\mathbb{V})}A \\ m &: \langle \mathbf{t}_1 \star, a \rangle \mapsto \lambda v. \langle v, a \rangle \\ & \quad \langle \mathbf{t}_2 v_0, a \rangle \mapsto \lambda v. \langle v_0, a \rangle \end{aligned}$$

This monad morphism maps $\text{update}^{\text{OW}(\mathbb{V})}$ to $\text{update}^{\text{GS}(\mathbb{V})}$:



2.3 Generic effects

Plotkin and Power [PP03] noted there is another common way to model effects:

Definition 2.3 (generic effects). *Let T be a monad over \mathcal{V} , and A, P be \mathcal{V} -objects. A generic effect of type $A \langle P \rangle$ is a \mathcal{V} -morphism $\text{gen} : P \rightarrow TA$.*

Example 2-4. The global state monad has the following two generic effects:

$$\begin{array}{ll}
 \text{deref!} : \mathbb{V} & \text{set!} : \mathbb{1} \langle \mathbb{V} \rangle \\
 \text{deref!} : \mathbb{1} \rightarrow T_{\text{GS}(\mathbb{V})} \mathbb{V} & \text{set!} : \mathbb{V} \rightarrow T_{\text{GS}(\mathbb{V})} \mathbb{1} \\
 \text{deref!} : \star \mapsto \lambda v. \langle v, v \rangle & \text{set!} : v_0 \mapsto \lambda v. \langle v_0, \star \rangle
 \end{array}$$

Similarly, the environment and overwrite monads have analogous generic effects:

$$\begin{array}{ll}
 \text{deref!} : \mathbb{V} & \text{set!} : \mathbb{1} \langle \mathbb{V} \rangle \\
 \text{deref!} : \mathbb{1} \rightarrow T_{\text{Env}(\mathbb{V})} \mathbb{V} & \text{set!} : \mathbb{V} \rightarrow T_{\text{OW}(\mathbb{V})} \mathbb{1} \\
 \text{deref!} : \star \mapsto \lambda v. v & \text{set!} : v_0 \mapsto \langle t_2 v_0, \star \rangle
 \end{array}$$

Plotkin and Power [PP03] noted the following bijection between algebraic operations and generic effects. While they discussed it in the monadic setting, we generalise the connection to arbitrary resolutions.

Theorem 2.4. *Let $F \dashv U : \mathcal{C} \rightarrow \mathcal{V}$ be a resolution of a strong monad T , and A, P be \mathcal{V} -objects such that all exponentials $(U-)^A$, $(U-)^P$ exist. Algebraic operations $\text{op} : A \langle P \rangle$ and generic effects $\text{gen} : A \langle P \rangle$ are in bijection via*

$$\begin{aligned}
 \text{gen}_{\text{op}} &:= P \xrightarrow{\langle \lambda A. \eta \circ \pi_2, P \rangle} (TA)^A \times P \xrightarrow{\text{op}_{FA} \times P} (TA)^P \times P \xrightarrow{\text{eval}} TA \\
 \text{op}_{\underline{B}}^{\text{gen}} &:= \lambda P. \left((\underline{UB})^A \times P \xrightarrow{(\underline{UB})^A \times \text{gen}} (\underline{UB})^A \times TA \xrightarrow{\text{eval}^\dagger} \underline{UB} \right)
 \end{aligned}$$

Example 2-5. In Examples 2-3 and 2-4 we saw, respectively, an algebraic operation and a generic effect for the environment monad $T_{\text{Env}(\mathbb{V})}A = A^{\mathbb{V}}$:

$$\begin{aligned} \text{lookup}_{\underline{Y}} &: \lambda v. b_v \mapsto \lambda \star. \underline{Y} \llbracket \lambda v. b_v \rrbracket \\ \text{deref!} &: \star p \mapsto \lambda v. v \end{aligned}$$

Calculating the corresponding generic effect and algebraic operations yields:

$$\begin{aligned} \text{gen}_{\text{lookup}} &: \star \xrightarrow{\langle \lambda A. \eta \circ \pi_2, P \rangle} \langle \eta, \star \rangle \xrightarrow{\text{lookup}_{F_{\text{Env}(\mathbb{V})}} \times P} \langle \lambda \star. \mu \circ \eta_F, \star \rangle \xrightarrow{\text{eval}} \lambda v. v \\ \text{op}_{\underline{Y}}^{\text{deref!}} &: \lambda v. y_v \mapsto \lambda \star. \text{eval}^\dagger(\lambda v. y_v, \lambda \hat{v}. \hat{v}) = \lambda \star. \underline{Y} \llbracket \lambda v. \text{eval}(\lambda \hat{v}. y_{\hat{v}}, v) \rrbracket = \lambda \star. \underline{Y} \llbracket \lambda v. y_v \rrbracket \end{aligned}$$

Therefore, $\text{gen}_{\text{lookup}} = \text{deref!}$ and $\text{lookup} = \text{op}^{\text{deref!}}$.

Similar calculations show, for the overwrite monad $T_{\text{OW}(\mathbb{V})}A = (\mathbb{1} + \mathbb{V}) \times A$, we have $\text{gen}_{\text{update}} = \text{set!}$ and $\text{update} = \text{op}^{\text{set!}}$. \square

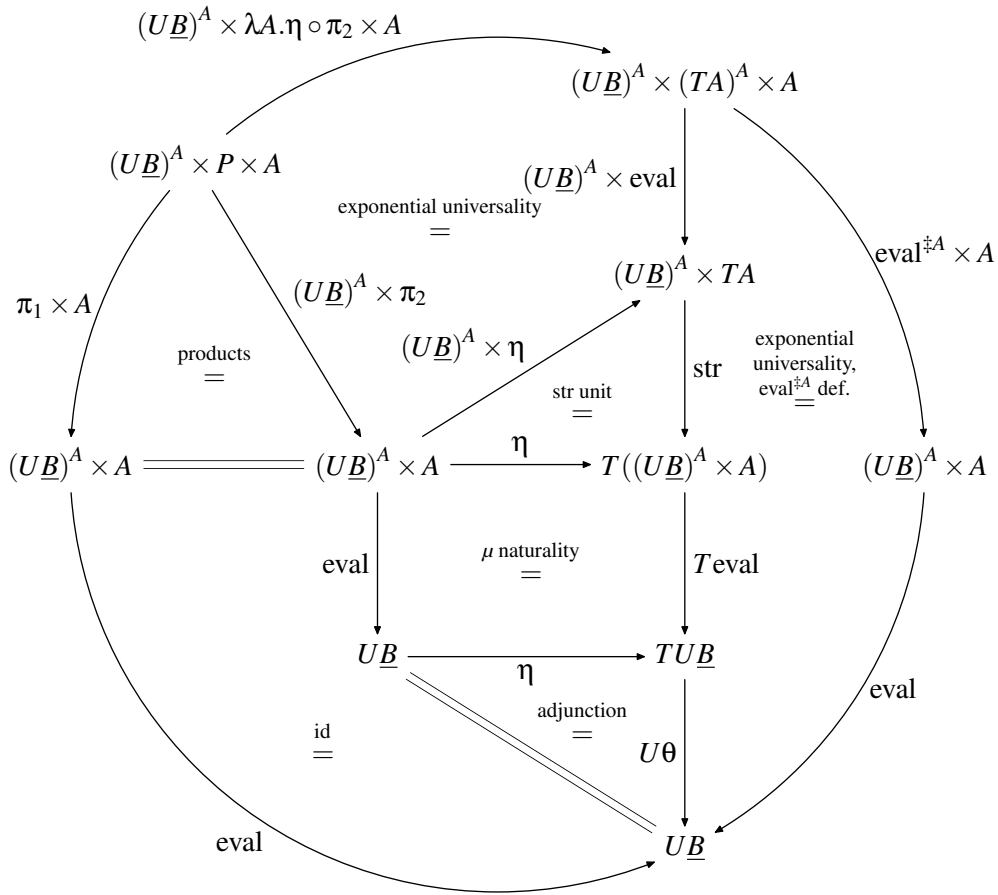
Proof

The following calculations show that: (1) op^{gen} is an algebraic operation, i.e., op^- is well-defined; (2) $\text{op}^{\text{gen op}} = \text{op}$; and (3) $\text{gen}_{\text{op}^{\text{gen}}} = \text{gen}$. These calculations become clearer through string diagrams (see, for example, Baez and Stay [BS11]). However, we keep to commuting diagrams to avoid the overhead imposed by additional notation.

1. For all $f : \Gamma \times X \rightarrow U\mathbb{B}$, we have

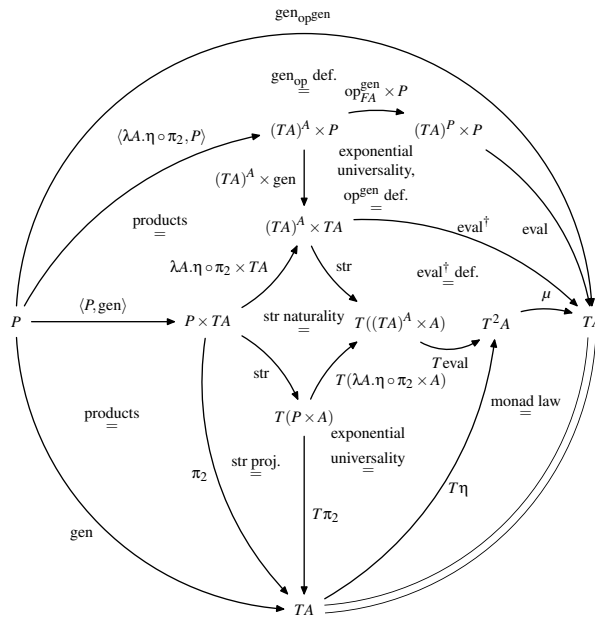
The diagram illustrates the commutativity of various operations and naturalities. Key components include:

- Top Level:** $\Gamma \times (TX)^A \times P$ (left) and $(U\mathbb{B})^A \times P$ (right), connected by $f^{\sharp A} \times P$.
- Middle Level:** $\Gamma \times (TX)^A \times TA$ and $(U\mathbb{B})^A \times TA$, connected by $f^{\sharp A} \times TA$.
- Bottom Level:** $\Gamma \times TX$ and $TU\mathbb{B}$, connected by Tf .
- Operations:** \times (bifunctionality), T (functoriality), U (forgetful functor), eval , str (string), str associativity , str naturality , $\text{str multiplication}$, $\text{exponential universality}$, op^{gen} def., μ naturality, θ naturality.
- Resolutions:** resol-ution (circled) for μ and θ .



Therefore, by universality of exponentials, $\text{op}^{\text{gen}_{\text{op}}} = \text{op}$.

3. Finally, the following diagram shows that $\text{gen}_{\text{op}^{\text{gen}}} = \text{gen}$:



A close inspection of Theorem 2.4's proof shows that every algebraic operation $\text{op} : A \langle P \rangle$ is uniquely determined by the component op_{FA} :

Corollary 2.5. *Let $F \dashv U$ be a resolution of a strong monad T , and A, P be objects such that all exponentials $(U-)^A$, $(U-)^P$ exist. For every morphism $g : (TA)^A \rightarrow (TA)^P$ satisfying*

$$\begin{array}{ccc} (TA)^A \times (TA)^A & \xrightarrow{\text{eval}^{\ddagger A}} & (TA)^A \\ (TA)^A \times g \downarrow & = & \downarrow g \\ (TA)^A \times (TA)^P & \xrightarrow{\text{eval}^{\ddagger P}} & (TA)^P \end{array}$$

there exists a unique algebraic operation $\text{op} : A \langle P \rangle$ such that $\text{op}_{FA} = g$.

Proof

For uniqueness, note that gen_{op} is completely determined by op_{FA} , which satisfies the required condition. Therefore, by Theorem 2.4, op is determined uniquely by g . For existence, given any such g , define a generic effect by

$$\text{gen} := P \xrightarrow{\langle \lambda A. \eta \circ \pi_2, P \rangle} (TA)^A \times P \xrightarrow{g \times P} (TA)^P \times P \xrightarrow{\text{eval}} TA$$

By Theorem 2.4, $\text{op} := \text{op}^{\text{gen}}$ is an algebraic operation. Part (2) of the proof remains valid if we replace op_{FA} by g , and gen_{op} by gen , showing that $\text{op}_{FA} = g$. ■

Thus, algebraic operations are independent of the particular choice of a resolution of the monad, as op_{FA} depends solely on the monadic structure.

Corollary 2.6. *Let $F_1 \dashv U_1$, $F_2 \dashv U_2$ be resolutions for a strong monad T , and A, P be objects such that all exponentials $(U_i-)^A$, $(U_i-)^P$ exist for $i = 1, 2$. Algebraic operations $\text{op}^1 : P \langle A \rangle$ for $F_1 \dashv U_1$ and algebraic operations $\text{op}^2 : P \langle A \rangle$ for $F_2 \dashv U_2$ are in bijection via the equation $\text{op}_{FA}^1 = \text{op}_{FA}^2$.*

Proof

The condition in Corollary 2.5 involves only the monadic structure, which coincides for the two resolutions. ■

Therefore, our definition for algebraic operations captures precisely Plotkin and Power's notion of algebraic operations [PP03], but is applicable to resolutions other than the Kleisli and Eilenberg-Moore resolutions. In the sequel we will therefore speak of algebraic operations for a monad without referring to a particular resolution of that monad.

Definition 2.7. Let $m : T \rightarrow \hat{T}$ be a strong monad morphism, and $\text{gen}, \text{gen}' : A \langle P \rangle$ be two generic effects for T, \hat{T} respectively. We say that m maps gen to gen' if

$$\begin{array}{ccc} & & TA \\ & \text{gen} \nearrow & \downarrow m \\ P & = & T'A \\ & \text{gen}' \searrow & \end{array}$$

Example 2-6. In Example 2-4 we encountered the lookup and update effects for the environment and global state monad:

$$\begin{array}{ll} \text{deref!}^{\text{GS}} : \star \mapsto \lambda v. \langle v, v \rangle & \text{set!}^{\text{GS}} : v_0 \mapsto \lambda v. \langle v_0, \star \rangle \\ \text{deref!}^{\text{Env}} : \star \mapsto \lambda v. v & \text{set!}^{\text{OW}} : v_0 \mapsto \langle \iota_2 v_0, \star \rangle \end{array}$$

Recall the two monad morphisms from Example 2-3:

$$\begin{array}{ll} m_{\text{Env}} : T_{\text{Env}(V)}A \rightarrow T_{\text{GS}(V)}A & m_{\text{OW}} : T_{\text{OW}(V)}A \rightarrow T_{\text{GS}(V)}A \\ m_{\text{Env}} : \lambda v. a_v \mapsto \lambda v. \langle v, a_v \rangle & m_{\text{OW}} : \langle \iota_1 \star, a \rangle \mapsto \lambda v. \langle v, a \rangle \\ & \langle \iota_2 v_0, a \rangle \mapsto \lambda v. \langle v_0, a \rangle \end{array}$$

Straightforward calculation shows that m_{Env} maps $\text{deref!}^{\text{Env}}$ to $\text{deref!}^{\text{GS}}$, and that m_{OW} maps set!^{OW} to set!^{GS} . \square

The bijection between generic effects and algebraic operations preserves the relation of being mapped by a monad morphism:

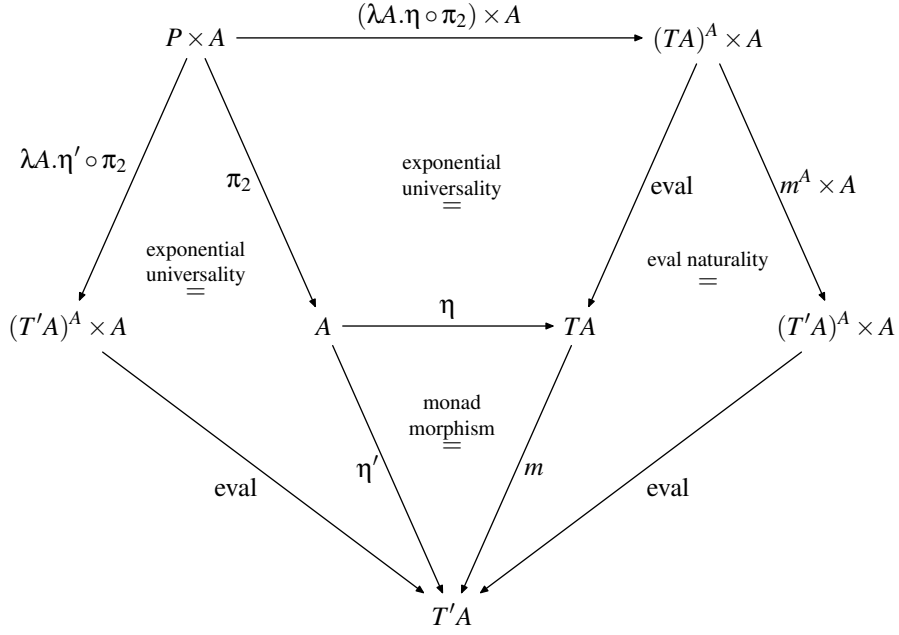
Theorem 2.8. Let $\text{op}, \text{op}' : A \langle P \rangle$ be algebraic operations for any resolutions $F \dashv U, F' \dashv U'$ of any strong monads T, T' , respectively, and $m : T \rightarrow T'$ a strong monad morphism. Then m maps op to op' if and only if m maps gen_{op} to $\text{gen}_{\text{op}'}$.

Proof

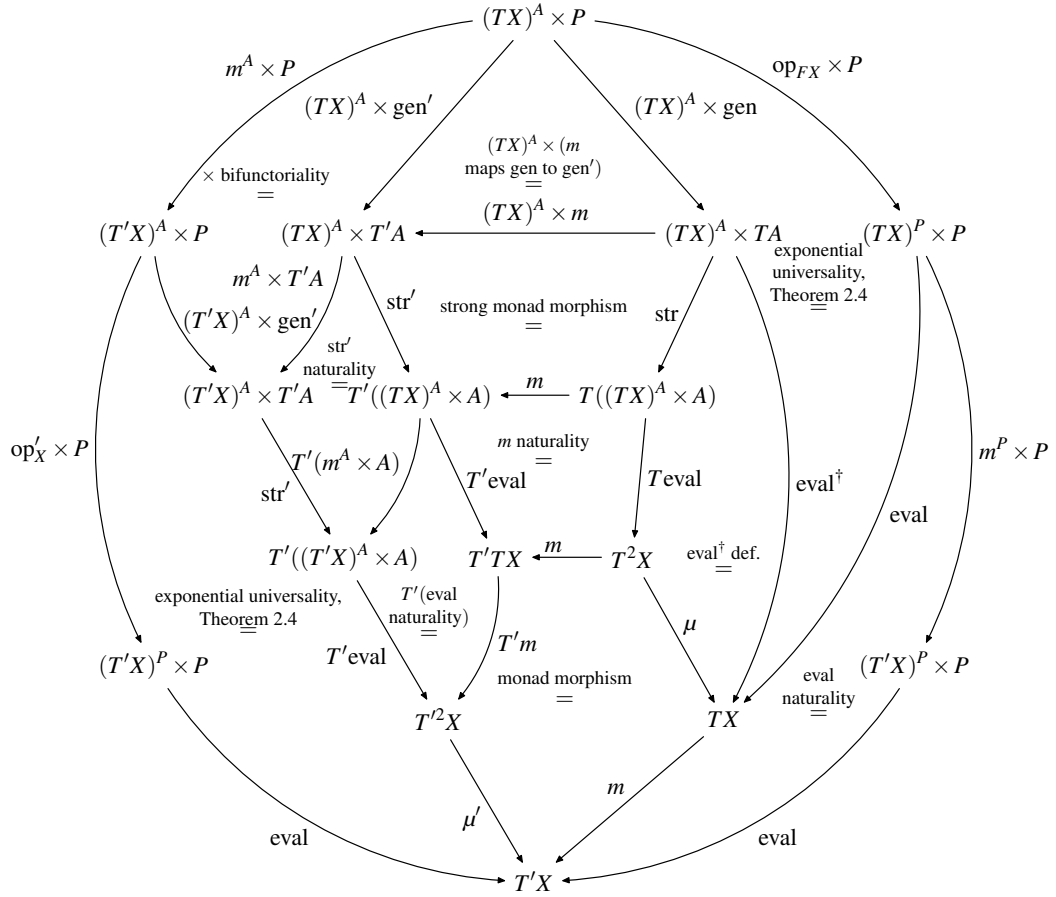
First, assume m maps op to op' . We have

$$\begin{array}{ccc} & P & \\ & \downarrow \delta & \\ & P \times P & \\ \text{gen}_{\text{op}} \nearrow & \begin{array}{c} \text{gen}_{\text{op}} \text{ def.} \\ \text{=} \\ (\lambda A. \eta \circ \pi_2) \times P \\ \text{=} \\ (\ast) \times P \\ \text{=} \\ \lambda A. \eta \circ \pi_2 \times P \end{array} & \searrow \text{gen}_{\text{op}'} \\ & (TA)^A \times P \xrightarrow{m^A \times P} (T'A)^A \times P & \\ \text{op} \times P \nearrow & \begin{array}{c} (m \text{ maps } \text{op} \text{ to } \text{op}') \times P \\ \text{=} \\ m^P \times P \end{array} & \searrow \text{op}' \times P \\ & (TA)^P \times P \xrightarrow{m^P \times P} (T'A)^P \times P & \\ \text{eval} \nearrow & \text{eval naturality} & \searrow \text{eval} \\ & \text{=} & \\ TA & \xrightarrow{m} & T'A \end{array}$$

where (*) follows, by the universal property of exponentials, from



Conversely, assume $\text{gen} := \text{gen}_{\text{op}}$ is mapped to $\text{gen}' := \text{gen}_{\text{op}'}$. We then have, for every object X :



Therefore, by the universal property of exponentials, we have:

$$\begin{array}{ccc}
(TX)^A & \xrightarrow{\text{op}_{FX}} & (TX)^P \\
m^A \downarrow & = & \downarrow m^P \\
(T'X)^A & \xrightarrow{\text{op}'_{F'X}} & (T'X)^P
\end{array}$$

and m maps op to op' . ■

If m maps gen to gen' , then gen is uniquely determined by m and gen : $\text{gen}' = m \circ \text{gen}$. Theorems 2.4 and 2.8 let us transfer this observation to algebraic operations:

Corollary 2.9. *Let $F \dashv U$, $F' \dashv U'$ be any two resolutions of any pair of strong monads T , T' , and $m : T \rightarrow T'$ a strong monad morphism. For every algebraic operation $\text{op} : A \langle P \rangle$ of $F \dashv U$, there exists a unique algebraic operation $\text{op}' : A \langle P \rangle$ of $F' \dashv U'$ such that m maps op to op' .*

Proof

If $\text{op}', \text{op}'' : A \langle P \rangle$ are two algebraic operations such that m maps op to both op' and op'' , then, by Theorem 2.8, m maps gen_{op} to both $\text{gen}_{\text{op}'}$ and $\text{gen}_{\text{op}''}$. Necessarily, $\text{gen}_{\text{op}'} = \text{gen}_{\text{op}''}$. By Theorem 2.4 we deduce that $\text{op}' = \text{op}''$. For existence, by fiat, m maps gen_{op} to $\text{gen}' := m \circ \text{gen}_{\text{op}}$. Theorem 2.8 implies m maps op to $\text{op}^{\text{gen}'}$. ■

Example 2-7. To demonstrate this corollary, recall we found a strong monad morphism $m : T_{\text{Env}(\mathbb{V})} \rightarrow T_{\text{GS}(\mathbb{V})}$ that maps the algebraic operation for look-up of $T_{\text{Env}(\mathbb{V})}$ to that of $T_{\text{GS}(\mathbb{V})}$ (Example 2-3), and the generic effect for look-up of $T_{\text{Env}(\mathbb{V})}$ to that of $T_{\text{GS}(\mathbb{V})}$ (Example 2-6). Finally, recall that the generic effect for look-up corresponds to the algebraic operation for lookup (Example 2-5). Therefore, by Corollary 2.9, the generic effect corresponding to the lookup algebraic operation for the global state monad is the lookup generic effect presented in Example 2-4. □

2.4 CBPV models

We now synthesise algebraic operations with CBPV models. In order to accommodate operations in our meta-theory, we parametrise our definition by a set Π whose elements we call *effect operation names* such as $\Pi := \{\text{lookup}, \text{update}\}$.

Definition 2.10. *Let Π be a set and \mathcal{V} a category. A type assignment for Π in \mathcal{V} is an assignment $\text{type}(\text{op}) = \langle A, P \rangle$ of a pair of \mathcal{V} -objects to each $\text{op} \in \Pi$. We call A the arity of op , and P the parameter type of op .*

We write $\text{op} : A \langle P \rangle$ when $\text{type}(\text{op}) = \langle A, P \rangle$ and the assignment is clear from the context. When the parameter type is the terminal object $\mathbb{1}$, we write $\text{op} : A$ instead of $\text{op} : A \langle \mathbb{1} \rangle$.

Example 2-8. Consider $\Pi := \{\text{lookup}, \text{update}\}$. Let \mathbb{V} be a set denoting storable values. A set-theoretic type assignment for the global memory \mathbb{V} -cell operations is:

$$\text{lookup} : \mathbb{V} \quad \text{update} : \mathbb{1} \langle \mathbb{V} \rangle \quad \square$$

Example 2-9. Consider $\Pi := \{\text{input}, \text{output}, \text{raise}\}$. Let Char be a set modeling the characters in an interactive terminal. For example, we may choose $\text{Char} := \{0, 1, 2, \dots, 127\}$ for modeling ASCII characters. Let Str be Char^* , the set of strings, i.e., finite sequences of characters. A set-theoretic type assignment for I/O and exceptions is:

$$\text{input} : \text{Char} \quad \text{output} : \mathbb{1} \langle \mathbb{V} \rangle \quad \text{raise} : 0 \langle \text{Str} \rangle \quad \square$$

We restrict our interest to the following notion of a CBPV model:

Definition 2.11. An (Eilenberg-Moore) CBPV model is a pair $\langle \mathcal{V}, T \rangle$ where:

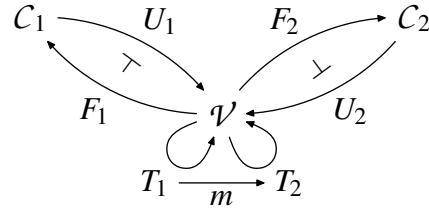
- \mathcal{V} is a distributive category;
- T is a strong monad over \mathcal{V} ; and
- \mathcal{V} has all exponentials $|B|^A$ of all T -algebra carriers $|B|$ by all objects A in \mathcal{V} .

General models of CBPV form a properly larger class of adjunctions. In our development, we are only going to use Eilenberg-Moore models of CBPV. Therefore, we restrict attention to those models only. This restriction simplifies the development considerably. We leave the treatment of arbitrary CBPV models as further work.

We also need a notion of morphisms between CBPV models. For our purposes, it suffices to consider *strong monad morphisms* $m : T \rightarrow T'$. There are other possible choices. For example, one choice of morphisms is as functors $U_m : \mathcal{V}^T \rightarrow \mathcal{V}^{T'}$ between the Eilenberg-Moore categories of the monads that factor the right adjoint $U' : \mathcal{V}^{T'} \rightarrow \mathcal{V}$ as $U \circ U_m$. Another choice of morphisms is to require that those functors U_m to have a left adjoint. In our cases of interest, in which algebraic semantics is possible, these choices coincide. Therefore, we choose strong monad morphisms as our morphisms between CBPV models. Future use of the theory will determine which notion is more appropriate for applications.

Definition 2.12. Let \mathcal{V} be a distributive category. The category $\text{CBPV}_{\mathcal{V}}$ has as **objects** CBPV models with $F \dashv U : \mathcal{C} \rightarrow \mathcal{V}$, and as **morphisms** $(F_1 \dashv U_1) \rightarrow (F_2 \dashv U_2)$ strong monad morphisms $m : T_1 \rightarrow T_2$.

The following diagram summarises the relevant data in a morphism between CBPV models:



The following definition ties all the previous concepts together:

Definition 2.13. Let Π be a set. A CBPV Π -model is a quadruple

$$\langle \mathcal{V}, \text{type}, T, O[-] \rangle$$

where:

- \mathcal{V} is a distributive category, called the value category;
- type is a type assignment for Π in \mathcal{V} ;
- $\langle \mathcal{V}, T \rangle$ is a CBPV model; and
- $O[-]$ assigns to every $\text{op} : A \langle P \rangle$ in Π an algebraic operation $O[\text{op}] : A \langle P \rangle$ for T .

Thus, effect operation names op denote algebraic operations $O[\text{op}]$.

Example 2-10. Let Π be a two element set $\{\text{lookup}, \text{update}\}$, and \mathbb{V} a set denoting storable values. We present a CBPV Π -model for a global memory \mathbb{V} -cell.

We take: \mathcal{V} to be **Set**; type to be the type assignment $\text{lookup} : \mathbb{V}$ and $\text{update} : \mathbb{1} \langle \mathbb{V} \rangle$; from Example 2-9; T to be the global state monad $T_{\text{GS}(\mathbb{V})}$; and $O[-]$ to interpret lookup and update as the look-up and update operations for $T_{\text{GS}(\mathbb{V})}$, as in Examples 2-1 and 2-2. □

To summarise, we reviewed Plotkin and Power's algebraic operations and generic effects in the adjunctive setting, and their preservation by monad morphisms, and incorporated them into the CBPV model structure.

Chapter 3

Models

I'm a model, you know what I mean.

—Right Said Fred



In this chapter we define our notion of models for Gifford-style effect systems. There is tension inherent in this definition resulting from the degree of structure these models include. On the one hand, we want to axiomatise the similarities in structure between a range of different type and effect analyses and their semantics, as studied by Tolach [Tol98], Wadler [Wad98, WT03], Kieburtz [Kie98], Benton et al. [BK99, BKHB06, BB07, BKBH07, BKBH09], and Thamsborg and Birkedal [TB11]. On the other hand, we want our axioms to capture as much common structure as possible, allowing us to derive general properties and principles pertaining to uses of effect analysis, such as validating optimisations.

Our first step is to parameterise the theory over the hierarchy of effect sets in the chosen Gifford-style effect system of study:

Definition 3.1. *An effect hierarchy Σ is a pair $\langle \Pi, \mathcal{E} \rangle$ where Π is a set, and $\mathcal{E} \subseteq \mathcal{P}(\Pi)$.*

Given a hierarchy Σ , we call the elements of Π *effect operation names*, or effect operations for brevity, ranged over by op . We call the elements of \mathcal{E} the *effect sets*, ranged over by \mathcal{E} . For example, for a global memory cell, we take the hierarchy to be the full powerset:

$$\langle \{\{\text{lookup}, \text{update}\}, \{\emptyset, \{\text{lookup}\}, \{\text{update}\}, \{\text{lookup}, \text{update}\}\} \rangle$$

Ideally, the effect hierarchy includes all finite subsets of effect operations. However, in that case, the semantics involve exponentially large amount of data. Therefore, Tolmach [Tol98] and Kierbutz [Kie98] only consider sub-hierarchies of the finite

powerset hierarchy. As a consequence, we do not commit to a particular choice of hierarchy.

In Section 3.1, we introduce the model class of a given effect hierarchy Σ , and discuss the exponential quantity of data involved in specifying such models. Then, in Section 3.2, we utilise generic operations and describe techniques for specifying these models with smaller, yet still exponential, amounts of data. Finally, in Section 3.3, we conclude by instantiating our categorical definition to the category of sets and presenting a more elementary and accessible definition.

3.1 Categorical models



Without further ado, we define our notion of models for Gifford-style effect analysis:

Definition 3.2. *Let Σ be a hierarchy. A Σ -model is a quadruple*

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, O_- \llbracket - \rrbracket \rangle$$

where:

- \mathcal{V} is a distributive category, called the value category;
- type is a Σ -type assignment in \mathcal{V} ;
- \mathbb{P} is a functor $\mathcal{E} \rightarrow \text{CBPV}_q$, called the model hierarchy;
- $O_- \llbracket - \rrbracket$ assigns to each $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$, $\text{op} : A \langle P \rangle$ an algebraic operation $O_\varepsilon \llbracket \text{op} \rrbracket$ of type $A \langle P \rangle$ for $\mathbb{P}\varepsilon$;

and, for all $\varepsilon \subseteq \varepsilon'$, and $\text{op} \in \varepsilon$, $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ maps $O_\varepsilon \llbracket \text{op} \rrbracket$ to $O_{\varepsilon'} \llbracket \text{op} \rrbracket$. Thus, $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ preserves the operations.

Note that we considered the poset \mathcal{E} as a category in this definition.

Example 3-1. Given any effect hierarchy Σ and any Σ -model

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, O_- \llbracket - \rrbracket \rangle$$

we have a hierarchy of CBPV models. For every $\varepsilon \in \mathcal{E}$, we have a CBPV ε -model

$$\langle \mathcal{V}, \text{type}_\varepsilon, \mathbb{P}\varepsilon, O_\varepsilon \llbracket - \rrbracket \rangle$$

where type_ε is the restriction of the type assignment type to ε , namely $\text{type}|_\varepsilon$. □

Example 3-2. We illustrate Definition 3.2 by using global state.

The operations of our hierarchy Σ are given by $\Pi := \{\text{lookup}, \text{update}\}$, with \mathcal{E} given by the entire powerset $\mathcal{P}(\Pi)$. Let \mathbb{V} be a set of storable values. We describe a Σ -model $\langle \mathbf{Set}, \text{type}, \mathbb{P}, O_-[-] \rangle$ for a single global memory cell of type \mathbb{V} .

Assign types to the effect operations as in Example 2-9:

$$\text{lookup} : \mathbb{V} \quad \text{update} : \mathbb{1} \langle \mathbb{V} \rangle$$

We define the object part of \mathbb{P} by the identity, environment, overwrite, and global state monads:

$$\begin{aligned} \mathbb{P} \emptyset &:= T_{\text{id}} & \mathbb{P} \{\text{lookup}\} &:= T_{\text{Env}(\mathbb{V})} \\ \mathbb{P} \{\text{update}\} &:= T_{\text{OW}(\mathbb{V})} & \mathbb{P} \{\text{lookup}, \text{update}\} &:= T_{\text{GS}(\mathbb{V})} \end{aligned}$$

For each $\varepsilon \subseteq \Pi$, we define $\mathbb{P}(\emptyset \subseteq \varepsilon)$ to be the monadic unit $\eta : \text{id} \rightarrow T_\varepsilon$. As T_ε is strong, $\mathbb{P}(\emptyset \subseteq \varepsilon)$ is a strong monad morphism. Define:

$$\begin{aligned} \mathbb{P}(\{\text{lookup}\} \subseteq \Pi)_X &: T_{\text{Env}(\mathbb{V})}X \rightarrow T_{\text{GS}(\mathbb{V})}X \\ \mathbb{P}(\{\text{lookup}\} \subseteq \Pi)_X : k &\mapsto \lambda v. \langle v, k(v) \rangle \\ \mathbb{P}(\{\text{update}\} \subseteq \Pi)_X &: T_{\text{OW}(\mathbb{V})}X \rightarrow T_{\text{GS}(\mathbb{V})}X \\ \mathbb{P}(\{\text{update}\} \subseteq \Pi)_X : \langle \mathbf{1}_1 \star, x \rangle &\mapsto \lambda v. \underline{Y} \llbracket \langle v, x \rangle \rrbracket \\ &\langle \mathbf{1}_2 v', x \rangle \mapsto \lambda v. \underline{Y} \llbracket \langle v', x \rangle \rrbracket \end{aligned}$$

These are the two strong monad morphisms from Example 2-3.

The preservation of units under monad morphisms implies the commutativity of the following diagram.

$$\begin{array}{ccc} & \mathbb{P}|\Sigma| & \\ \mathbb{P}(\{\text{lookup}\} \subseteq |\Sigma|) \nearrow & & \nwarrow \mathbb{P}(\{\text{update}\} \subseteq |\Sigma|) \\ \mathbb{P}\{\text{lookup}\} & \xrightarrow{\text{monad morphism}} & \mathbb{P}\{\text{update}\} \\ \nwarrow \eta_{\{\text{lookup}\}} & \uparrow \eta_{|\Sigma|} & \nearrow \eta_{\{\text{update}\}} \\ & \mathbb{P}\emptyset & \end{array}$$

Thus, by setting $\mathbb{P}(\varepsilon \subseteq \varepsilon) := \text{id}$, we obtain a functor $\mathbb{P} : \mathcal{E} \rightarrow \text{CBPV } \mathbf{Set}$, and the model hierarchy component of the model is defined.

We choose the algebraic operations as in Examples 2-2 and 2-3, which show they are preserved by \mathbb{P} .

In conclusion, the quadruple $\langle \mathcal{V}, \text{type}, \mathbb{P}, O_-[-] \rangle$ is a Σ -model. \square

The last example demonstrates that in order to exhibit a Σ -model, we need to specify a large amount of data. To illustrate the problem, consider the typical case when the effect set Π comprises of n different operations, and we take the entire powerset $\mathcal{E} := \mathcal{P}(\Pi)$ for our hierarchy. Then, each Σ -model includes 2^n different monads. The number of monad morphisms we need to specify is

$$\sum_{\emptyset \neq \varepsilon \in \mathcal{E}} |\{m_{\varepsilon \setminus \{\text{op}\}} \mid \text{op} \in \varepsilon\}| = \sum_{k=1}^n \binom{n}{k} \cdot k = n2^{n-1}$$

The number of algebraic operations we need to specify is $n2^{n-1}$ as well:

$$\sum_{\varepsilon \in \mathcal{E}} |\varepsilon| = \sum_{k=0}^n \binom{n}{k} \cdot k = n2^{n-1}$$

Besides exhibiting these data, we also have to show it has the appropriate properties.

In particular:

- for each monad, we have to exhibit a strength, which also needs to be preserved by the monad morphisms, totaling in 4 commutative diagrams for each of the 2^n monads and a commutative diagram for each of the $n2^{n-1}$ monad morphisms;
- for each of the $n2^{n-1}$ algebraic operations we need to calculate $-^\dagger$ and $-^{\ddagger A}$, and establish the commutativity of a diagram; and,
- for each monad morphism $m_{\varepsilon \setminus \{\text{op}\}} \subseteq \varepsilon$ and $\text{op}' \in \varepsilon \setminus \{\text{op}\}$, establish the commutativity of the diagram for preserving op' :

$$\sum_{\emptyset \neq \varepsilon \in \mathcal{E}} \sum_{\text{op} \in \varepsilon} |\varepsilon \setminus \{\text{op}\}| = \sum_{k=1}^n \binom{n}{k} \cdot k(k-1) = n(n-1)2^{n-2}$$

In the following sections we will encounter different methods to elaborate the full model structure from a smaller set of data, and to avoid some of these proof-obligations.

3.2 Generic models



In light of this equivalence of algebraic operations and generic effects, we can recast our model definition in terms of generic operations. To aid the presentation, we use asterisks in our numbering. Thus, Definition 3.2* reflects that we recast Definition 3.2 in terms of generic effects. The shaded text highlights which parts of Definition 3.2 changed in Definition 3.2*.

Definition 3.2*. Let Σ be a hierarchy. A **generic Σ -model** is a quadruple

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_-[-] \rangle$$

where:

- \mathcal{V} is a distributive category, called the **value category**;
- type is a Σ -type assignment in \mathcal{V} ;
- \mathbb{P} is a functor $\mathcal{E} \rightarrow \text{CBPV}_{\mathcal{V}}$, called the **model hierarchy**;
- $\mathcal{G}_-[-]$ assigns to each $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$, $\text{op} : A \langle P \rangle$ a **generic effect $\mathcal{G}_\varepsilon[\text{op}]$** of type $A \langle P \rangle$ for $\mathbb{P}\varepsilon$;

and, for all $\varepsilon \subseteq \varepsilon'$, and $\text{op} \in \varepsilon$, $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ maps $\mathcal{G}_\varepsilon[\text{op}]$ to $\mathcal{G}_{\varepsilon'}[\text{op}]$. Thus, $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ preserves the **effects**.

Example 3-2*. Consider a global state model $\langle \text{Set}, \text{type}, \mathbb{P}, \mathcal{G}_-[-] \rangle$ where type and \mathbb{P} are as in Example 3-2, and $\mathcal{G}_-[-]$ assigns the generic effects from Example 2-4:

$$\begin{aligned} \mathcal{G}_{\{\text{lookup}, \text{update}\}}[\text{lookup}] : \star &\mapsto \lambda v. \langle v, v \rangle & \mathcal{G}_{\{\text{lookup}, \text{update}\}}[\text{update}] : v_0 &\mapsto \lambda v. \langle v_0, \star \rangle \\ \mathcal{G}_{\{\text{lookup}\}}[\text{lookup}] : \star &\mapsto \lambda v. v & \mathcal{G}_{\{\text{update}\}}[\text{update}] : v_0 &\mapsto \langle \text{!}_2 v_0, \star \rangle \quad \square \end{aligned}$$

The two Σ -model notions are equivalent:

Theorem 3.3. Let Σ be a hierarchy. Σ -models $\langle \mathcal{V}, \text{type}, \mathbb{P}, O_-[-] \rangle$ and generic Σ -models $\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_-[-] \rangle$ are in bijection given by $\text{gen}_{O_\varepsilon[\text{op}]} = \mathcal{G}_\varepsilon[\text{op}]$.

Proof

Immediate, in light of Theorems 2.4 and 2.8. ■

Generic models allow us to do away with the proof obligations regarding algebraic operations. We can simplify our models further by additional assumptions on the hierarchy Σ . We describe one possible way to simplify them.

Definition 3.4. We say that a hierarchy Σ is **reducible** if, for every $\text{op} \in \Pi$, the set $\{\varepsilon \in \mathcal{E} \mid \text{op} \in \varepsilon\}$ has a minimum ε_{op} .

Example 3-3. Given any set Π , the powerset $\mathcal{E} := \mathcal{P}(\Pi)$ is a reducible hierarchy, with $\varepsilon_{\text{op}} = \{\text{op}\}$. More generally, any hierarchy that includes all singletons $\{\text{op}\}$ is reducible. □

Non-example 3-4. For the set $\Pi := \{\text{lookup}, \text{update}, \text{raise}\}$, the hierarchy given by

$$\mathcal{E} := \{\{\text{lookup}, \text{raise}\}, \{\text{update}, \text{raise}\}, \{\text{lookup}, \text{update}, \text{raise}\}\}$$

is irreducible, as the set: $\{\varepsilon \in \mathcal{E} \mid \text{raise} \in \varepsilon\} = \mathcal{E}$ has no minimum. \square

Definition 3.5. Let Σ be a *reducible hierarchy*. A *reduced Σ -model* is a quadruple

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_{\varepsilon_-} \llbracket - \rrbracket \rangle$$

where:

- \mathcal{V} is a distributive category, called the value category;
- type is a Σ -type assignment in \mathcal{V} ;
- \mathbb{P} is a functor $\mathcal{E} \rightarrow \text{CBPV}_q$, called the model hierarchy;
- $\mathcal{G}_{\varepsilon_-} \llbracket - \rrbracket$ assigns to each $\text{op} \in \Pi$ a generic effect $\mathcal{G}_{\varepsilon_{\text{op}}} \llbracket \text{op} \rrbracket$ of type $A \langle P \rangle$ for $\mathbb{P} \varepsilon_{\text{op}}$;

(and that is all).

In other words, we no longer have to supply effect operations for every set in the hierarchy, merely to ε_{op} , for every effect operation op . Thus, just as in Example 3-2*, a reduced model for global state only includes the definitions for the lookup and update generic effects. The rest of the model structure can be elaborated from the reduced model:

Theorem 3.6. Let Σ be a *reducible hierarchy*. Reduced Σ -models $\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_{\varepsilon_-} \llbracket - \rrbracket \rangle$ and generic Σ -models $\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_- \llbracket - \rrbracket \rangle$ (and hence Σ -models) are in bijection given by the factorisation of $\mathcal{G}_- \llbracket - \rrbracket$ through $\mathcal{G}_{\varepsilon_-} \llbracket - \rrbracket$ as

$$\mathcal{G}_{\varepsilon_-} \llbracket - \rrbracket = \mathcal{G}_- \llbracket - \rrbracket \circ \langle \varepsilon_-, - \rangle$$

Proof

Given a generic Σ -model, the corresponding assignment $\mathcal{G}_{\varepsilon_-} \llbracket - \rrbracket$ is completely determined from $\mathcal{G}_- \llbracket - \rrbracket$. Conversely, preservation of generic effects implies that any Σ -model has to satisfy: $\mathcal{G}_{\varepsilon} \llbracket \text{op} \rrbracket = \mathbb{P}(\varepsilon_{\text{op}} \subseteq \varepsilon) \circ \mathcal{G}_{\varepsilon_{\text{op}}} \llbracket \text{op} \rrbracket$. By Corollary 2.9, this choice uniquely determines the generic effect $\mathcal{G}_{\varepsilon} \llbracket \text{op} \rrbracket$ preserved by \mathbb{P} . \blacksquare

Using generic effects, we succeeded in discharging the $n2^{n-1}$ proof obligations required in the definition for algebraic operations. We also demonstrated how the structure of the hierarchy can simplify the model structure further: our reduced models do away with $n^{n-1} - n$ operations and discharge all of the $n(n-1)2^{n-2}$ proof-obligations dealing with effect preservation. However, we are still left with the burden of specifying the 2^n monads and $n \cdot 2^{n-1}$ monad morphisms between them. Our definition of reducible hierarchies is not the only way to reduce the burden of model specification using the correspondence of generic effects and algebraic operations, but merely illustrates one way to do so. For example, we can also reduce the specification of models for the hierarchy from Non-example 3-4 by specifying lookup and update in their respective sets ε_{op} , specifying raise in the two subsets $\{\text{lookup}, \text{raise}\}$, $\{\text{update}, \text{raise}\}$, and requiring that they are mapped to the same effect by the monad morphisms in \mathbb{P} .

Such techniques are present in the functional programming community. In the Haskell programming language, such hierarchies of monads arise with modular decomposition of effectful programs, and are specified manually by the programmer. The requirement to also specify the operations manually in each level led to several methods for lifting the operations from the lower monads to the higher monads in the hierarchy. For example, Guts et al. [SGLH11] use a reduced model (as in Definition 3.5) and deduce from a given hierarchy of monads and monad morphisms how to lift the operations. As another example, Schrijvers and Oliveira [SO11] specify the operations at the lowest levels of the hierarchy and develop a domain specific language for choosing the appropriate monad morphism used to map the operation to a monad higher in the hierarchy.

3.3 Set-theoretic models



To aid accessibility, we remove some of the categorical scaffolding of our account. We instantiate our general categorical models to the category of sets. To aid the presentation, we use the (revisited) suffix to the reformulated categorical result. Thus, Definition 2.10 (revisited) is the set-theoretic specialisation of Definition 2.10.

Definition 2.10 (revisited). *Let Σ be a hierarchy. A set-theoretic type assignment for Σ is an assignment $\text{type}(\text{op}) = \langle A, P \rangle$ of a pair of sets to each $\text{op} \in \Pi$. We call A the arity of op , and P the parameter type of op .*

We write $\text{op} : A \langle P \rangle$ when $\text{type}(\text{op}) = \langle A, P \rangle$ and the assignment is clear from the

context. When the parameter type is a singleton $\mathbb{1} = \{\star\}$, we write $\text{op} : A$ instead of $\text{op} : A \langle \mathbb{1} \rangle$.

Example 2-9 (revisited). A set-theoretic type assignment for the global memory cell hierarchy is:

$$\text{lookup} : \mathbb{V} \quad \text{update} : \mathbb{1} \langle \mathbb{V} \rangle \quad \square$$

Definition 3.2* (revisited). Let Σ be a hierarchy. A set-theoretic Σ -model is a quadruple

$$\langle \text{type}, T_-, m_-, \mathcal{G}_- \llbracket - \rrbracket \rangle$$

where:

- type is a set-theoretic type assignment for Σ ;
- T_- assigns to each ε in \mathcal{E} a monad T_ε ;
- m_- assigns to each $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} a monad morphism $m_{\varepsilon \subseteq \varepsilon'} : T_\varepsilon \rightarrow T_{\varepsilon'}$;
- $\mathcal{G}_- \llbracket - \rrbracket$ assigns to each $\varepsilon \in \mathcal{E}$, and $\text{op} : A \langle P \rangle$ in ε a function $\mathcal{G}_\varepsilon \llbracket \text{op} \rrbracket : P \rightarrow T_\varepsilon A$;
(Such functions are called generic effects for T_ε .)
- for all $\varepsilon \subseteq \varepsilon' \subseteq \varepsilon''$ in \mathcal{E} :

$$m_{\varepsilon \subseteq \varepsilon} = \text{id}, \quad m_{\varepsilon' \subseteq \varepsilon''} \circ m_{\varepsilon \subseteq \varepsilon'} = m_{\varepsilon \subseteq \varepsilon''}$$

i.e., m_- is functorial;

and, for all $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} and $\text{op} : A \langle P \rangle$ in ε :

$$\begin{array}{ccc} & & T_\varepsilon A \\ & \nearrow \mathcal{G}_\varepsilon \llbracket \text{op} \rrbracket & \downarrow m_{\varepsilon \subseteq \varepsilon'} \\ P & = & \\ & \searrow \mathcal{G}_{\varepsilon'} \llbracket \text{op} \rrbracket & T_{\varepsilon'} A \end{array}$$

Thus, m_- preserves the generic effects.

Example 3-2* (revisited). We construct a set-theoretic model for global state.

The global state hierarchy is given by $\Pi := \{\text{lookup}, \text{update}\}$ and $\mathcal{E} := \mathcal{P}(\Pi)$.

Let \mathbb{V} be a set of storable values. We describe a set-theoretic Σ -model for a single global memory cell of type \mathbb{V} , $\langle \text{type}, T_-, m_-, \mathcal{G}_- \llbracket - \rrbracket \rangle$.

First, we assign types to the effect operations:

$$\text{lookup} : \mathbb{V} \quad \text{update} : \mathbb{1} \langle \mathbb{V} \rangle$$

Next, we choose the following monads:

$$\begin{aligned} T_{\emptyset} & \quad X := T_{\text{id}} \quad X := X \\ T_{\{\text{lookup}\}} & \quad X := T_{\text{Env}(\mathbb{V})} X := X^{\mathbb{V}} \\ T_{\{\text{update}\}} & \quad X := T_{\text{OW}(\mathbb{V})} X := X := (\mathbb{1} + \mathbb{V}) \times X \\ T_{\{\text{lookup}, \text{update}\}} & \quad X := T_{\text{GS}(\mathbb{V})} X := (\mathbb{V} \times X)^{\mathbb{V}} \end{aligned}$$

i.e., the identity, environment, overwrite, and global state monads.

For each $\varepsilon \subseteq \Pi$, we define $m_{\emptyset \subseteq \varepsilon}$ to be the monadic unit $\eta_{\varepsilon} : \text{id} \rightarrow T_{\varepsilon}$.

Define:

$$\begin{aligned} m_{\{\text{lookup}\} \subseteq \Pi} & : (-)^{\mathbb{V}} \rightarrow (\mathbb{V} \times (-))^{\mathbb{V}} \\ m_{\{\text{lookup}\} \subseteq \Pi} & : k \quad \mapsto \lambda v. \langle v, k(v) \rangle \end{aligned}$$

Calculation shows $m_{\{\text{lookup}\} \subseteq \Pi}$ is a monad morphism from $T_{\{\text{lookup}\}}$ to T_{Π} . Define:

$$\begin{aligned} m_{\{\text{update}\} \subseteq \Pi} & : (\mathbb{1} + \mathbb{V}) \times (-) \rightarrow (\mathbb{V} \times (-))^{\mathbb{V}} \\ m_{\{\text{update}\} \subseteq \Pi} & : \langle \delta, x \rangle \quad \mapsto \begin{cases} \lambda v. \langle v, x \rangle & \delta = \mathbf{1}_1 \star \\ \lambda v. \langle v', x \rangle & \delta = \mathbf{1}_2 v' \end{cases} \end{aligned}$$

Calculation shows $m_{\{\text{update}\} \subseteq \Pi}$ is also a monad morphism, this time from $T_{\{\text{update}\}}$ to T_{Π} .

The preservation of units under monad morphisms implies the commutativity of the following diagram.

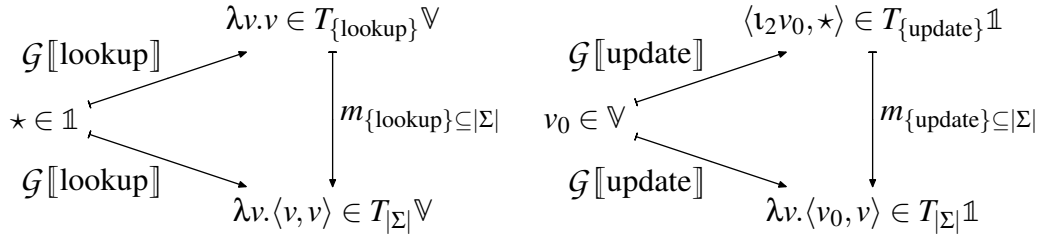
$$\begin{array}{ccc} & T_{|\Sigma|} & \\ m_{\{\text{lookup}\} \subseteq |\Sigma|} \nearrow & & \nwarrow m_{\{\text{update}\} \subseteq |\Sigma|} \\ T_{\{\text{lookup}\}} & \text{monad} & T_{\{\text{update}\}} \\ & \underline{\text{morphism}} & \\ \eta_{\{\text{lookup}\}} \nwarrow & \eta_{|\Sigma|} & \nearrow \eta_{\{\text{update}\}} \\ & T_{\emptyset} & \end{array}$$

Thus, by setting $m_{\varepsilon \subseteq \varepsilon} := \text{id}$, we obtain a functorial family m_{-} .

We choose the generic effects:

$$\begin{aligned} \mathcal{G}_{\{\text{lookup}, \text{update}\}} \llbracket \text{lookup} \rrbracket & : \star \mapsto \lambda v. \langle v, v \rangle & \mathcal{G}_{\{\text{lookup}, \text{update}\}} \llbracket \text{update} \rrbracket & : v_0 \mapsto \lambda v. \langle v_0, \star \rangle \\ \mathcal{G}_{\{\text{lookup}\}} \llbracket \text{lookup} \rrbracket & : \star \mapsto \lambda v. v & \mathcal{G}_{\{\text{update}\}} \llbracket \text{update} \rrbracket & : v_0 \mapsto \langle \mathbf{1}_2 v_0, \star \rangle \end{aligned}$$

Finally, we show preservation of operations by m . It amounts to chasing two diagrams:



In conclusion, the quadruple $\langle \text{type}, T_-, m_-, \mathcal{G}_-[-] \rangle$ is a set-theoretic Σ -model. \square

Example 3-5. We construct a set-theoretic model for exceptions and input/output interactions with a terminal (terminal I/O).

We choose the hierarchy as $\Pi := \{\text{raise}, \text{input}, \text{output}\}$, and set $\mathcal{E} := \mathcal{P}(\Pi)$.

Let Char be a set modeling the characters in an interactive terminal. For example, we may choose $\text{Char} := \{0, 1, 2, \dots, 127\}$ for modeling ASCII characters. Let Str be Char^* , the set of strings, i.e., finite sequences of characters.

The model $\langle \text{type}, T_-, m_-, \mathcal{G}_-[-] \rangle$ is given as follows.

The type assignment is given by

$$\text{raise} : \emptyset \langle \text{Str} \rangle \qquad \text{input} : \text{Char} \qquad \text{output} : \mathbb{1} \langle \text{Char} \rangle$$

Given any $\varepsilon \in \mathcal{E}$, any set X , and any three different sets $I, O, R \notin X$, we define $T_\varepsilon X$ inductively as follows:

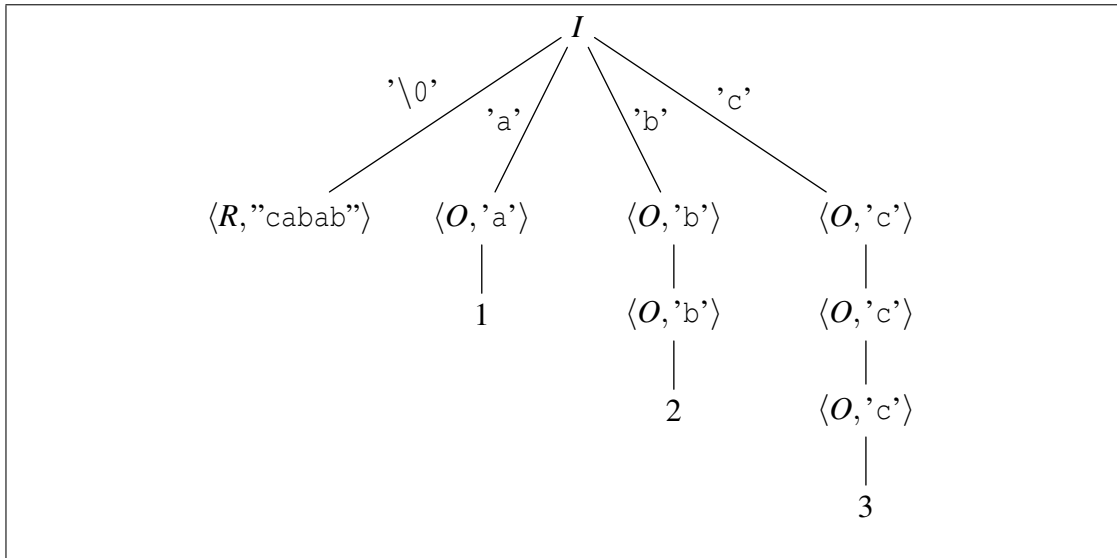
- if $x \in X$, then x is in $T_\varepsilon X$;
- if $\text{raise} \in \varepsilon$, and $s \in \text{Str}$, then $\langle R, s \rangle$ is in $T_\varepsilon X$.
- if $\text{input} \in \varepsilon$, and for all $c \in \text{Char}$, $t_c \in T_\varepsilon X$, then $\langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle$ is in $T_\varepsilon X$;
- if $\text{output} \in \varepsilon$, $c \in \text{Char}$, and $t \in T_\varepsilon X$, then $\langle O, c, t \rangle$ is in $T_\varepsilon X$;

In other words, $T_\varepsilon X$ is the set of labeled trees, whose leaves are X -elements. If $\text{input} \in \varepsilon$, then these trees may contain nodes, labeled by I , whose branches are indexed by Char . If $\text{output} \in \varepsilon$ then these trees may contain nodes, labeled by a pair of labels $\langle O, c \rangle$ (where $c \in \text{Char}$), with a single branch stemming from them. If $\text{raise} \in \varepsilon$, then these trees may also have leaves from Str , and these strings are distinguishable from the X -elements.

In terms of universal algebra, $T_\Pi X$, for example, is the free algebra for the following signature σ :

$$\{\text{raise}_s : \emptyset, \text{input} : |\text{Char}|, \text{output}_c : \mathbb{1} \mid c \in \text{Char}, s \in \text{Str}\}$$

We can also model the set $T_\Pi X$ using the SML algebraic datatype

Figure 3.1: An example element of $T_{\Pi}N$

```

- datatype 'a T = Just of 'a
=           | Raise of string
=           | Input of char -> 'a T
=           | Output of char * 'a T

```

We can present these trees diagrammatically. See Figure 3.1 for an example element of $T_{\Pi}N$, when $\text{Char} = \{\backslash 0, 'a', 'b', 'c'\}$.

The monadic unit is given by the inclusion $\eta_{\varepsilon} : X \subseteq T_{\varepsilon}X$. The monadic multiplication is given inductively:

- $\mu_{\varepsilon}(t) := t$;
- If `raise` $\in \varepsilon$, $\mu_{\varepsilon}\langle R, s \rangle := \langle R, s \rangle$.
- If `input` $\in \varepsilon$, $\mu_{\varepsilon}\langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle := \langle I, \langle \mu_{\varepsilon}(t_c) \rangle_{c \in \text{Char}} \rangle$;
- If `output` $\in \varepsilon$, $\mu_{\varepsilon}\langle O, c, t \rangle := \langle O, c, \mu_{\varepsilon}(t) \rangle$;

i.e., by collapsing the nested tree structure. Figure 3.2 demonstrates the action of the multiplication. The monad $\langle T_{\Pi}, \eta, \mu \rangle$ is the free algebra monad for the universal algebra signature σ we described above. It is also the *free monad* for the functor:

$$FX := \text{Str} + X^{\text{Char}} + \text{Char} \times X$$

Similarly, the other monads T_{ε} are free algebra monads for analogous signatures and free monads for analogous functors.

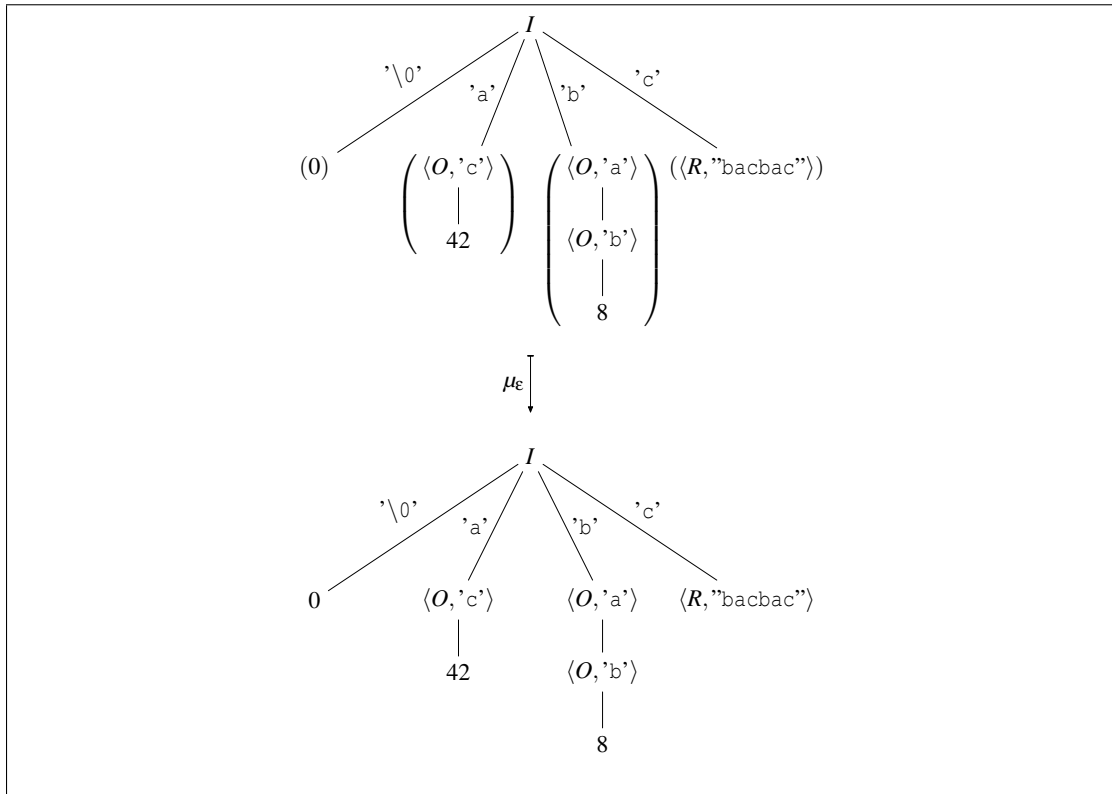


Figure 3.2: monadic multiplication

Thus we specified the monad hierarchy T_- .

The monad morphisms are given by inclusion: $(m_{\varepsilon \subseteq \varepsilon'})_X : T_\varepsilon X \subseteq T_{\varepsilon'} X$, hence functorial.

The generic effects are given by:

- if $\text{input} \in \varepsilon$:

$$\begin{aligned} \mathcal{G}_\varepsilon[\text{input}] &: \mathbb{1} \rightarrow T_\varepsilon \text{Char} \\ \mathcal{G}_\varepsilon[\text{input}] &: \star \mapsto \langle I, \langle c \rangle_{c \in \text{Char}} \rangle \end{aligned}$$

- if $\text{output} \in \varepsilon$:

$$\begin{aligned} \mathcal{G}_\varepsilon[\text{output}] &: \text{Char} \rightarrow T_\varepsilon \mathbb{1} \\ \mathcal{G}_\varepsilon[\text{output}] &: c \mapsto \langle O, c, \star \rangle \end{aligned}$$

- if $\text{raise} \in \varepsilon$:

$$\begin{aligned} \mathcal{G}_\varepsilon[\text{raise}] &: \text{Str} \rightarrow T_\varepsilon \mathbb{0} \\ \mathcal{G}_\varepsilon[\text{raise}] &: s \mapsto \langle R, s \rangle \end{aligned}$$

Note that the inclusions m_- preserve the generic effects, as $\mathcal{G}_-[-]$ were identically defined for every ε .

In conclusion, $\langle \text{type}, T_-, m_-, \mathcal{G}_-[-] \rangle$ is a set-theoretic Σ -model. \square

Contrast the last two examples. For global state, we explicitly defined an ad-hoc monad for each effect set ε . For exceptions and I/O, our definition was uniform for all effect sets: we define a single monad T_{Π} , and omit the irrelevant elements according to the effect set. The same idea applies to the generic effects: they are defined in the same manner for all ε . Specifying Σ -models in this manner dramatically decreases the amount of data we have to specify: as we have 3 effects, instead of specifying 9 monads, 12 monad morphisms, and 12 generic effects, we specify, uniformly, 1 monad, 1 monad morphism, and 3 generic effects. We will see in Chapters 7 and 8 that this process can be generalised to arbitrary algebraic models, including the global state model.



To conclude this section, we show that set-theoretic models coincide with categorical models with $\mathcal{V} = \mathbf{Set}$:

Proposition 3.7. *Let Σ be a hierarchy. Set-theoretic Σ -models $\langle \text{type}, T_-, m_-, G_- \llbracket - \rrbracket \rangle$ and (categorical) Σ -models $\langle \mathbf{Set}, \text{type}, \mathbb{P}, G_- \llbracket - \rrbracket \rangle$ are in bijection given by*

- for all $\varepsilon \in \mathcal{E}$, $\mathbb{P}\varepsilon = T_\varepsilon$; and
- for all $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} , $\mathbb{P}(\varepsilon \subseteq \varepsilon') = m_{\varepsilon \subseteq \varepsilon'}$.

Proof

Functoriality of m_- guarantees the functor \mathbb{P} is uniquely determined by the bijection. It remains to show well-definedness, by equipping each monad T_ε with a strength to form a strong monad $\mathbb{P}\varepsilon$, such that $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ is a strong monad morphisms with respect to the strengths of $\mathbb{P}\varepsilon$ and $\mathbb{P}\varepsilon'$.

The following result follows directly from Kock [Koc72]: every monad over \mathbf{Set} has a unique strength, given by $\text{str}_{A,B} : \langle a, k \rangle \mapsto T(\lambda b. \langle a, b \rangle)(k)$. Thus $\mathbb{P}\varepsilon$ is a well-defined strong monad.

For all $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} , $m_{\varepsilon \subseteq \varepsilon'}$ is also a *strong* monad morphism, by chasing a diagram using the naturality of $m_{\varepsilon \subseteq \varepsilon'}$:

$$\begin{array}{ccc}
 \langle a, k \rangle & \xrightarrow{\text{str}^\varepsilon} & T_\varepsilon(\lambda b. \langle a, b \rangle)(k) \\
 \downarrow A \times m_{\varepsilon \subseteq \varepsilon'} & & \downarrow m_{\varepsilon \subseteq \varepsilon'} \\
 & & m(T_\varepsilon(\lambda b. \langle a, b \rangle)(k)) \\
 & & \parallel \leftarrow m_{\varepsilon \subseteq \varepsilon'} \text{ naturality} \\
 \langle a, m(k) \rangle & \xrightarrow{\text{str}^{\varepsilon'}} & T_{\varepsilon'}(\lambda b. \langle a, b \rangle)(m(k))
 \end{array}$$

Thus \mathbb{P} is uniquely determined by T_- and m_- , and we are done. ■



In summary, we defined the notion of an effect hierarchy, and categorical hierarchical models. We saw that in order to specify a complete hierarchy we need to produce exponential amounts of data. We managed to do away with some of this burden, but we could not go below an exponential amount.

Chapter 4

Recursion

A scar on my arm that says: Domain

—Lou Reed



In this section we isolate a subclass of our categorical models suitable for modelling languages with recursion. To achieve this, we refine the definitions from the previous chapters by requiring additional structure.

Programs involving recursive functions may fail to terminate. Thus they exhibit *divergence* as an effect. Therefore, effect hierarchies for recursive languages have divergence as a distinguished effect:

Definition 3.1 ω . A recursion effect hierarchy is a pair $\langle \Sigma, \text{diverge} \rangle$ where Σ is an effect hierarchy with a specified effect operation $\text{diverge} \in \Pi_{\Sigma}$.

To aid the presentation, we use the ω suffix to reflect that the revisited unit is a modification of a previously presented unit to accommodate recursion. Thus, Definition 3.1 ω modifies Definition 3.1.

We treat a recursion hierarchy $\langle \Sigma, \text{diverge} \rangle$ as a hierarchy Σ , and always denote the specified operation by “diverge”. For example, a recursion hierarchy Σ for non-determinism consists of two operations $\Pi := \{\text{choose}, \text{diverge}\}$ and the full powerset.

For the more semantic aspects of our models, we turn to *domain theory*. Domain theory provides the standard mathematical machinery needed to model recursion. The simplest type of domain sufficient for our purposes is an ω -complete partial order (ω -cpo). We refine our models to account for recursion by demanding them to be suitably *enriched* in the category of ω -cpos. For this purpose, we review the key concepts from the previous chapters and define their enriched counterparts.

We proceed as follows. We first recapitulate some standard domain-theoretic concepts and notation. Then, we recall how to incorporate recursion into (Eilenberg-Moore) CBPV models, and enrich the notions of algebraic operations for an enriched monad. Next, we refine the structure of our categorical models to accommodate recursion. Finally, we specialise to the category of ω -cpos, removing many of the categorical concepts, for a more accessible account.

4.1 Domain-theoretic preliminaries

An ω -complete partial order (ω -cpo) is a partial order $W = \langle |W|, \leq \rangle$, closed under suprema $\bigvee_{n \in \mathbb{N}} w_n$ of increasing ω -chains $\langle w_n \rangle_{n \in \mathbb{N}}$. A (Scott) continuous function $f : W \rightarrow W'$ between ω -cpos is a monotone function preserving such suprema, i.e., $f(\bigvee_n w_n) = \bigvee_n f(w_n)$ for every increasing ω -chain $\langle w_n \rangle$.

A discrete ω -cpo is a set ordered by equality $\langle X, = \rangle$. The empty ω -cpo $\mathbb{0}$ is $\langle \emptyset, = \rangle$. The singleton ω -cpo $\mathbb{1}$ is $\langle \{\star\}, = \rangle$. We write \mathfrak{n} for the discrete domain over n elements.

The product $W_1 \times W_2$ of two ω -cpos is the component-wise partial order over their cartesian product $|W_1| \times |W_2|$:

$$\langle w_1, w_2 \rangle \leq \langle w'_1, w'_2 \rangle \text{ if and only if } w_1 \leq w'_1 \text{ and } w_2 \leq w'_2$$

This order does give rise to an ω -cpo, with suprema taken componentwise.

The exponential $W_2^{W_1}$ of two ω -cpos is the ω -cpo consisting of all continuous functions from W_1 to W_2 , ordered pointwise:

$$f \leq g \text{ if and only if, for all } W \in W_1, f(w) \leq g(w)$$

This order does give rise to an ω -cpo, with suprema taken pointwise. The evaluation map is the continuous function $\text{eval} : W_2^{W_1} \times W_1 \rightarrow W_2$ given by $\langle f, w \rangle \mapsto f(w)$.

An ω -cpo W is *pointed* if it has a least element \perp . If W is pointed, then, for all V , W^V is pointed, with $\lambda v. \perp$ its least element. A continuous function $f : V \rightarrow W$ between pointed domains is called *strict* if $f(\perp) = \perp$.

In categorical terms, the category $\omega\mathbf{CPO}$ has ω -cpos as objects, and Scott continuous functions between them as morphisms. An endofunctor $F : \omega\mathbf{CPO} \rightarrow \omega\mathbf{CPO}$ is *locally continuous* if, for every pair of ω -cpos V, W , the morphism map F , sending a continuous function $f : V \rightarrow W$ to $Tf : FV \rightarrow FW$, is a continuous function $F : W^V \rightarrow (FW)^{FV}$. An endofunctor F over $\omega\mathbf{CPO}$ is called *pointed* if, for every ω -cpo W , FW is pointed. A pointed functor is called *strict*, if, for all $f : V \rightarrow W$,

$(Ff)(\perp_V) = \perp_{FW}$, i.e., if every Ff is strict, even when f is not necessarily strict. A *locally continuous monad* is a monad whose underlying functor is locally continuous. We define *pointed monads* and *strict monads* similarly.



The evident forgetful functor $|-|$ from $\omega\mathbf{CPO}$ to \mathbf{Set} has a left adjoint, assigning to each set X the discrete ω -cpo with carrier X , $\langle X, = \rangle$. The empty ω -cpo $\mathbb{0}$ is the initial object in $\omega\mathbf{CPO}$. Binary sums are given by disjoint union:

$$W_1 + W_2 \cong \langle |W_1| + |W_2|, \{ \langle \iota_i w, \iota_i w' \rangle \mid i = 1, 2, w \leq_i w' \} \rangle$$

The singleton domain $\mathbb{1}$ is the terminal object. As left adjoints preserve coproducts, we indeed have

$$\mathbb{n} = \overbrace{\mathbb{1} + \dots + \mathbb{1}}^{n \text{ summands}}$$

The product $W \times V$ described earlier is the categorical product. In fact, $\omega\mathbf{CPO}$ is complete. If $D : J \rightarrow \omega\mathbf{CPO}$ is any small diagram, the limit $\text{Lim}D = \langle L, \sigma \rangle$ is given by:

- $|L|$ and $|\sigma|$ form the limit of the underlying set-theoretic diagram:

$$\text{Lim}_{\mathbf{Set}} |D| = \langle |L|, |\sigma| \rangle$$

- the partial order is given by

$$w \leq v \text{ if and only if, for all } j \in \text{Ob}(J): \sigma_j(w) \leq \sigma_j v$$

and

- suprema are given by the following property:

$$w = \bigvee_n w_n \text{ if and only if, for all } j \in \text{Ob}(J): \sigma_j(w) = \bigvee_n \sigma_j(w_n)$$

The exponential W^V is the categorical exponential of W by V , and the evaluation map is the categorical evaluation morphism. Given a Scott continuous function $f : U \times V \rightarrow W$, the map $\lambda V.f$ is given by currying $u \mapsto \lambda v.f(u, v)$, just as in \mathbf{Set} . Therefore, $\omega\mathbf{CPO}$ is cartesian closed, hence a distributive category.

An $\omega\mathbf{CPO}$ -enriched category is a category \mathcal{V} such that, for all objects W, V in $\text{Ob}(\mathcal{V})$, the homset $\mathcal{V}(W, V)$ is equipped with a specified partial order making it an ω -cpo, also denoted by $\mathcal{V}(W, V)$, and, for all $W, V, U \in \text{Ob}(\mathcal{V})$, composition is a Scott continuous function:

$$\circ : \mathcal{V}(V, W) \times \mathcal{V}(U, V) \rightarrow \mathcal{V}(U, W)$$

Let \mathcal{V} be an $\omega\mathbf{CPO}$ -enriched category, I a set, and $W_i \in \text{Ob}(\mathcal{V})$ an I -indexed family of objects. An $\omega\mathbf{CPO}$ -enriched product $\prod_{i \in I} W_i$ is an ordinary product $\prod_{i \in I} W_i$ in the underlying ordinary category of \mathcal{V} such that, for all $V \in \text{Ob}(\mathcal{V})$, the evident tupling map is continuous:

$$\langle - \rangle_{i \in I} : \prod_{i \in I} \mathcal{V}(V, W_i) \rightarrow \mathcal{V}(V, \prod_{i \in I} W_i)$$

Similarly, an $\omega\mathbf{CPO}$ -enriched coproduct $\sum_{i \in I} W_i$ is an ordinary coproduct $\sum_{i \in I} W_i$ for which, for all $V \in \text{Ob}(\mathcal{V})$, the evident cotupling map is continuous:

$$[-]_{i \in I} : \prod_{i \in I} \mathcal{V}(W_i, V) \rightarrow \mathcal{V}(\sum_{i \in I} W_i, V)$$

Thus, an $\omega\mathbf{CPO}$ -enriched distributive category is an $\omega\mathbf{CPO}$ -enriched category with all finite $\omega\mathbf{CPO}$ -enriched products and coproducts such that finite products distribute over finite coproducts.

Let $W, V \in \mathcal{V}$ be two objects in an $\omega\mathbf{CPO}$ -enriched category. An $\omega\mathbf{CPO}$ -enriched exponential W^V is an ordinary exponential W^V for which, for all $U \in \text{Ob}(\mathcal{V})$, the currying map is continuous:

$$\lambda_V . - : \mathcal{V}(U \times V, W) \rightarrow \mathcal{V}(U, W^V)$$

Let $\mathcal{V}, \mathcal{V}'$ be $\omega\mathbf{CPO}$ -enriched categories. An $\omega\mathbf{CPO}$ -enriched functor $F : \mathcal{V} \rightarrow \mathcal{V}'$ is a functor $F : \mathcal{V} \rightarrow \mathcal{V}'$ between the underlying ordinary categories, such that, for all $A, B \in \text{Ob}(\mathcal{V})$, the induced morphism map is Scott continuous:

$$F : \mathcal{V}(A, B) \rightarrow \mathcal{V}'(FA, FB)$$

Thus, a locally continuous functor is exactly an $\omega\mathbf{CPO}$ -enriched endofunctor over $\omega\mathbf{CPO}$. We note that there is no need to differentiate between $\omega\mathbf{CPO}$ -enriched natural transformations and ordinary natural transformations between $\omega\mathbf{CPO}$ -enriched functors. Thus, an $\omega\mathbf{CPO}$ -enriched (strong) monad is an ordinary (strong) monad whose underlying functor is $\omega\mathbf{CPO}$ -enriched. Similarly, an $\omega\mathbf{CPO}$ -enriched adjunction is an ordinary adjunction $F \dashv G : \mathcal{C} \rightarrow \mathcal{V}$ between the underlying ordinary functors of two $\omega\mathbf{CPO}$ -enriched functors F and G . The bijection $\varphi : \mathcal{C}(FA, \underline{B}) \cong \mathcal{V}(A, G\underline{B})$ of the adjunction is then Scott continuous as, by naturality, $\varphi(f) = G(f) \circ \eta$. Finally, an $\omega\mathbf{CPO}$ -enriched resolution of an $\omega\mathbf{CPO}$ -enriched monad is an $\omega\mathbf{CPO}$ -enriched adjunction whose underlying ordinary adjunction resolves the underlying ordinary monad. A

direct consequence of Kock's work [Koc72] is that every endofunctor (monad) over $\omega\mathbf{CPO}$ has a unique candidate for strength, i.e., the map:

$$\begin{aligned} \text{str} : V \times FW &\rightarrow F(V \times W) \\ \text{str} : \langle v, t \rangle &\mapsto F(\lambda w. \langle v, w \rangle)(t) \end{aligned}$$

If this candidate is indeed Scott-continuous, this functor (monad) is locally continuous (strong).

The cartesian closed structure of $\omega\mathbf{CPO}$ itself is an $\omega\mathbf{CPO}$ -enriched cartesian closed category. In particular, it is an $\omega\mathbf{CPO}$ -enriched distributive category.

Lemma 4.1. *Let \mathcal{V} be an $\omega\mathbf{CPO}$ -enriched category, and let $A \in \text{Ob}(\mathcal{V})$ be any object such that, for all $\Gamma \in \text{Ob}(\mathcal{V})$, the $\omega\mathbf{CPO}$ -enriched exponential A^Γ exists. If, for all $\Gamma \in \text{Ob}(\mathcal{V})$, $\mathcal{V}(\Gamma, A)$ is pointed, then, for all $\Delta \in \text{Ob}(\mathcal{V})$, $\mathcal{V}(\Delta, A^\Gamma)$ is pointed, and $\perp_{\Delta, A^\Gamma} = \lambda\Gamma. \perp_{\Delta \times \Gamma, A}$.*

Proof

By fiat, $\lambda\Gamma. (-) : \mathcal{V}(\Delta \times \Gamma, A) \xrightarrow{\cong} \mathcal{V}(\Delta, A^\Gamma)$ is a continuous isomorphism. As $\perp_{\Delta \times \Gamma, A}$ is the least element of $\mathcal{V}(\Delta \times \Gamma, A)$, $\lambda\Gamma. \perp_{\Delta \times \Gamma, A}$ is the least element of $\mathcal{V}(\Delta, A^\Gamma)$. Thus, $\mathcal{V}(\Delta, A^\Gamma)$ is pointed. ■

4.2 $\omega\mathbf{CPO}$ -enriched CBPV models



We turn to incorporate recursion into CBPV. The following notion appears in Levy's work [Lev01, Lev04]:

Definition 4.2. *Let T be a strong monad over a cartesian category \mathcal{V} , let $\underline{B}, \underline{B}'$ in $\text{Ob}(\mathcal{V}^T)$ be two algebras, and let $\Gamma \in \text{Ob}(\mathcal{V})$ be any \mathcal{V} -object. A T -algebra homomorphism from \underline{B} to \underline{B}' over Γ is a morphism $h : \Gamma \times |\underline{B}| \rightarrow |\underline{B}'|$, satisfying:*

$$\begin{array}{ccc} \Gamma \times T|\underline{B}| & \xrightarrow{\text{str}} & T(\Gamma \times |\underline{B}|) \xrightarrow{Th} T|\underline{B}'| \\ \downarrow \Gamma \times \underline{B}[-] & & \downarrow \underline{B}'[-] \\ \Gamma \times |\underline{B}| & \xrightarrow{h} & |\underline{B}'| \end{array} \quad =$$

We will make use of the following properties of algebras:

Lemma 4.3. *Let T be a strong monad over a cartesian category \mathcal{V} , and $\underline{B}, \underline{C}$ be any T -algebras.*

1. If $h : \underline{B} \rightarrow \underline{C}$ is a T -algebra homomorphism in the usual sense, then, for all Γ , the following morphism is an algebra homomorphism over Γ :

$$\Gamma \times |\underline{B}| \xrightarrow{\pi_2} |\underline{B}| \xrightarrow{h} |\underline{C}|$$

2. If the exponential $|\underline{C}|^\Gamma$ exists in \mathcal{V} , then there is a T -algebra structure over $|\underline{C}|^\Gamma$, denoted by \underline{C}^Γ , given by:

$$\lambda\Gamma. \left(T(|\underline{C}|^\Gamma) \times \Gamma \xrightarrow{\text{costr}} T(|\underline{C}| \times \Gamma) \xrightarrow{T\text{eval}} T|\underline{C}| \xrightarrow{\underline{C}[-]} |\underline{C}| \right)$$

In this case, every algebra homomorphism over Γ , $h : \Gamma \times |\underline{B}| \rightarrow |\underline{C}|$, gives rise to a (unique) algebra homomorphism from \underline{B} to \underline{C}^Γ :

$$\lambda\Gamma. \left(|\underline{B}| \times \Gamma \xrightarrow{\text{swap}} \Gamma \times |\underline{B}| \xrightarrow{h} |\underline{C}| \right)$$

3. For all morphisms $f : \Gamma \times A \rightarrow |\underline{B}|$, the lifting $f^\dagger : \Gamma \times TA \rightarrow |\underline{B}|$ is the unique T -algebra homomorphism from FA to \underline{B} over Γ , satisfying

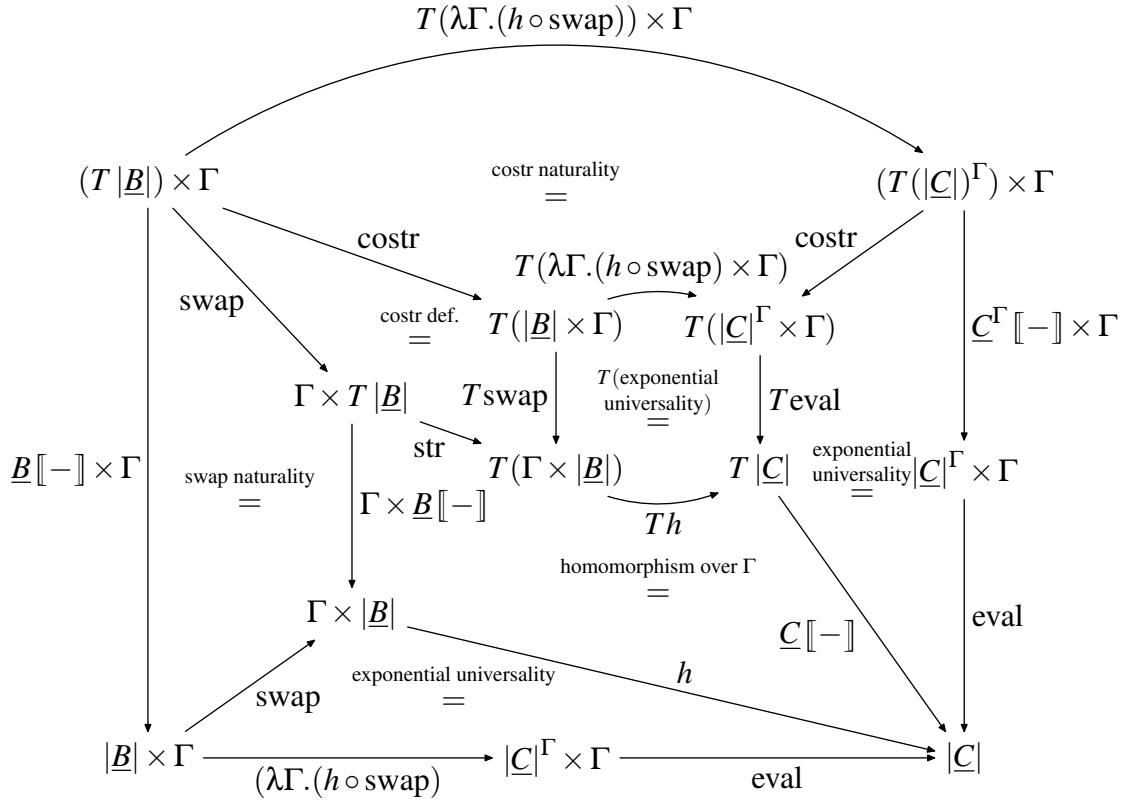
$$\begin{array}{ccc} & \Gamma \times TA & \\ & \uparrow & \searrow f^\dagger \\ \Gamma \times \eta & & \\ & \Gamma \times A & \xrightarrow{f} |\underline{B}| \\ & & \end{array} \quad =$$

Proof

The following diagram proves part 1:

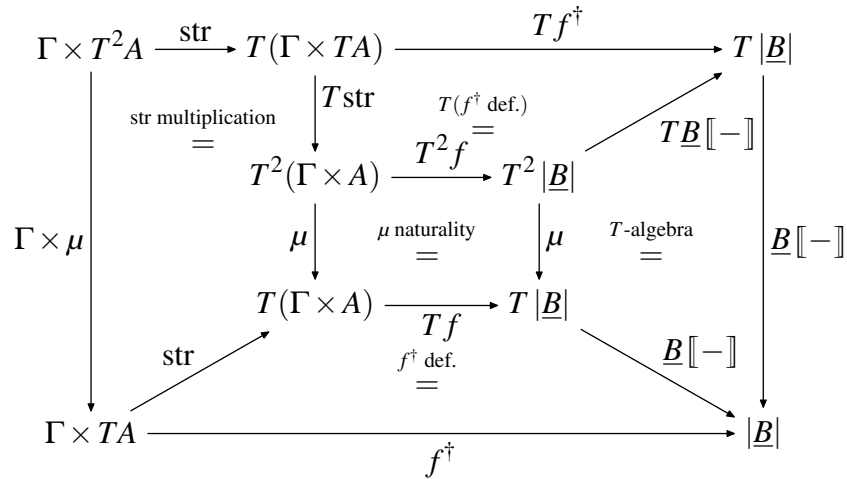
$$\begin{array}{ccccccc} \Gamma \times T|\underline{B}| & \xrightarrow{\text{str}} & T(\Gamma \times |\underline{B}|) & \xrightarrow{T\pi_2} & T|\underline{B}| & \xrightarrow{Th} & T|\underline{C}| \\ \downarrow \Gamma \times \underline{B}[-] & \searrow \text{str proj.} & \downarrow \pi_2 & \searrow \text{algebra homomorphism} & \downarrow \underline{B}[-] & \searrow \underline{C}[-] & \downarrow \underline{C}[-] \\ & \xrightarrow{\pi_2} & \Gamma \times |\underline{B}| & \xrightarrow{\pi_2} & |\underline{B}| & \xrightarrow{h} & |\underline{C}| \\ & \searrow \pi_2 \text{ naturality} & & \searrow & & & \end{array}$$

In part 2, the fact that \underline{C}^Γ is a T -algebra is well-known, and follows by routine calculation. The fact that $\lambda\Gamma.(h \circ \text{swap})$ is a homomorphism follows, by the exponential universal property, from the following diagram:



The uniqueness proof is also straightforward, but we omit it, as we will not make use of it in the sequel.

Finally, the following diagram proves part 3:



Straightforward calculation shows that

$$\begin{array}{ccc}
& \Gamma \times TA & \\
& \uparrow \Gamma \times \eta & \text{---} f^\dagger \text{---} \\
& \Gamma \times A & \xrightarrow{f} \underline{|B|} \\
& & \text{---} = \text{---}
\end{array}$$

and that f^\dagger is the unique such homomorphism, but we omit these calculations as we will not make use of them in the sequel. ■

Definition 2.11 ω . An ω CPO-enriched (Eilenberg-Moore) CBPV model is a pair $\langle \mathcal{V}, T \rangle$ where:

- \mathcal{V} is an ω CPO-enriched distributive category;
- T is an ω CPO-enriched strong monad over \mathcal{V} ; and
- \mathcal{V} has all ω CPO-enriched exponentials $|B|^A$ of all T -algebra carriers $|B|$ by all objects A in \mathcal{V} .

A recursion CBPV model is an ω CPO-enriched model $\langle \mathcal{V}, T \rangle$ for which:

- for all objects $\Gamma \in \text{Ob}(\mathcal{V})$ and algebras $\underline{B} \in \text{Ob}(\mathcal{V}^T)$, $\mathcal{V}(\Gamma, |\underline{B}|)$ is pointed; and,
- for all $f : \Delta \rightarrow \Gamma$ in \mathcal{V} and homomorphisms h from \underline{B} to \underline{C} over Γ :

$$\perp_{\Gamma, \underline{B}} \circ f = \perp_{\Delta, \underline{B}} \qquad h \circ \langle \Gamma, \perp_{\Gamma, \underline{B}} \rangle = \perp_{\Gamma, \underline{C}}$$

Example 4-1. Recall the collection of monads T_- for exceptions and terminal I/O from Example 3-5, and let $\varepsilon \subseteq \{\text{raise, input, output}\}$ be any effect set. If W is an ω -cpo, then $T_\varepsilon |W|$ inherits an ω -cpo structure, by inductively taking the smallest partial order satisfying:

- if $w \leq w'$ in W then $w \leq_{T_\varepsilon W} w'$;
- if $\text{input} \in \varepsilon$, and for all $c \in \text{Char}$, $t_c \leq_{T_\varepsilon W} t'_c$, then

$$\langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle \leq_{T_\varepsilon W} \langle I, \langle t'_c \rangle_{c \in \text{Char}} \rangle$$

- if $\text{output} \in \varepsilon$, $c \in \text{Char}$, and $t \leq t'$, then

$$\langle O, c, t \rangle \leq_{T_\varepsilon W} \langle O, c, t' \rangle$$

As trees, two elements of $T_{\varepsilon}W$ are comparable if they have the same shape, and corresponding leaves are compatibly comparable in W . Thus, an ω -chain of such trees is a sequence of trees with the same shape, whose corresponding leaves form ω -chains. Hence, the suprema of such an ω -chain is the same tree, with the suprema of each leaf ω -chain in its corresponding leaf.

The monadic structure of T_{ε} is ω CPO-enriched, and $\langle \omega\text{CPO}, T_{\varepsilon} \rangle$ is an ω CPO-enriched CBPV model. If W is not pointed, then clearly $T_{\varepsilon}W$ is not pointed. Therefore this model is not a recursion model. \square

Example 4-2. The *lifting monad* T_{\perp} is given by

$$\begin{aligned} |T_{\perp}W| &:= \{\perp\} + |W| & w \leq w' &\iff w = \iota_1 \perp, \text{ or} \\ T_{\perp}f &:= \{\perp\} + f & & w = \iota_2 \hat{w}, w' = \iota_2 \hat{w}', \text{ and } \hat{w} \leq_W \hat{w}' \\ \eta : w &\mapsto \iota_2 w & \mu : \iota_1 \perp &\mapsto \iota_1 \perp \\ \text{str}_{W,V} : \langle w, \iota_1 \perp \rangle &\mapsto \iota_1 \perp & \mu : \iota_2 \iota_1 \perp &\mapsto \iota_1 \perp \\ \text{str}_{W,V} : \langle w, \iota_2 w \rangle &\mapsto \iota_2 \langle w, v \rangle & \mu : \iota_2 \iota_2 w &\mapsto \iota_2 w \end{aligned}$$

Note that a monad T is strict if and only if there is a monad morphism $T_{\perp} \rightarrow T$.

The morphism map

$$T_{\perp} = \{\perp\} + - = [\iota_1, \iota_2 \circ -]$$

is continuous, hence T_{\perp} is an ω CPO-enriched strong monad. Thus $\langle \omega\text{CPO}, T_{\perp} \rangle$ is an ω CPO-enriched CBPV model.

The T_{\perp} -algebras are precisely the pointed ω -cpos. Indeed, let \underline{B} be any T_{\perp} -algebra. By fiat, $\iota_1 \perp$ is the least element of $T_{\perp} \underline{B}$. Thus, for all $b \in \|\underline{W}\|$:

$$\begin{array}{ccc} \underline{W}[-]\text{-continuity} & & \text{algebra} \\ \downarrow & & \downarrow \\ \underline{W}[\iota_1 \perp] \leq \underline{W}[\iota_2 w] = \underline{W}[\eta(w)] = w \end{array}$$

We have shown that every T_{\perp} -algebra is pointed. Therefore, $\gamma \mapsto \underline{W}[\iota_1 \perp]$ is the least element of $\omega\text{CPO}(\Gamma, \|\underline{W}\|)$. Conversely, if W is pointed with a least element \perp_W then the algebra structure over W is given by

$$\underline{W}[[w]] := \begin{cases} \perp_W & w = \iota_1 \perp \\ w' & w = \iota_2 w \end{cases}$$

If $f : \Delta \rightarrow \Gamma$, then $\perp_{\Gamma, \underline{B}}(f(\delta)) = \underline{W}[\iota_1 \perp] = \perp(\delta)$, hence $\perp \circ f = \perp$. Let h be a homomorphism from $\|\underline{V}\|$ to $\|\underline{W}\|$ over Γ . By the definition of homomorphisms over Γ , we have:

$$\begin{array}{ccccc}
\langle \gamma, \underline{V}[\iota_1 \perp] \rangle & \xrightarrow{\text{str}} & \iota_1 \perp & \xrightarrow{Th} & \iota_1 \perp \\
\downarrow \Gamma \times \underline{V}[-] & & & & \downarrow \underline{W}[-] \\
\langle \gamma, \underline{V}[\iota_1 \perp] \rangle & \xrightarrow{h} & h(\gamma, \underline{V}[\iota_1 \perp]) = \underline{W}[\iota_1 \perp] & &
\end{array}$$

Thus, $h \circ \langle \Gamma, \perp \rangle = \perp$. In conclusion, we have a recursion CBPV model. \square

In light of Lemma 4.3, we can replace the condition on homomorphisms over objects with homomorphisms in the usual sense:

Theorem 4.4. *Let $\langle \mathcal{V}, T \rangle$ be an $\omega\mathbf{CPO}$ -enriched CBPV model. It is a recursion model if and only if:*

- for all objects $\Gamma \in \text{Ob}(\mathcal{V})$ and algebras $\underline{B} \in \text{Ob}(\mathcal{V}^T)$, $\mathcal{V}(\Gamma, |\underline{B}|)$ is pointed; and
- for all $f : \Delta \rightarrow \Gamma$ and homomorphisms h from \underline{B} to \underline{C} :

$$\perp_{\Gamma, \underline{B}} \circ f = \perp_{\Delta, \underline{B}} \quad h \circ \perp_{\Gamma, \underline{B}} = \perp_{\Gamma, \underline{C}}$$

Proof

We show that the two homomorphism conditions are equivalent:

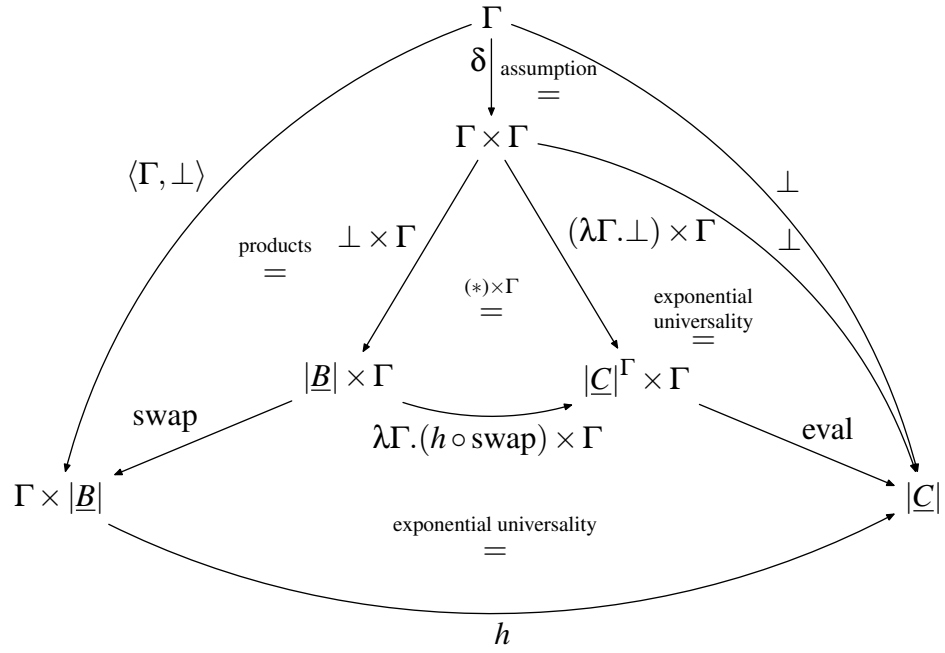
\implies Take any object $\Gamma \in \text{Ob}(\mathcal{V})$ and homomorphism h from \underline{B} to \underline{C} in the usual sense. By Lemma 4.3(1), $h \circ \pi_2$ is a homomorphism from \underline{B} to \underline{C} over Γ , hence:

$$\perp = (h \circ \pi_2) \circ \langle \Gamma, \perp \rangle = h \circ \perp$$

\impliedby Take any object $\Gamma \in \text{Ob}(\mathcal{V})$ and homomorphism h from \underline{B} to \underline{C} over Γ . As $\langle \mathcal{V}, T \rangle$ is an $\omega\mathbf{CPO}$ -enriched CBPV, the $\omega\mathbf{CPO}$ -enriched exponential $|\underline{C}|^\Gamma$ exists in \mathcal{V} . By Lemma 4.3(2), $\lambda\Gamma.(h \circ \text{swap})$ is an algebra homomorphism from \underline{B} to \underline{C}^Γ , therefore:

$$\begin{array}{c}
\text{Lemma 4.1} \\
\downarrow \\
(\lambda\Gamma.(h \circ \text{swap})) \circ \perp_{\Gamma, \underline{B}} = \perp_{\Gamma, |\underline{C}|^\Gamma} = \lambda\Gamma.\perp_{\Gamma \times \Gamma, |\underline{C}|} : \mathbb{1} \rightarrow |\underline{C}|^\Gamma \quad (*)
\end{array}$$

The commutativity of the following diagram completes the proof:



Example 4-3. We present a recursion model for exception raising computations.

Let $T_{\{\text{raise}\}}$ be the monad for exception raising from Example 4-1, and T_{\perp} the lifting monad from Example 4-4. Define $T := T_{\perp} \circ T_{\{\text{raise}\}}$. Let W be any ω CPO, then

$$|TW| = \{\perp\} + \text{Str} + |W|$$

and the ω -cpo structure is given by $x \leq y$ if and only if either:

(\perp). $x = \perp$; or

(raise). $x = y = \langle R, s \rangle$, $s \in \text{Str}$; or

(W) $x = w \in W$, $y = v \in W$ and $w \leq v$.

and suprema are given by:

$$\bigvee x_n = \begin{cases} \perp & \text{for all } n, x_n = \perp \\ \langle R, s \rangle & \text{for some } n_0, \text{ for all } n \geq n_0, x_n = \langle R, s \rangle \\ \bigvee w_n & \text{for some } n_0, \text{ for all } n \geq n_0, x_n = w_n \in W \end{cases}$$

Consider $T_{\{\text{raise}\}} \circ T_{\perp} W$. Its carrier set is also $\{\perp\} + \text{Str} + |W|$, and the ω -cpo structure is given by $x \leq y$ iff either:

(raise). $x = y = \langle R, s \rangle$, $s \in \text{Str}$; or

(\perp'). $x = \perp$, $y = w \in W$; or

(W). $x = w \in W, y = v \in W$ and $w \leq v$.

and least upper bounds given by:

$$\bigvee x_n = \begin{cases} \langle R, s \rangle & \text{for some } n_0, \text{ for all } n \geq n_0, x_n = \langle R, s \rangle, \text{ for some } n_0 \\ \perp & \text{for all } n, x_n = \perp \\ \bigvee w_n & \text{for some } n_0, \text{ for all } n \geq n_0, x_n = w_n \in W \end{cases}$$

Therefore, the map $\lambda x.x$ from $T_{\{\text{raise}\}} \circ T_{\perp} W$ to $T_{\perp} \circ T_{\{\text{raise}\}} W$ is continuous. Direct calculation shows it yields a distributive law of T_{\perp} over $T_{\{\text{raise}\}}$.

Thus, T has a strong monad structure, given by the inclusion $\eta : W \subseteq TW$ and

$$\begin{array}{llll} \mu : \perp & \mapsto \perp & \text{str} : \langle \gamma, \perp \rangle & \mapsto \perp \\ \mu : \langle R, s \rangle & \mapsto \langle R, s \rangle & \text{str} : \langle \gamma, \langle R, s \rangle \rangle & \mapsto \langle R, s \rangle \\ \mu : x & \mapsto x & \text{str} : \langle \gamma, w \rangle & \mapsto \langle \gamma, w \rangle \end{array}$$

If \underline{B} is any T -algebra, then $|\underline{B}|$ is pointed, with $\underline{B}[\perp]$ the least element. Therefore, $\omega\mathbf{CPO}(\Gamma, |\underline{B}|)$ is pointed with $\lambda\gamma.\perp$ the least element.

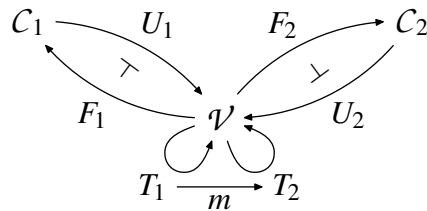
Given any function $f : \Delta \rightarrow \Gamma$ and T -algebra homomorphism $h : \underline{B} \rightarrow \underline{C}$, we have:

$$(h \circ (\lambda\gamma.\perp) \circ f)(\delta) = h(\perp) = h(\underline{B}[\perp]) \stackrel{\text{T-algebra homomorphism}}{\downarrow} = \underline{C}[(Th)(\perp)] = \underline{C}[\perp]$$

hence $h \circ \perp \circ f = \perp$, and, by Theorem 4.4, $\langle \omega\mathbf{CPO}, T \rangle$ is a recursion model. \square

Finally, we define our category of CBPV models:

Definition 2.12 ω . Let \mathcal{V} be an $\omega\mathbf{CPO}$ -enriched distributive category. The category $\omega\mathbf{CPO}\text{-Adj } \mathcal{V}$ has as **objects** CBPV models with $F \dashv U : C \rightarrow \mathcal{V}$, and as **morphisms** $(F_1 \dashv U_1) \rightarrow (F_2 \dashv U_2)$ strong monad morphisms $m : T_1 \rightarrow T_2$.



4.3 ω CPO-enriched algebraic operations



In the ω CPO-enriched case, enriched algebraic operations coincide with the ordinary ones:

Definition 2.1 ω (enriched algebraic operations). *Let $F \dashv U : \mathcal{C} \rightarrow \mathcal{V}$ be an ω CPO-enriched resolution of an ω CPO-enriched strong monad $T = UF$, and A, P be \mathcal{V} objects such that all ω CPO-enriched exponentials of objects UB by A, P exist. An ω CPO-enriched algebraic operation of type $A \langle P \rangle$ for $F \dashv U$ is an algebraic operation for the underlying ordinary resolution $F \dashv U$.*

Similarly, the definition of an enriched monad morphism mapping an enriched operation to an enriched operation coincides with its ordinary counterpart.

Example 4-4. Consider the lifting monad T_{\perp} from Example 4-4. Given any T_{\perp} -algebra B , define:

$$\begin{aligned} \perp_B : \underline{B}^0 &\rightarrow \underline{B}^1 \\ \perp_B : \star &\mapsto \lambda \star. \perp \end{aligned}$$

To see that \perp is an algebraic operation of type $\perp : \mathbb{0}$, consider any $f : \Gamma \times W \rightarrow |B|$. By Lemma 4.3(3), $f^{\dagger} : \Gamma \times T_{\perp}W \rightarrow |B|$ is a homomorphism over Γ . As $\langle \omega\text{CPO}, T_{\perp} \rangle$ is a recursion model, we deduce that $f^{\dagger}(\gamma, \perp) = \perp$ for every $\gamma \in \Gamma$. Therefore:

$$\begin{array}{ccc} \langle \gamma, \star \rangle & \xrightarrow{f^{\dagger 0}} & \star \\ \Gamma \times \perp_{FX} \downarrow & & \downarrow \perp_B \\ \langle \gamma, \lambda \star. \perp_{F_{\perp}W} \rangle & \xrightarrow{f^{\dagger 1}} & \lambda \star. f^{\dagger}(\gamma, \perp) = \perp \end{array}$$

Hence $\perp : \mathbb{0}$ is an ω CPO-enriched algebraic operation for T_{\perp} . □

The argument in the last example can be generalised to arbitrary recursion models:

Lemma 4.5. *Let $\langle \mathcal{V}, T \rangle$ be a recursion model. Then there is an algebraic operation $\text{diverge} : \mathbb{0}$ for the Eilberg-Moore resolution of T , given by:*

$$\text{diverge}_{\underline{B}} := \lambda \mathbb{1}. \perp_{|B|^0 \times \mathbb{1}} : |B|^0 \rightarrow |B|^1$$

The corresponding generic effect is then given by

$$\text{gen}_{\text{diverge}} := \perp_{\mathbb{1}, F\mathbb{0}} : \mathbb{1} \rightarrow T\mathbb{0}$$

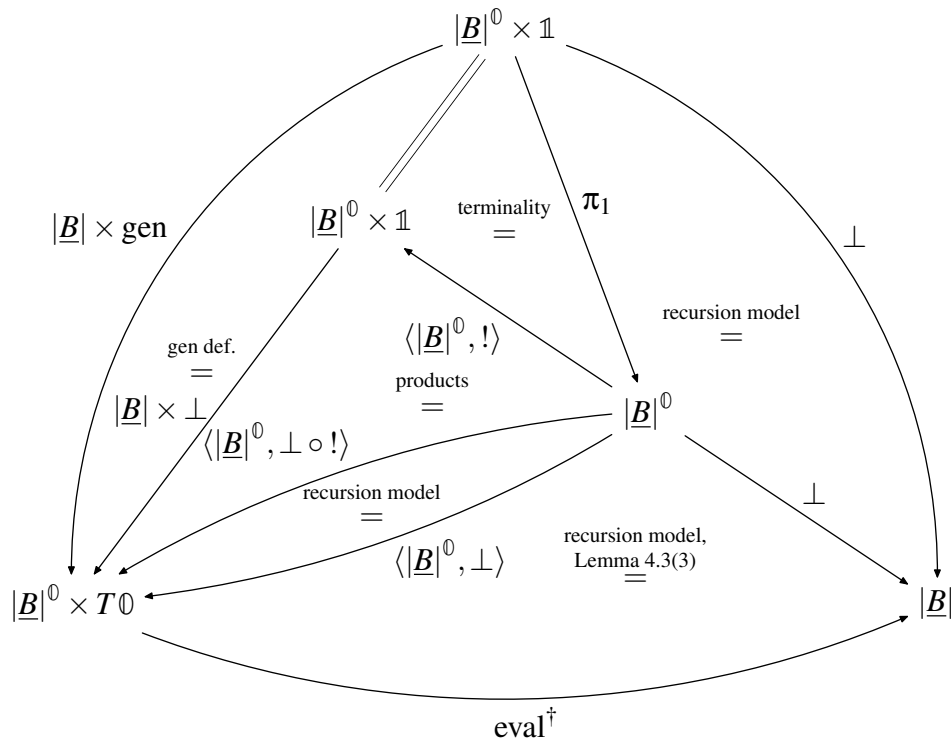
Proof

As $\langle \mathcal{V}, T \rangle$ is a recursion model, $\mathcal{V}(\mathbb{1}, T\mathbb{0})$ is pointed, hence the generic operation

$$\text{gen} := \perp_{\mathbb{1}, F\mathbb{0}} : \mathbb{1} \rightarrow T\mathbb{0}$$

is well-defined. Define $\text{diverge} := \text{op}^{\text{gen}}$ and obtain, by Theorem 2.4, an algebraic operation $\text{diverge} : \mathbb{0}$ satisfying: $\text{gen}_{\text{diverge}} = \perp$. Note that the enriched model structure guarantees diverge is an enriched operation.

Let \underline{B} be any algebra. We have:



Thus, by Theorem 2.4, $\text{diverge}_{\underline{B}} = \lambda \mathbb{1}. \perp_{|B|^0 \times \mathbb{1}}$, and we are done. \blacksquare

Example 4-5. Recall the monads for exceptions and terminal I/O from Example 4-1.

Using Theorem 2.4 we obtain enriched algebraic operations for T_{ε} :

- if $\text{raise} \in \varepsilon$:

$$\text{raise} : \mathbb{0}$$

$$\text{raise}_{\underline{B}} : \star \mapsto \lambda s. \underline{B} \llbracket \langle R, s \rangle \rrbracket$$

- if $\text{input} \in \varepsilon$:

$$\text{input} : \text{Char}$$

$$\text{input}_{\underline{B}} : \lambda c. b_c \mapsto \lambda \star. \underline{B} \llbracket \langle I, \langle b_c \rangle_{c \in \text{Char}} \rangle \rrbracket$$

- if $\text{output} \in \varepsilon$:

$$\begin{aligned} \text{output} &: \mathbb{1} \langle \text{Char} \rangle \\ \text{output}_{\underline{B}} &: \lambda \star. b \quad \mapsto \lambda c. \underline{B} \llbracket \langle O, c, b \rangle \rrbracket \end{aligned}$$

For each $\varepsilon \subseteq \varepsilon'$, the inclusions are continuous. \square

The following definition ties all the previous concepts together:

Definition 2.13 ω . Let Π be a set. An ω CPO-enriched CBPV Π -model is a quadruple

$$\langle \mathcal{V}, \text{type}, T, O \llbracket - \rrbracket \rangle$$

where:

- \mathcal{V} is an ω CPO-enriched distributive category, called the value category;
- type is a type assignment for Π in \mathcal{V} ;
- $\langle \mathcal{V}, T \rangle$ is an ω CPO-enriched CBPV model; and
- $O \llbracket - \rrbracket$ assigns to every $\text{op} : A \langle P \rangle$ in Π an ω CPO-enriched algebraic operation $O \llbracket \text{op} \rrbracket : A \langle P \rangle$ for T .

We say that an ω CPO-enriched CBPV Π -model $\langle \mathcal{V}, \text{type}, T, O \llbracket - \rrbracket \rangle$ is a recursion CBPV Π -model if Π has a distinguished effect operation diverge such that:

- $\langle \mathcal{V}, T \rangle$ is a CBPV recursion model; and
- $O \llbracket \text{diverge} \rrbracket$ is the divergence operation (see Lemma 4.5).

Example 4-6. The monads T_ε for exceptions and terminal I/O from Example 4-1 give rise to ω CPO-enriched ε -CBPV models, as $\langle \omega\text{CPO}, T_\varepsilon \rangle$ is an enriched CBPV model (see Example 4-1), and the operations are interpreted as the enriched operations from Example 4-5. This model is not a recursion model. \square

Example 4-7. The lifting monad T_\perp gives rise to a recursion $\{\text{diverge}\}$ -CBPV model, as $\langle \omega\text{CPO}, T_\perp \rangle$ is a recursion CBPV model (see Example 4-4), and we interpret the distinguished (and only) diverge operation as the divergence algebraic operation. \square

4.4 Recursion models



We are finally ready to incorporate recursion into our models for effect analysis:

Definition 2.10 ω . Let Σ be a recursion hierarchy and \mathcal{V} an $\omega\mathbf{CPO}$ -enriched distributive category. A recursion Σ -type assignment in \mathcal{V} is a Σ -type assignment in the underlying ordinary category of \mathcal{V} , such that $\text{diverge} : \mathbb{0}$, i.e., $\text{diverge} : \mathbb{0} \langle \mathbb{1} \rangle$.

Example 4-8. A recursion type assignment in $\omega\mathbf{CPO}$ for non-determinism is:

$$\text{choose} : \mathbb{2} \quad \text{diverge} : \mathbb{0} \quad \square$$

We are now ready to define recursion Σ -models:

Definition 3.2 ω . Let Σ be a recursion hierarchy. A recursion Σ -model is a quadruple

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, O_- \llbracket - \rrbracket \rangle$$

where:

- \mathcal{V} is an $\omega\mathbf{CPO}$ -enriched distributive category, called the value category;
- type is a recursion Σ -type assignment in \mathcal{V} ;
- \mathbb{P} is a functor $\mathcal{E} \rightarrow \omega\mathbf{CPO}\text{-Adj } \mathcal{V}$, called the model hierarchy, such that, for all $\varepsilon \in \mathcal{E}$ with $\text{diverge} \in \varepsilon$, $\mathbb{P}\varepsilon$ is a recursion model;
- $O_- \llbracket - \rrbracket$ assigns to each $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$, $\text{op} : A \langle P \rangle$ an (enriched) algebraic operation $O_\varepsilon \llbracket \text{op} \rrbracket$ of type $A \langle P \rangle$ for $\mathbb{P}\varepsilon$, such that, for all $\varepsilon \in \mathcal{E}$ with $\text{diverge} \in \varepsilon$, $O_\varepsilon \llbracket \text{diverge} \rrbracket$ is the divergence operation (see Lemma 4.5);

and $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ preserves the operations;

Example 4-9. We construct a domain-theoretic model for exceptions, terminal I/O, and recursion.

To keep this example simple, we choose the recursion hierarchy by setting Π to be $\{\text{raise}, \text{input}, \text{output}, \text{diverge}\}$, and

$$\mathcal{E} := \mathcal{P}(\{\text{raise}, \text{input}, \text{output}\}) \cup \mathcal{P}(\{\text{diverge}, \text{raise}\})$$

We will extend this model to the full powerset hierarchy in Section 4.5 (Example 3-5 ω).

Take Char and Str be the discrete ω -cpos whose carriers are the corresponding sets Char and Str from Example 3-5.

The recursion model $\langle \omega\mathsf{CPO}, \text{type}, \mathbb{P}, O_- \llbracket - \rrbracket \rangle$ is given as follows.

The type assignment is given by

$$\text{raise} : \mathbf{0} \langle \mathsf{Str} \rangle \quad \text{input} : \mathsf{Char} \quad \text{output} : \mathbf{1} \langle \mathsf{Char} \rangle \quad \text{diverge} : \mathbf{0}$$

Recall the enriched models $\langle \omega\mathsf{CPO}, T_- \rangle$ for exceptions and terminal I/O from Example 4-1, the recursion model $\langle \omega\mathsf{CPO}, T_{\{\text{diverge}\}} \rangle$ from Example 4-4 given by the lifting monad, and the recursion model $\langle \omega\mathsf{CPO}, T_{\{\text{raise}, \text{diverge}\}} \rangle$ for exceptions from Example 4-3. Note that, for every $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} , we have a continuous inclusion

$$m_{\varepsilon \subseteq \varepsilon'} : T_\varepsilon \subseteq T_{\varepsilon'}$$

Thus, by choosing $\mathbb{P}\varepsilon$ to be $\langle \omega\mathsf{CPO}, T_\varepsilon \rangle$ and $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ to be $m_{\varepsilon \subseteq \varepsilon'}$, we obtain a functor from \mathcal{E} to $\omega\mathsf{CPO}\text{-Adj}\omega\mathsf{CPO}$, and in the cases where $\text{diverge} \in \varepsilon$, namely $\{\text{diverge}\}$ and $\{\text{raise}, \text{diverge}\}$, we have a recursion model.

For every $\varepsilon \in \mathcal{E}$, we define:

$$\begin{aligned} \text{if raise} \in \varepsilon, \text{ define: } O_\varepsilon \llbracket \text{raise} \rrbracket_B : \star &\mapsto \lambda s. \underline{B} \llbracket \langle R, s \rangle \rrbracket \\ \text{if input} \in \varepsilon, \text{ define: } O_\varepsilon \llbracket \text{input} \rrbracket_B : \lambda c. b_c &\mapsto \lambda \star. \underline{B} \llbracket \langle I, \langle b_c \rangle_{c \in \mathsf{Char}} \rangle \rrbracket \\ \text{if output} \in \varepsilon, \text{ define: } O_\varepsilon \llbracket \text{output} \rrbracket_B : \lambda \star. b &\mapsto \lambda c. \underline{B} \llbracket \langle O, c, b \rangle \rrbracket \\ \text{if diverge} \in \varepsilon, \text{ define: } O_\varepsilon \llbracket \text{diverge} \rrbracket_B : \star &\mapsto \lambda s. \underline{B} \llbracket \perp \rrbracket \end{aligned}$$

Note how $O_\varepsilon \llbracket \text{diverge} \rrbracket$ is the operation from Lemma 4.5, that $O_\varepsilon \llbracket \text{raise} \rrbracket$ does indeed define an algebraic operation for $T_{\{\text{raise}, \text{diverge}\}}$, and that the inclusions $\mathbb{P}\varepsilon \subseteq \mathbb{P}\varepsilon'$ do indeed preserve the operations.

Thus $\langle \mathcal{V}, \text{type}, \mathbb{P}, O_- \llbracket - \rrbracket \rangle$ is a recursion Σ -model. \square

As in Section 2.3, we define models in terms of generic effects:

Definition 3.2* ω . *Let Σ be a hierarchy. A generic recursion Σ -model is a quadruple*

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_- \llbracket - \rrbracket \rangle$$

where:

- \mathcal{V} is an $\omega\mathsf{CPO}$ -enriched distributive category, called the value category;
- type is a recursion Σ -type assignment in \mathcal{V} ;

- \mathbb{P} is a functor $\mathcal{E} \rightarrow \omega\mathbf{CPO}\text{-Adj } \mathcal{V}$, called the model hierarchy, such that, for all $\varepsilon \in \mathcal{E}$ with $\text{diverge} \in \varepsilon$, $\mathbb{P}\varepsilon$ is a recursion model;
- $\mathcal{G}_-[-]$ assigns to each $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$, $\text{op} : A \langle P \rangle$ a generic effect $\mathcal{G}_\varepsilon[\text{op}]$ of type $A \langle P \rangle$ for $\mathbb{P}\varepsilon$, such that, for all $\varepsilon \in \mathcal{E}$ with $\text{diverge} \in \varepsilon$, $\mathcal{G}_\varepsilon[\text{diverge}]$ is the generic divergence effect (see Lemma 4.5);

and $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ preserves the effects.

As $\omega\mathbf{CPO}$ -enrichment for algebraic operations, generic effects, and their preservation coincides with the preservation of their ordinary counterparts, we obtain the following theorem:

Theorem 3.3 ω . Let Σ be a recursion hierarchy. Recursion Σ -models

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, O_-[-] \rangle$$

and generic recursion Σ -models

$$\langle \mathcal{V}, \text{type}, \mathbb{P}, \mathcal{G}_-[-] \rangle$$

are in bijection given via $\text{gen}_{O_\varepsilon[\text{op}]} = \mathcal{G}_\varepsilon[\text{op}]$.

Finally, recursion models are a subclass of our categorical models:

Theorem 4.6. Let $\langle \Sigma, \text{diverge} \rangle$ be a recursion hierarchy. Every recursion $\langle \Sigma, \text{diverge} \rangle$ -model can be seen as a Σ -model, by forgetting the enriched structure and non-distinguishing diverge .

4.5 Domain-theoretic models



We now specialise to the category of ω -cpos. The following definition characterises all categorical recursion models of this category:

Definition 2.10 (revisited) ω . Let Σ be a recursion hierarchy. A domain-theoretic recursion type assignment for Σ is an assignment $\text{type}(\text{op}) = \langle A, P \rangle$ of a pair of ω -cpos to each $\text{op} \in \Pi$, such that $\text{diverge} : \emptyset$.

Example 4-8 (revisited). A recursion type assignment in $\omega\mathbf{CPO}$ for non-determinism is:

$$\text{choose} : 2 \quad \text{diverge} : \emptyset \quad \square$$

Definition 3.2* (revisited) ω . Let Σ be a recursion hierarchy. A domain-theoretic recursion Σ -model is a quadruple

$$\langle \text{type}, T_-, m_-, \mathcal{G}_- \llbracket - \rrbracket \rangle$$

where:

- type is a domain-theoretic recursion type assignment for Σ ;
- T_- assigns to each ε in \mathcal{E} a locally continuous monad T_ε , such that, for all $\varepsilon \in \mathcal{E}$ with $\text{diverge} \in \varepsilon$, T_ε is strict;
- m_- assigns to each $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} a monad morphism $m_{\varepsilon \subseteq \varepsilon'} : T_\varepsilon \rightarrow T_{\varepsilon'}$;
- $\mathcal{G}_- \llbracket - \rrbracket$ assigns to each $\varepsilon \in \mathcal{E}$, and $\text{op} : A \langle P \rangle$ in ε a continuous generic effect $\mathcal{G}_\varepsilon \llbracket \text{op} \rrbracket : P \rightarrow T_\varepsilon A$, such that $\mathcal{G}_\varepsilon \llbracket \text{diverge} \rrbracket : \star \mapsto \perp$;
- m_- is functorial;

and m_- preserves the generic effects.

Example 3-5 ω . We construct a domain-theoretic model for exceptions, terminal I/O, and recursion.

We choose the hierarchy by $\Pi := \{\text{raise}, \text{input}, \text{output}, \text{diverge}\}$, and set \mathcal{E} to be $\mathcal{P}(\Pi)$.

Take Char and Str be the discrete ω -cpo whose carriers are the corresponding sets Char and Str from Example 3-5.

The recursion model $\langle \text{type}, T_-, m_-, \mathcal{G}_- \llbracket - \rrbracket \rangle$ is given as follows.

The type assignment is given by

$$\text{raise} : \mathbf{0} \langle \text{Str} \rangle \quad \text{input} : \text{Char} \quad \text{output} : \mathbf{1} \langle \text{Char} \rangle \quad \text{diverge} : \mathbf{0}$$

First, we define T_ε for $\varepsilon \subseteq \Pi \setminus \{\text{diverge}\}$. Let W be any ω -cpo, and take any $\varepsilon \subseteq \{\text{raise}, \text{input}, \text{output}\}$. We take carrier set $|TW|$ to be $|T_\varepsilon||W|$, where $|T_\varepsilon|$ is the set-theoretic monad for exceptions and terminal I/O from Example 3-5.

The set $|T_\varepsilon||W|$ inductively inherits an ω -cpo structure from W :

- if $w \leq w'$ in W then $w \leq w'$ in T_ε ;
- for all $s \in \text{Str}$, $\langle R, s \rangle \leq \langle R, s \rangle$;
- if $\text{input} \in \varepsilon$, and for all $c \in \text{Char}$, $t_c \leq t'_c$, then

$$\langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle \leq \langle I, \langle t'_c \rangle_{c \in \text{Char}} \rangle$$

- if $\text{output} \in \varepsilon$, $c \in \text{Char}$, and $t \leq t'$, then

$$\langle O, c, t \rangle \leq \langle O, c, t' \rangle$$

As trees, two elements of $T_\varepsilon W$ are comparable if they have the same shape, and corresponding leaves are compatibly comparable in W . Thus, an ω -chain of such trees is a sequence of trees with the same shape, whose corresponding leaves form ω -chains. Thus, the suprema of such an ω -chain is the same tree, with the suprema of each such ω -chain in that leaf. Inductive calculations show that the unit and multiplication of $|T_\varepsilon|$ are continuous, hence define a monad T_ε over $\omega\mathbf{CPO}$. An additional calculation shows T_ε is locally continuous.

Next, we define T_ε for ε with $\text{diverge} \in \varepsilon$. This time, we proceed co-inductively. Let W be any ω -cpo. We define $|T_\varepsilon W|$ as the largest set X such that if $t \in X$ then either:

$$(\perp) \quad t = \perp; \text{ or}$$

$$(W) \quad t \in W; \text{ or}$$

when $\text{raise} \in \varepsilon$: $(\text{raise}) \quad t = \text{raise}; \text{ or}$

when $\text{input} \in \varepsilon$: $(\text{input}) \quad t = \langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle$, where, for all $c \in \text{Char}$, $t_c \in X$; or

when $\text{output} \in \varepsilon$: $(\text{output}) \quad t = \langle O, c, t' \rangle$, where $t' \in X$.

Thus $|T_\varepsilon W|$ is the set of finite and infinite trees whose nodes are labeled by R , I , and O , depending on whether raise , input , and output are in ε , and whose leaves are either the bottom element \perp , or a W element. In co-algebraic terms, $|T_\varepsilon W|$ is the final co-algebra for the following polynomial functor:

$$\{\perp\} + |W| + \text{Str} + (-)^{\text{Char}} + \text{Char} \times (-)$$

where each of the last three summands is included depending on whether raise , input , and output are in ε .

As any final co-algebra, $|T_\varepsilon W|$ is equipped with a co-induction principle. A *bisimulation* is a binary relation \mathcal{R} over $|T_\varepsilon W|$ for which $\langle t, s \rangle \in \mathcal{R}$ implies:

$$(\perp_{\text{bis}}) \quad t = s = \perp; \text{ or}$$

$$(W_{\text{bis}}) \quad t = s = w, w \in W; \text{ or}$$

when $\text{raise} \in \varepsilon$: $(\text{raise}_{\text{bis}})$ $t = s = \langle R, s \rangle$, $s \in \text{Str}$; or

when $\text{input} \in \varepsilon$: $(\text{input}_{\text{bis}})$ $t = \langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle$, $s = \langle I, \langle s_c \rangle_{c \in \text{Char}} \rangle$, where, for all c in Char , $\langle t_c, s_c \rangle \in \mathcal{R}$; or

when $\text{output} \in \varepsilon$: $(\text{output}_{\text{bis}})$ $t = \langle O, c, t' \rangle$, $s = \langle O, c, s' \rangle$, where $\langle t', s' \rangle \in \mathcal{R}$.

The co-induction principle states that all bisimulations are subsets of the equality relation on $|T_\varepsilon W|$, i.e., if \mathcal{R} is a bisimulation, and $\langle t, s \rangle \in \mathcal{R}$ then $t = s$.

The partial order structure on $|T_\varepsilon W|$ is also defined co-inductively. We define \leq as the largest relation \mathcal{R} over $|T_\varepsilon W|$ for which $\langle t, s \rangle \in \mathcal{R}$ implies either:

$$(\perp_{\leq}) \quad t = \perp; \text{ or}$$

$$(W_{\leq}) \quad t = w, s = v, w, v \in W, w \leq v; \text{ or}$$

when $\text{raise} \in \varepsilon$: (raise_{\leq}) $t = s = \langle R, s \rangle$, $s \in \text{Str}$; or

when $\text{input} \in \varepsilon$: (input_{\leq}) $t = \langle I, \langle t_c \rangle_{c \in \text{Char}} \rangle$, $s = \langle I, \langle s_c \rangle_{c \in \text{Char}} \rangle$ where, for all $c \in \text{Char}$, $\langle t_c, s_c \rangle \in \mathcal{R}$; or

when $\text{output} \in \varepsilon$: (output_{\leq}) $t = \langle O, c, t' \rangle$, $s = \langle O, c, s' \rangle$ where $\langle t', s' \rangle \in \mathcal{R}$.

Informally, we have $t \leq s$ when s can be obtained from t by substituting the leaves w with larger leaves $w' \geq w$, and by substituting \perp with any other term. Using this informal description, the order is ω -complete: the least upper bound of an ω -chain is obtained by taking the least upper bounds on the leaves from W , and adding infinite branches where the shape of the term does not stabilise.

We establish that \leq equips $|T_\varepsilon W|$ with an ω -cpo structure more formally.

Proposition. *The relation \leq is a partial order.*

Proof

Case analyses show that the diagonal relations $\Delta := \{\langle t, t \rangle \mid t \in |T_\varepsilon W|\}$ and \leq^2 both satisfy the condition in \leq 's definition, hence $\Delta \subseteq \leq$ and $\leq^2 \subseteq \leq$, ergo \leq is reflexive and transitive. Case analysis shows that $\leq \cap \geq$ is a bisimulation, hence $(\leq \cap \geq) \subseteq =$, ergo \leq is anti-symmetric. ■

Given a poset P , denote by $\omega\text{Chains}(P)$ its set of ascending ω -chains. Note we may conduct the following case analysis on any ω -chain $\langle t_n \rangle_{n \in \mathbb{N}}$ in $\omega\text{Chains}(|T|_{\varepsilon}W)$:

(\perp_{\vee}) for all n , $t = \perp$; otherwise, there is a least n_0 for which $t_{n_0} \neq \perp$, and either:

(W_{\vee}) $t_{n_0} = w$; then, for all $n \geq n_0$, $t_n \geq w_{n_0}$, hence $t_n = w_n \in W$, and w_n is an ω -chain in W ; or

when $\text{raise} \in \varepsilon$: (raise_{\vee}) $t_{n_0} = \langle R, s \rangle$; then, as in the previous case, for all $n \geq n_0$, $t_n = \langle R, s \rangle$; or

when $\text{input} \in \varepsilon$: (input_{\vee}) $t_{n_0} = \langle I, \langle t_{n_0}^c \rangle_{c \in \text{Char}} \rangle$; then, for all $n \geq n_0$,

$$t_n = \langle I, \langle t_n^c \rangle_{c \in \text{Char}} \rangle$$

and, for all $c \in \text{Char}$, $\langle t_n^c \rangle_{n \geq n_0}$ is an ω -chain; or

when $\text{output} \in \varepsilon$: (output_{\vee}) $t_{n_0} = \langle O, c, t'_{n_0} \rangle$; then, for all $n \geq n_0$, $t_n = \langle O, c, t'_n \rangle$, and $\langle t'_n \rangle_{n \geq n_0}$ is an ω -chain.

Using this case analysis, finality of $|T_{\varepsilon}W|$ implies there exists a unique function $\vee : \omega\text{Chains}(|T_{\varepsilon}W|) \rightarrow |T_{\varepsilon}W|$ satisfying the following conditions. For all $\langle t_n \rangle_{n \in \mathbb{N}}$, using the previous notation:

(\perp_{\vee}) $\vee \langle t_n \rangle = \perp$;

(W_{\vee}) $\vee \langle t_n \rangle = \bigvee_{n \geq n_0} w_n$;

when $\text{raise} \in \varepsilon$: (raise_{\vee}) $\vee \langle t_n \rangle = \langle R, s \rangle$;

when $\text{input} \in \varepsilon$: (input_{\vee}) $\vee \langle t_n \rangle = \langle I, \langle \bigvee_{n \geq n_0} t_n^c \rangle_{c \in \text{Char}} \rangle$;

when $\text{output} \in \varepsilon$: (output_{\vee}) $\vee \langle t_n \rangle = \langle O, c, \bigvee_{n \geq n_0} t'_n \rangle$.

More explicitly, the above case analysis gives $\omega\text{Chains}(|T_{\varepsilon}W|)$ a co-algebra structure, and finality yields \vee .

Proposition. For all ω -chains $\langle t_n \rangle$ in $\omega\text{Chains}(|T_{\varepsilon}W|)$, $\bigvee_n t_n$ is the least upper bound of $\langle t_n \rangle$.

Proof

Define the following two relations:

$$\mathcal{R}^{\text{bound}} := \left\{ \langle t_m, \bigvee_n t_n \rangle \mid \langle t_n \rangle_{n \in \mathbb{N}} \in \omega\text{Chains}(|T_\varepsilon W|), m \in \mathbb{N} \right\}$$

$$\mathcal{R}^{\text{least}} := \left\{ \langle \bigvee_n t_n, s \rangle \mid \langle t_n \rangle_{n \in \mathbb{N}} \in \omega\text{Chains}(|T_\varepsilon W|), \text{ for all } n \in \mathbb{N}, s \geq t_n \right\}$$

Case analyses on $\langle t_n \rangle$ show that $\mathcal{R}^{\text{bound}}$ and $\mathcal{R}^{\text{least}}$ satisfy the condition in the definition of \leq , hence $\mathcal{R}^{\text{bound}} \subseteq \leq$, hence $\bigvee_n t_n$ is an upper bound of $\langle t_n \rangle$, but then $\mathcal{R}^{\text{least}} \subseteq \leq$ shows $\bigvee_n t_n$ is the least upper bound. \blacksquare

Thus, $T_\varepsilon W$ is an ω -cpo.

We define the morphism map structure of T_ε by mapping each $f : V \rightarrow W$ to the the unique function $T_\varepsilon f : T_\varepsilon V \rightarrow T_\varepsilon W$ satisfying:

$$(\perp_{\text{mor}}) \quad T_\varepsilon f(t) = \perp;$$

$$(W_{\text{mor}}) \quad T_\varepsilon f(t) = f(w);$$

$$\text{when raise } \in \varepsilon: (\text{raise}_{\text{mor}}) \quad T_\varepsilon f(t) = \langle R, s \rangle;$$

$$\text{when input } \in \varepsilon: (\text{input}_{\text{mor}}) \quad T_\varepsilon f(t) = \langle I, \langle T_\varepsilon f(t_c) \rangle_{c \in \text{Chor}} \rangle;$$

$$\text{when output } \in \varepsilon: (\text{output}_{\text{mor}}) \quad T_\varepsilon f(t) = \langle O, c, T_\varepsilon f(t') \rangle.$$

Proposition. T_ε is a strict functor over ωCPO .

Proof

Case analyses on t show that the following two relations are bisimulations:

$$\{ \langle t, T_\varepsilon \text{id}(t) \rangle \mid t \in |T_\varepsilon W| \}, \quad \{ \langle T_\varepsilon(f \circ g)(t), (T_\varepsilon f \circ T_\varepsilon g)(t) \rangle \mid t \in |T_\varepsilon W| \}$$

Thus, T_ε is a functor. Case analysis on t shows, by \leq 's definition, that if $f \leq g$, for any $f, g : V \rightarrow W$, then

$$\{ \langle T_\varepsilon f(t), T_\varepsilon g(t) \rangle \mid t \in |T_\varepsilon W| \} \subseteq \leq$$

Thus, T_ε is locally monotone. Note that, by the definitions of \leq and \bigvee , the maps $\langle t_c \rangle \mapsto \langle I, \langle t_c \rangle \rangle$ and $t \mapsto \langle O, c, t \rangle$ are continuous. For any ω -chain of functions $\langle f_n \rangle$, we use this continuity within further case analysis on t to show that the following relation is a bisimulation:

$$\{ \langle T_\varepsilon(\bigvee_n f_n(t)), \bigvee_n (T_\varepsilon f_n(t)) \rangle \mid t \in |T_\varepsilon W| \}$$

Thus, T_ε is a locally continuous functor.

Note T_ε is pointed, with $\perp \in T_\varepsilon W$ is the least element, and by fiat $T_\varepsilon f$ preserves \perp . Thus T_ε is strict. \blacksquare

Next, we equip T_ε with a monad structure. The unit is given by the inclusion $W \subseteq T_\varepsilon$. The definition of the multiplication map $\mu^\varepsilon(\bar{t})$ is given co-inductively, by case analysis on any $\bar{t} \in T_\varepsilon^2 W$:

$$(\perp) \quad \perp;$$

$$(W) \quad s, \text{ where } \bar{t} = s;$$

$$\text{when raise} \in \varepsilon: (\text{raise}) \quad \langle R, s \rangle;$$

$$\text{when input} \in \varepsilon: (\text{input}) \quad \langle I, \langle \mu^\varepsilon(\bar{t}_c) \rangle \rangle;$$

$$\text{when output} \in \varepsilon: (\text{output}) \quad \langle O, c, \mu^\varepsilon(\bar{t}') \rangle.$$

Note that some care is needed in the case (W), and it requires further case analysis on s to properly define the required co-algebra structure.

Straightforward calculations show that these maps are indeed continuous natural transformations that satisfy the monad laws.

To summarise, we defined the monads T_- required in the definition of domain-theoretic recursion models.

Note that if $\text{diverge} \in \varepsilon \subseteq \varepsilon'$, then, by definition, $T_\varepsilon W \subseteq T_{\varepsilon'} W$, and the inclusion is continuous. Also, if $\text{diverge} \notin \varepsilon$, then $T_\varepsilon W$ is the subset of $T_{\varepsilon \cup \{\text{diverge}\}} W$ that includes only the finite trees containing no \perp . In this case, the inclusion $T_\varepsilon W \subseteq T_{\varepsilon \cup \{\text{diverge}\}} W$ is continuous. In both cases, the inclusions in fact form monad morphisms. Therefore, for all $\varepsilon \subseteq \varepsilon' \in \mathcal{E}$, we choose $m_{\varepsilon \subseteq \varepsilon'}$ as the inclusions, and obtain a functorial family of (continuous) monad morphisms.

We choose the following generic effects, whenever they belong to the effect set ε :

$$\begin{array}{ll} \mathcal{G}_\varepsilon[\text{diverge}] : \mathbb{1} \rightarrow T_\varepsilon \mathbb{0} & \mathcal{G}_\varepsilon[\text{raise}] : \text{Str} \rightarrow T_\varepsilon \mathbb{0} \\ \mathcal{G}_\varepsilon[\text{diverge}] : \star \mapsto \perp & \mathcal{G}_\varepsilon[\text{raise}] : s \rightarrow \langle R, s \rangle \\ \\ \mathcal{G}_\varepsilon[\text{input}] : \mathbb{1} \mapsto T_\varepsilon \text{Char} & \mathcal{G}_\varepsilon[\text{output}] : \text{Char} \mapsto T_\varepsilon \mathbb{1} \\ \mathcal{G}_\varepsilon[\text{input}] : \star \mapsto \langle I, \langle c \rangle_{c \in \text{Char}} \rangle & \mathcal{G}_\varepsilon[\text{output}] : c \mapsto \langle O, c, \star \rangle \end{array}$$

As the definitions of the generic effects are uniform in ε , they are preserved by the inclusions.

In summary, $\langle \text{type}, T_-, m_-, \mathcal{G}_-[-] \rangle$ is a domain-theoretic recursion Σ -model. \square

Note that in the last example our treatment was only semi-uniform. It was uniform for all ε without diverge, and separately uniform for all ε containing diverge. However, when moving between the two cases, some care was required. Also, the two cases were different in nature: when diverge was not present, the treatment was completely inductive. With diverge present, the natural models become co-inductive. Compare this account to Example 4-1, in which our account of exceptions and divergence was also inductive. This phenomena is a well-known consequence of the limit-colimit coincidence [Sco72, PS82]. In particular, despite their co-inductive appearance, the models can still be viewed as initial algebras. The conservative restriction construction we will present in Chapter 7 construct these hierarchical models for I/O, Exception, recursion, and global state in a uniform manner.



We conclude this section by establishing that domain-theoretic recursion models indeed characterise all recursion models over $\omega\mathbf{CPO}$.

Lemma 4.7. *An $\omega\mathbf{CPO}$ -enriched CBPV model $\langle \omega\mathbf{CPO}, T \rangle$ is a recursion model if and only if T is strict.*

Proof

First, assume the model is a recursion model. For any ω -cpo W , TW is pointed, as $TW \cong \omega\mathbf{CPO}(\perp, TW)$ via $w \mapsto \lambda_{\star}.w$. Given any $f : V \rightarrow W$, $Tf : TV \rightarrow TW$ is a homomorphism, hence:

$$\perp = (\lambda_{\star}.\perp)(\star) \stackrel{\text{Theorem 4.4}}{\downarrow} = (Tf) \circ (\lambda_{\star}.\perp)(\star) = (Tf)(\perp)$$

Thus T is strict.

Conversely, assume T is strict. Let \underline{B} be any T -algebra. As T is pointed, $|T|$ has a least element \perp . For any $b \in |\underline{B}|$, we have $\perp \leq \eta(b)$. Therefore:

$$\begin{array}{ccc} \text{continuity} & & T\text{-algebra} \\ \downarrow & & \downarrow \\ \underline{B}[\perp] \leq \underline{B}[\eta(b)] & = & b \end{array}$$

Hence $|\underline{B}|$ is pointed with $\underline{B}[\perp]$ the least element. By Lemma 4.1, for all $\Gamma \in \text{Ob}(\omega\mathbf{CPO})$, $\omega\mathbf{CPO}(\Gamma, |\underline{B}|)$ is pointed, with $\lambda_{\gamma}.\perp$ the least element.

If $f : \Delta \rightarrow \Gamma$ is any function and $h : \underline{B} \rightarrow \underline{C}$ any T -algebra homomorphism, we have:

$$(h \circ (\lambda_{\gamma}.\perp) \circ f)(\delta) = h(\underline{B}[\text{bot}]) \stackrel{T\text{-algebra homomorphism}}{\downarrow} = \underline{C}[Th(\perp)] \stackrel{T \text{ is strict}}{\downarrow} = \underline{C}[\perp] = \perp$$

Hence, by Theorem 4.4, $\langle \omega\mathbf{CPO}, T \rangle$ is a recursion model. ■

Proposition 3.7 ω . *Let Σ be a recursion hierarchy. Domain-theoretic recursion Σ -models $\langle \text{type}, T_-, m_-, \mathcal{G}_-[-] \rangle$ and (categorical) recursion Σ -models $\langle \mathbf{Set}, \text{type}, \mathbb{P}, \mathcal{G}_-[-] \rangle$ are in bijection given via*

- for all $\varepsilon \in \mathcal{E}$, $\mathbb{P}\varepsilon = T_\varepsilon$; and
- for all $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} , $\mathbb{P}(\varepsilon \subseteq \varepsilon') = m_{\varepsilon \subseteq \varepsilon'}$.

Proof

From Kock's work [Koc72] follows that all locally continuous monads over $\omega\mathbf{CPO}$ have a unique continuous strength. By Lemma 4.7, the conditions on recursion models and strict monads are equivalent. The rest of the proof is analogous to that of Proposition 3.7. ■

To summarise, we defined recursion effect hierarchies, recursion models, and designated the specified divergence effect.

Chapter 5

Locally presentable categories

This presentation of my ploy

—No Doubt



This chapter is a brief interlude, in which we define *locally presentable categories*. The locally presentable categories include **Set** and $\omega\mathbf{CPO}$, our categories of interest. These categories form a central ingredient of Power's *enriched Lawvere theories*, as they ease the construction of limits and adjoint functors, which Power needs in abundance. This introduction is by no means exhaustive, we refer to Adámek and Rosický [AR94] for a full account of locally presentable categories. This chapter contains mostly background and folklore material, with the exception of Section 5.4 which contains a correction of a slight mistake in the literature, complicating our account.

In Section 5.1, we begin by defining directed colimits and their set-theoretic and domain-theoretic characterisations. Next, in Section 5.2, we define the central notion of presentable objects. We characterise them in **Set**, and provide a counterexample to Adámek and Rosický's characterisation of the countably presentable domains. In Section 5.3, we define the locally presentable categories and state their relevant properties. Finally, in Section 5.4, we characterise the countably presentable domains and show that $\omega\mathbf{CPO}$ is indeed locally countably presentable.

5.1 Directed colimits

A *regular cardinal* is an infinite cardinal λ such that, for all cardinals $\alpha < \lambda$, and α -indexed family of cardinals $\langle \lambda_i \rangle_{i < \alpha}$ satisfying $\lambda_i < \lambda$ for all $i < \alpha$, we have $\sum_{i < \alpha} \lambda_i < \lambda$.

We will be concerned with the first two regular cardinals \aleph_0 and \aleph_1 , but the definitions below apply to arbitrary regular cardinals.

Definition 5.1. A λ -directed poset is a poset I such that for every $X \subseteq I$ with $|X| < \lambda$, there exists some $i \in I$ with $i \geq x$, for all $x \in X$. In other words, every subset of I whose cardinality is less than λ has an upper bound in I .

The term ‘finitely directed’ is interchangeable with ‘ \aleph_0 -directed’. Similarly, ‘countably directed’ is interchangeable with ‘ \aleph_1 -directed’.

Example 5-1. Let X be any set. Let $\mathcal{P}_{\aleph_0}(X)$ be the collection of finite subsets of X , ordered by inclusion. Given any finite family \mathcal{F} of X -subsets, their union $\bigcup \mathcal{F}$ is again a finite X -subset. As $\bigcup \mathcal{F}$ bounds every set in \mathcal{F} , we deduce that $\mathcal{P}_{\aleph_0}(X)$ is finitely directed. \square

For the next example, we recall a few domain theoretic concepts. We write $V \subseteq W$ when V is a *subdomain* of W . That is, V consists of an ω -chain-closed subset $|V| \subseteq |W|$, with the ordering induced by W . Given any subset $X \subseteq W$, its *closure* is the smallest subdomain $\text{Cl}X \subseteq W$ containing X . The closure can also be given predicatively by transfinite recursion:

$$\begin{aligned} X_0 &:= X \\ X_{n+1} &:= X_n \cup \{\vee x_k \mid \langle x_k \rangle \text{ is an } \omega\text{-chain from } X_n\} \\ X_\lambda &:= \bigcup_{i < \lambda} X_i \end{aligned}$$

Regularity of \aleph_1 implies this sequence stabilises at step ω_1 . Indeed, if $\langle x_k \rangle$ is an ω -chain from X_{ω_1} , then for every $k < \aleph_0$, $x_k \in X_{\lambda_k}$, for $\lambda_k < \omega_1$. Regularity of \aleph_1 implies that for $\lambda := \sum_{k < \omega_0} \lambda_k$, $\lambda + 1 < \aleph_1$. But then x_k is an ω -chain in X_λ , hence $\vee x_k$ is in $X_{\lambda+1} \subseteq X_{\omega_1}$. Thus, $X_{\omega_1+1} = X_{\omega_1}$, and the process stabilises. In particular, X_{ω_1+1} is a subdomain of W containing X , and transfinite induction shows it is contained in every subdomain of W containing X . Thus, $\text{Cl}X = X_{\omega_1}$. An immediate consequence of this construction is an upper bound on the cardinality of the closure:

$$|\text{Cl}X| \leq \max\{2, |X|\}^{\aleph_0}$$

Indeed, if we denote $\kappa := \max\{2, |X|\}$, then transfinite induction establishes this bound:

$$\begin{aligned} |X_0| &= |X| \leq \kappa^{\aleph_0} \\ |X_{n+1}| &\leq |X_n| + |X_n|^{\aleph_0} \leq \kappa^{\aleph_0} + (\kappa^{\aleph_0})^{\aleph_0} \\ &= \kappa^{\aleph_0} + \kappa^{\aleph_0 \times \aleph_0} = \kappa^{\aleph_0} + \kappa^{\aleph_0} = 2 \cdot \kappa^{\aleph_0} \leq \kappa^{1+\aleph_0} = \kappa^{\aleph_0} \end{aligned}$$

and, for a limit ordinal $\lambda \leq \omega_1$:

$$\begin{array}{c} \lambda \leq \aleph_1 \leq 2^{\aleph_0} \leq \aleph^{\aleph_0} \\ \downarrow \\ |X_\lambda| \leq \sum_{i < \lambda} |X_i| = \sum_{i < \lambda} \aleph^{\aleph_0} = \lambda \cdot \aleph^{\aleph_0} \leq \aleph^{\aleph_0} \cdot \aleph^{\aleph_0} = \aleph^{\aleph_0 + \aleph_0} = \aleph^{\aleph_0} \end{array}$$

A subset $X \subseteq W$ is *dense* if and only if $\text{Cl}X = W$. A *separable* domain is one with a countable dense subset. From the upper bound of $\text{Cl}X$ we established, it follows that for every cardinal λ , there is, up to isomorphism, only a set of domains with a dense subset of cardinality λ . In particular, there is a set of separable domains, up to (continuous) isomorphism.

Example 5-2. Let W be any ωCPO . Let $\mathcal{D}_{\aleph_1}(W)$ be the set of W 's separable subdomains, ordered by inclusion, i.e.

$$\mathcal{D}_{\aleph_1}(W) := \{V \subseteq W \mid V \text{ has a dense subset } B \text{ with } |B| < \aleph_1\}$$

Given a countable collection \mathcal{F} of separable subdomains, we then have a countable collection \mathcal{B} consisting of the dense subsets of each of the countably many separable domains in \mathcal{F} . Consider the domain $V := \text{Cl} \bigcup \mathcal{B}$. The countable union $\bigcup \mathcal{B}$ of countable dense subsets is then a countable dense subset of V , hence $V \in \mathcal{D}_{\aleph_1}(W)$. As V bounds every $U \in \mathcal{F}$, we deduce that $\mathcal{D}_{\aleph_1}(W)$ is a countably directed poset. \square

Definition 5.2. Let C be any category. A λ -directed diagram in C is a diagram $D : I \rightarrow C$ with I a λ -directed poset considered as a category. A λ -directed colimit is a colimit for a λ -directed diagram.

The following lemma characterises the finitely directed colimits in **Set**.

Lemma 5.3 (see Adámek and Rosický [AR94, Exercise 1.a(2)]). Let $D : I \rightarrow \mathbf{Set}$ be a finitely directed diagram. A compatible cocone $\langle C, c \rangle$ is colimiting if and only if:

1. the cocone is collectively epi: $C = \bigcup_{i \in I} c_i[Di]$; and
2. if $c_{i'}(d') = c_i(d)$, then there exists some $j \geq i, i'$ such that

$$D(j \geq i')(d') = D(j \geq i)(d)$$

Example 5-3 (see Adámek and Rosický [AR94, Example 1.10(1)]). Every set is the finitely directed colimit of its finite subsets. Indeed, given any set X , consider the finitely directed diagram $\mathcal{P}_{\aleph_0}(X)$ of X 's finite subsets (see Example 5-1). Consider the

cocone consisting of the inclusions $n \subseteq X$ for every finite subset of X . Due to the singleton inclusions, this cocone is collectively epi. As the cocone comprises of injections and the diagram is finitely directed, condition 2 also holds. Thus, by Lemma 5.3, X is the colimit. \square

Similarly, we characterise the countably directed colimits in $\omega\mathbf{CPO}$.

Lemma 5.4. *Let $D : I \rightarrow \omega\mathbf{CPO}$ be a countably directed diagram. A compatible cocone $\langle C, c \rangle$ is colimiting if and only if:*

1. *the cocone is collectively epi: $|C| = \bigcup_{i \in I} c_i[Di]$; and*
2. *if $c_{i'}(d') \geq c_i(d)$, then there exists some $j \geq i, i'$ such that*

$$D(j \geq i')(d') \geq D(j \geq i)(d)$$

Proof

(\implies) First, assume $\langle C, c \rangle$ is colimiting. Let \mathbb{S} be the Sierpinski space $\{\perp \leq \top\}$.

Given any $d_0 \in Di_0$, define an I -indexed family of functions $e_i : Di \rightarrow \mathbb{S}$ by:

$$e_i^{d_0}(d) := \begin{cases} \perp & \text{there is some } j \geq i, i_0 \text{ such that} \\ & D(j \geq i_0)(d_0) \geq D(j \geq i)(d) \\ \top & \text{otherwise} \end{cases}$$

This function is Scott-continuous. Indeed, if $d_1 \leq d_2$ in Di , and if $e_i(d_2) = \perp$, then there is some $j \geq i, i_0$ satisfying:

$$\begin{array}{c} \text{monotonicity} \\ \downarrow \\ D(j \geq i_0)(d_0) \geq D(j \geq i)(d_2) \geq D(j \geq i)(d_1) \end{array}$$

Therefore, $e_i(d_1) = \perp$, and e_i is monotone. Next, take any ω -chain $\langle d_n \rangle$ in Di such that, for all n , $e_i(d_n) = \perp$. Therefore, for every n , we have some $j_n \geq i, i_0$ for which

$$D(j_n \geq i_0)(d_0) \geq D(j_n \geq i)(d_n)$$

As I is countably directed, there is an upper bound j for all the j_n . We then have, for every n :

$$\begin{aligned} D(j \geq i_0)(d_0) &= D(j \geq j_n) \circ D(j_n \geq i_0)(d_0) \\ &\geq D(j \geq j_n) \circ D(j_n \geq i)(d_n) = D(j_n \geq i)(d_n) \end{aligned}$$

Therefore,

$$D(j \geq i_0)(d_0) \geq \bigvee D(j \geq i)(d_n) \stackrel{\text{continuity}}{\downarrow} D(j \geq i)(\bigvee d_n)$$

Thus, $e_i(\bigvee d_n) = \perp = \bigvee c_i(d_n)$, and e_i is continuous.

Next, e forms a compatible cocone. Indeed, take any $i \leq i'$, any d in Di , and set $d' := D(i' \geq i)(d)$. If $e_i(d) = \perp$, then we have some $j \geq i, i_0$ for which:

$$D(j \geq i_0)(d_0) \geq D(j \geq i)(d)$$

As I is countably directed, we have some $j' \geq j, i'$. For this j' , we then have:

$$D(j' \geq i')(d') = D(j' \geq i)(d) = D(j' \geq j) \circ D(j \geq i)(d) \leq D(j' \geq i_0)(d_0)$$

Therefore, $e_{i'}(d') = \perp$. Conversely, if $e_{i'}(d') = \perp$, then we have some $j' \geq i', i_0$ for which:

$$D(j' \geq i_0)(d_0) \geq D(j' \geq i')(d') = D(j' \geq i)(d)$$

Therefore, $e_i(d) = \perp$. Thus, for every $i \leq i'$ we have $e_{i'} \circ D(i' \geq i) = e_i$. Thus $\langle \mathbb{S}, e \rangle$ is a compatible cocone, and we have a unique Scott-continuous function $f_{d_0} : C \rightarrow \mathbb{S}$, factoring e through c .

Given any $d \in Di$ and $d' \in Di'$ with $c_i(d) \leq c_{i'}(d')$, we then have

$$e_i^d(d) = f_{d'}(c_i(d)) \leq f_{d'}(c_{i'}(d')) = e_{i'}^{d'}(d') = \perp$$

with $j := i'$ witnessing the last equality. Therefore there exists some $j \geq i, i'$ for which:

$$D(j \geq i')(d') \geq D(j \geq i)(d)$$

and condition 2 holds.

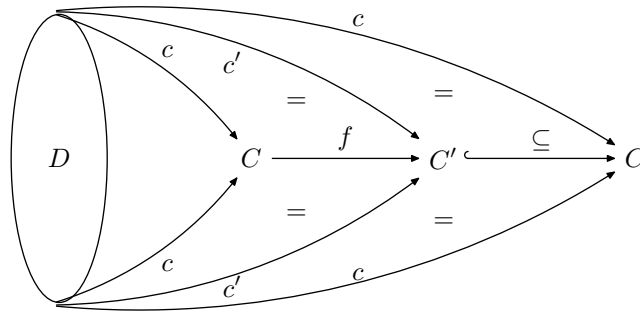
Next, let C' be the union $\bigcup_{i \in I} c_i[Di]$ inheriting the partial order from C . Consider any ω -chain $\langle c_{i_n}(d_n) \rangle$ in C' . For every n , by condition 2, we have some $j_n \geq i_n, i_{n+1}$ such that:

$$D(j_n \geq i_{n+1})(d_{n+1}) \geq D(j_n \geq i_n)(d_n)$$

Then, by considering an upper bound $j \geq j_n$, we have that $d'_n := D(j \geq j_n)(d_n)$ is an ω -chain in Dj . Set $d' = \bigvee d'_n$, and we have:

$$c_j(d') = c_j(\bigvee d'_n) = \bigvee c_j(d'_n) = \bigvee c_j(D(j \geq j_n)(d_n)) = \bigvee c_{j_n}(d_n)$$

Therefore, $\bigvee c_{j_n}(d_n)$ is also in C' , hence C' is a (sub)domain. By restricting the codomains of each c_i to $c'_i : Di \rightarrow C'$, we obtain a compatible cocone $\langle C', c' \rangle$. The couniversally induced function $f : C \rightarrow C'$ post-composed with the inclusion $C' \subseteq C$ then factors c' through c , hence $\subseteq \circ f = \text{id}$, and $C' = C$, and condition 1 follows.



(\Leftarrow) Conversely, assume conditions 1 and 2 hold. If $\langle E, e \rangle$ is any compatible cocone, define a map $f : C \rightarrow E$ by $f(w) := e_i(d)$, where i, d are any such that $c_i(d) = w$. The existence of such a function follows from condition 1. The choice of i and d does not matter: if i', d' also satisfy $c_{i'}(d') = w$, then by condition 2 and countable direction, there is some $j \geq i, i'$ such that $D(j \geq i)(d) = D(j \geq i')(d')$, hence:

$$e_i(d) = e_j(D(j \geq i)(d)) = e_j(D(j \geq i')(d')) = e_{i'}(d')$$

Note that f is Scott-continuous. Indeed, if $w' := c_{i'}(d') \geq c_i(d) =: w$, then, by condition 2, there is some $j \geq i, i'$ such that $D(j \geq i')(d') \geq D(j \geq i)(d)$. Therefore,

$$f(w') = e_j(D(j \geq i')(d')) \geq e_j(D(j \geq i)(d)) = f(w)$$

Therefore, f is monotone. Continuity follows similarly, by additional appeal to countable direction.

By definition, e factors as $c \circ f$. Consider any other function f' which factors e as $c \circ f'$. Then f' satisfies f 's definition, and we have uniqueness. Thus, $\langle C, c \rangle$ is a colimiting cocone. \blacksquare

We use the last lemma to establish directly that $\omega\mathbf{CPO}$ is countably directed co-complete:

Corollary 5.5. *All countably directed colimits exist in $\omega\mathbf{CPO}$.*

Proof

Let $D : I \rightarrow \omega\mathbf{CPO}$ be any countably directed diagram. Let $X := \sum_{i \in I} |Di|$. Define a relation on X by $\iota_i d \sqsubseteq \iota_{i'} d'$ if and only if there exists some $j \geq i, i'$ such that

$$D(j \geq i)(d) \geq D(j \geq i')(d')$$

and set the relation \equiv to be $\sqsubseteq \cap \sqsupseteq$. The relation \equiv is an equivalence relation, with transitivity following from countable directedness of I . The relation \sqsubseteq then factors through \equiv , and makes X/\equiv into a poset, which we denote by C . Given any ω -chain

$\langle w_n \rangle$ in C , by countable directedness, there is a j such that Dj contains an ω -chain $\langle d_n \rangle$ of representatives corresponding to $\langle w_n \rangle$, i.e., $w_n = [\iota_j d_n]$. Then, $\bigvee w_n = [\iota_j \bigvee d_n]$, hence C is an ω -cpo.

For every i , define $c_i(d) := [\iota_i d]$. Direct calculation shows that c_i is a Scott-continuous, collectively epi, compatible cocone. By definition, condition 2 of Lemma 5.4 holds, hence $\langle C, c \rangle$ is a colimiting cocone. ■

We also use Lemma 5.4 to present an ω -cpo as a colimit of a diagram of a subclass of its poset of subdomains $\text{Sub}W := \langle \{V \subseteq W\}, \subseteq \rangle$.

Theorem 5.6. *Let W be an ω -cpo, and $D : I \rightarrow \text{Sub}W$ a countably directed diagram. Then, in $\omega\mathbf{CPO}$, we have $\text{Colim}D = \langle \bigcup_{i \in I} Di, \subseteq \rangle$.*

Proof

Countable direction guarantees the union is an ω -cpo. The two conditions in Lemma 5.4 hold immediately. ■

As the singletons $\{w\}$ are separable, we obtain the following result:

Corollary 5.7. *For every ω -cpo W , we have $\text{Colim} \mathcal{D}_{\aleph_1}(W) = \langle W, \subseteq \rangle$.*

5.2 Presentable objects

The central notion in locally presentable categories is that of a *presentable object*.

Definition 5.8. *Let C be a category. A C -object K is λ -presentable if, for every λ -directed diagram $D : I \rightarrow C$, colimiting cocone $\langle C, c \rangle$, and morphism $f : K \rightarrow C$:*

- *there exist $i \in I$ and $g : K \rightarrow Di$ that factorises f as*

$$f : K \xrightarrow{g} Di \xrightarrow{c_i} C$$

- *and if $g' : K \rightarrow Di' \xrightarrow{c_{i'}} C$ is any other factorisation, then there exists some $j \geq i, i'$ such that $D(j \geq i) \circ g = D(j \geq i') \circ g'$*

This definition is equivalent to requiring that the representable functor induced by K , i.e., $C(K, -) : C \rightarrow \mathbf{Set}$, preserves λ -directed colimits. A λ -small colimit of a diagram whose objects are all λ -presentable objects is also λ -presentable [AR94, Proposition 1.16]. In particular, a finite coproduct of λ -presentable objects is also λ -presentable.

Example 5-4 (see Adámek and Rosický [AR94, Example 1.2(1)]). In **Set**, the finitely presentable objects are precisely the finite sets. Note that, indeed, a finite disjoint union of finite sets is a finite set. \square

Example 5-5. Every finite ω -cpo is countably presentable. Let K be any finite ω -cpo, $\langle C, c \rangle$ a colimiting cocone for a countably directed diagram $D : I \rightarrow \omega\mathbf{CPO}$, and $f : K \rightarrow C$ any continuous function.

Given $k \in K$, Lemma 5.4(1) implies there is some $d_k \in Di_k$ such that $c_{i_k}(d_k) = f(k)$. For any two $k \geq k'$, we have $f(k) \geq f(k')$. Lemma 5.4(2) then implies that there is some $i^{k,k'} \geq i_k, i_{k'}$ such that

$$D(i^{k,k'} \geq i_i)(d_k) \geq D(i^{k,k'} \geq i_{k'})(d_{k'})$$

As K is finite, the set $\{i_k, i^{k,k'} \mid k, k' \in K\}$ is finite, hence there is an upper bound i in I .

Define $g : K \rightarrow Di$ by $g(k) := D(i \geq i_k)(d_k)$. This function is Scott-continuous. For monotonicity, take any $k \geq k'$ and calculate:

$$\begin{aligned} g(k) &= D(i \geq i_k)(d_k) = D(i \geq i^{k,k'}) \circ D(i^{k,k'} \geq i_k)(d_k) \\ &\geq D(i \geq i^{k,k'}) \circ D(i^{k,k'} \geq i_{k'})(d_{k'}) = g(k') \end{aligned}$$

As g has a finite domain, monotonicity implies continuity.

By definition g factors c_i through f , as:

$$c_i \circ g(k) = c_i \circ D(i \geq i_k)(d_k) = c_{i_k}(d_k) = f(k)$$

Let $g' : K \rightarrow Di'$ be any other such factorisation. For every $k \in K$, we have:

$$c_i(g(k)) = f(k) = c_{i'}(g'(k))$$

By Lemma 5.4(2) and countable direction, there is some $j_k \geq i, i'$ such that:

$$D(j_k \geq i)(g(k)) = D(j_k \geq i')(g'(k))$$

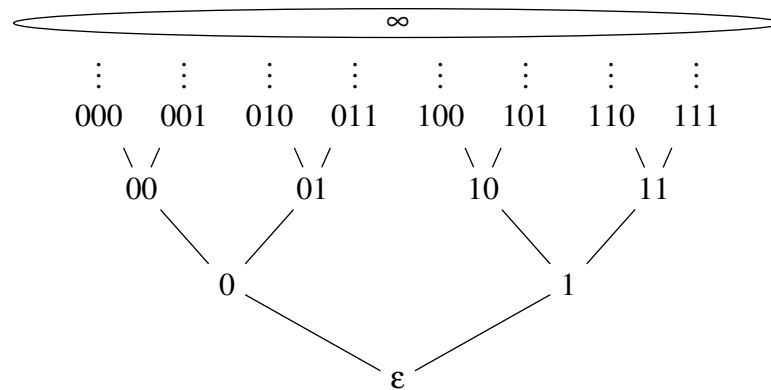
Thus there is some upper bound j for $\{j_k \mid k \in K\}$. This bound satisfies, for all $k \in K$:

$$D(j \geq i)(g(k)) = D(j \geq i')(g'(k))$$

Therefore, the factorisation is essentially unique, and we have that K is countably presentable. \square

The above argument fails if we consider countable domains instead of merely finite domains. Indeed, when the domain is infinite, it potentially has uncountably many ω -chains. The crucial step is to establish the continuity of the factorisation map g . Unfortunately, some sources in the literature incorrectly overlook this subtlety. For example, Adámek and Rosický [AR94, Example 1.14(4)] correctly state that “each finite ω -cpo is \aleph_1 -presentable”, but incorrectly state that “an infinite ω -cpo is λ -presentable iff it has cardinality smaller than λ .” Lack and Power [LP09] partially correct the mistake, and state that “the countably presentable objects of $\omega\mathbf{CPO}$ include [...] uncountable ω -cpo that have a countable presentation.” Unfortunately, they also overlooked the subtlety mentioned above¹, and state that “all countable ω -cpo [are countably presentable].”

Example 5-6. We present an ω -cpo with countably many elements that is not countably presentable. Consider the infinitely branching binary tree with a single limit point.



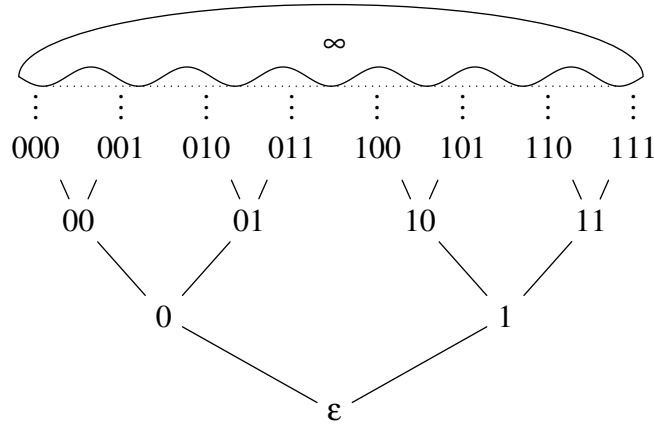
More formally, the carrier set of this domain is given by $|W| := \{0, 1\}^* + \{\infty\}$, with the order given by $w \leq w'$ if and only if $w' = \infty$, or w, w' are finite binary words and w is a prefix of w' .

Consider the following countably directed poset:

$$I := \left\{ A \subseteq \{0, 1\}^{\aleph_0} \mid A \text{ is countable} \right\}$$

For any $A \in I$, consider the infinitely branching binary tree W_A where only the sequences in A are identified with the infinity point, and the rest of the sequences remain separate:

¹John Power, private communication, 2012.



More formally, $|W_A| := \{0, 1\}^* + \{\infty\} + A^{\mathbb{C}}$, where $A^{\mathbb{C}} := \{0, 1\}^{\aleph_0} \setminus A$. The order is given by $w \leq w'$ if and only if $w' = \infty$ and w is a prefix of some $w^* \in A$, or $w' \in \{0, 1\}^* + A^{\mathbb{C}}$ and w is a prefix of w' . This W_A is an ω -cpo.

The function $W_{A \subseteq A'}$ is given by collapsing the points in $A' \setminus A$ to ∞ :

$$W_{A \subseteq A'}(w) := \begin{cases} \infty & w \in A' \\ w & \text{otherwise} \end{cases}$$

It is Scott-continuous.

We define a compatible cocone into W :

$$c_A(w) := \begin{cases} w & w \in \{0, 1\}^* \\ \infty & \text{otherwise} \end{cases}$$

This cocone is collective epi. If $c_A(w) \leq c_{A'}(w')$, then case analysis shows that for

$$\hat{A} := (\{w, w'\} \cap \{0, 1\}^{\aleph_0}) \cup A \cup A' \supseteq A, A'$$

we have $W_{\hat{A} \supseteq A'}(w') \geq W_{\hat{A} \supseteq A}(w)$. By Lemma 5.4, $\langle W, c_- \rangle$ is a colimiting cocone for the countably directed diagram W_- .

The identity $\text{id} : W \rightarrow W$ is Scott-continuous. Take any potential factorisation

$$\text{id} = W \xrightarrow{g} W_A \xrightarrow{c_A} W$$

For every w in $\{0, 1\}^*$, we have:

$$w = \text{id}(w) = c_A(g(w))$$

Therefore, by c 's definition, $g(w) = w$. As A is countable and $\{0, 1\}^{\aleph_0}$ is uncountable, there are some $w, w' \in A^{\mathbb{C}}$. Set $\langle w_n \rangle, \langle w'_n \rangle$ to be the corresponding sequences of finite prefixes. We have $\bigvee w_n = \infty = \bigvee w'_n$ in W , whereas in W_A we have:

$$\bigvee g(w_n) = \bigvee w_n = w \neq w' = \bigvee g(w'_n)$$

Thus, no factorisation of the identity is continuous. Therefore, W is not countably presentable. \square

Thus, not every countable domain is countably presentable. We will later see that the converse also holds: there are countably presentable domains with uncountably many elements. We defer the characterisation of the countably presentable domains to the end of the chapter, after we finish reviewing the relevant background on locally presentable categories.

5.3 Locally presentable categories

The main rôle of presentable objects is in relation to the following two notions (see Adámek and Rosický [AR94]):

Definition 5.9. *A category is called λ -accessible if:*

- *it has all λ -directed colimits; and*
- *it has a set \mathcal{A} of λ -presentable objects such that every object is a λ -directed colimit of objects in \mathcal{A} .*

A λ -accessible category is called locally λ -presentable if it is cocomplete.

Example 5-7 (see Adámek and Rosický [AR94, Example 1.10(1)]). The category **Set** is locally finitely presentable. Indeed, every set is a directed colimit of its finite subsets (see Example 5-3). Thus, by taking $\mathcal{A} := \aleph_0$, we deduce that **Set** is locally finitely presentable. \square

We can characterise locally presentable categories using completeness:

Theorem 5.10 (Adámek and Rosický [AR94, Corollary 2.47]). *A λ -accessible category is cocomplete if and only if it is complete. In this case, it is (by definition) locally λ -presentable.*

We summarise the required properties of locally presentable categories.

Proposition 5.11 (Adámek and Rosický [AR94, Remarks 1.9 and 1.19]). *In a locally λ -presentable category, there is, up to isomorphism, a set of λ -presentable objects.*

Theorem 5.12 (Adámek and Rosický [AR94, Theorem 1.46]). *A category is locally λ -presentable if and only if it is equivalent to a full, reflective subcategory of $\mathbf{Set}^{\mathcal{A}}$ for some small category \mathcal{A} .*

If \mathcal{A} is small, then $\mathbf{Set}^{\mathcal{A}}$ is a locally small category, and so is every subcategory thereof. As local smallness is preserved under equivalence, we deduce:

Corollary 5.13. *Every locally presentable category is locally small.*

Proposition 5.14 (Adámek and Rosický [AR94, Remark 1.56(1)]). *Every locally presentable category is well-powered, i.e., for every object A there is a set $\text{Sub}(A)$ of non-isomorphic representative subobjects $B \hookrightarrow A$.*

Definition 5.15. *Let \mathcal{B}, \mathcal{C} be accessible categories. A functor $F : \mathcal{B} \rightarrow \mathcal{C}$ is of rank λ , or λ -ary if it preserves λ -directed colimits. A monad is of rank λ if its underlying functor is of rank λ .*

Example 5-8. All monads encountered so far, i.e., the monads for modelling I/O, exceptions, and global state have a rank [HPP06]. When $\lambda \geq \aleph_0$ and $|\mathbb{V}| < \lambda$, then the global state, environment, and overwrite monads $T_{\text{GS}(\mathbb{V})}$, $T_{\text{Env}(\mathbb{V})}$, $T_{\text{OW}(\mathbb{V})}$, respectively, are λ -ranked. Similarly, when $|\text{Chor}| < \lambda$, the set theoretic monads for I/O and exceptions are λ -ranked. \square

Example 5-9. The powerset monad $\mathcal{P}(-)$ and the continuation monad X^{X^-} , for every set $|X| > 1$, have no rank.

Indeed, consider an arbitrary regular cardinal λ . Recall that a strong limit cardinal is a cardinal that cannot be reached via powersets of smaller cardinals. There exists a strong limit cardinal $\lambda' > \lambda$, for example, by using the $\beth_{\lambda+\omega}$ construction. We therefore have $\lambda' = \sup_{\mu < \lambda'} \mu$, but for all $\mu < \lambda'$, $|\mathcal{P}(\mu)| < \lambda'$. Thus:

$$\sup_{\mu < \lambda'} |\mathcal{P}(\mu)| \leq \lambda' < |\mathcal{P}(\lambda')|$$

Hence the powerset monad is not λ -ranked. A similar argument shows the continuation monad has no rank. \square

Theorem 5.16 ([AR94, Corollary 2.45]). *Let \mathcal{B}, \mathcal{C} be λ -accessible categories. Every λ -ranked functor $F : \mathcal{B} \rightarrow \mathcal{C}$ satisfies the solution set condition.*

Theorem 5.17 ([AR94, Proposition 2.23]). *A left or right adjoint between λ -accessible categories is λ -ranked.*

Corollary 5.18. *A λ -ranked functor between λ -accessible categories has a left adjoint if and only if it is continuous.*

Theorem 5.19 ([AR94, Section 2.H, final remark]). *Let T be a λ -ranked monad over a locally λ -presentable category \mathcal{C} . Then the Eilenberg-Moore category \mathcal{C}^T is locally λ -presentable.*

Every λ -ranked functor is determined, up to a natural isomorphism, by its behaviour over the λ -presentable objects and the morphisms between them [AR94, Remark 2.18(1)]. In what follows, we will require the construction more explicitly, therefore we briefly spell it out. Recall the functor category $\mathcal{C}^{\mathbb{S}} = \mathcal{C} \downarrow \mathcal{C}$, where \mathbb{S} is the category of two objects and a single non-trivial arrow between them, i.e., the Sierpinski space considered as a category.

Proposition 5.20. [AR94, Example 1.55(1) and Exercise 2.c(1)] *Let \mathcal{C} be a category.*

- *The λ -presentable objects of $\mathcal{C}^{\mathbb{S}}$ are precisely the arrows $K \xrightarrow{f} K'$ between λ -presentable objects*
- *If \mathcal{C} is λ -accessible, then so is $\mathcal{C}^{\mathbb{S}}$.*
- *If \mathcal{C} is locally λ -presentable, then so is $\mathcal{C}^{\mathbb{S}}$.*

Recall that colimits (and limits) in $\mathcal{C}^{\mathbb{S}}$ are given *pointwise* (see, for example, Mac Lane [ML98, Section V.3]). Explicitly, consider a diagram $D : J \rightarrow \mathcal{C}^{\mathbb{S}}$. For every $j \in \text{Ob}(J)$, denote $Dj : D^0 j \xrightarrow{f_j} D^1 j$. A colimiting cocone for this diagram is determined uniquely by two colimiting cocones $\langle C^0, c^0 \rangle, \langle C^1, c^1 \rangle$. The arrow in this colimiting cocone is the unique arrow $f : C^0 \rightarrow C^1$ satisfying, for every $j \in \text{Ob}(D)$:

$$\begin{array}{ccc}
 D^0 j & \xrightarrow{c_j^0} & C^0 \\
 \downarrow f_j & = & \downarrow f \\
 D^0 j & \xrightarrow{c_j^1} & C^1
 \end{array}$$

We can now describe how a λ -ranked functor is determined by its behaviour over the λ -presentable objects and arrows:

Proposition 5.21. *Let $F : \mathcal{B} \rightarrow \mathcal{C}$ be a λ -ranked functor. For every \mathcal{B} -arrow $f : A \rightarrow B$ and λ -directed diagram $D : I \rightarrow \mathcal{B}^{\mathbb{S}}$, if $f = \text{Colim} D$ then $Ff = \text{Colim} F^{\mathbb{S}} \circ D$.*

Recall that the functor $F^{\mathcal{S}} : \mathcal{B}^{\mathcal{S}} \rightarrow \mathcal{C}^{\mathcal{S}}$ acts by functor post-composition, i.e., by applying F componentwise: given a $\mathcal{B}^{\mathcal{S}}$ -object $f : A \rightarrow B$, we have

$$F^{\mathcal{S}} \langle A, B, f \rangle = \langle FA, FB, Ff \rangle$$

and given a $\mathcal{B}^{\mathcal{S}}$ -morphism $\langle g : A \rightarrow B, g' : A' \rightarrow B' \rangle$,

$$F^{\mathcal{S}} \langle g, g' \rangle = \langle Fg, Fg' \rangle$$

Note that, as \mathcal{C} is λ -accessible, so is $\mathcal{C}^{\mathcal{S}}$, hence there always exists a λ -directed diagram D , whose vertices are all λ -presentable objects, satisfying $f = \text{Colim} D$. The proposition states that any such diagram can be used to calculate Ff .

Proof

Consider any colimiting cocone for D with f as vertex. Then, for all $i \in I$, we have the following diagram:

$$\begin{array}{ccc} FD^0 j & \xrightarrow{Fc_i^0} & FA \\ \downarrow Ff_i & = & \downarrow Ff \\ FD^0 j & \xrightarrow{Fc_i^1} & FB \end{array}$$

Because F preserves the components D^0, D^1 , the top and bottom arrows are the components of the colimiting cocones for $F \circ D^0$ and $F \circ D^1$. Thus Ff is the colimit of $F^{\mathcal{S}} \circ D$. ■

Similarly, natural transformations between λ -ranked functors are uniquely determined by their behaviour over the λ -presentable objects:

Proposition 5.22. *Let $F, G : \mathcal{B} \rightarrow \mathcal{C}$ be λ -ranked functors. Given a natural transformation $m : F \rightarrow G$, and a diagram $D : I \rightarrow \mathcal{C}$, denote by $D_m : I \rightarrow \mathcal{C}^{\mathcal{S}}$ the diagram given by*

$$\begin{aligned} D_m i &: FDi \xrightarrow{m_{Di}} GDi & i \in \text{Ob}(I) \\ D_m f &= \left\langle FDi \xrightarrow{FDf} FDj, GDi \xrightarrow{GDf} GDj \right\rangle & f : i \rightarrow j \text{ in } I \end{aligned}$$

If D is a λ -directed diagram such that $\text{Colim} D = A$, then $m_A = \text{Colim} D_m$.

Proof

The naturality of m implies that D_m is indeed a well-defined functor into $\mathcal{C}^{\mathcal{S}}$.

Let $\langle A, c \rangle$ be a colimiting cocone for D . The components of D_m are $F \circ D$ and $G \circ D$. As F and G are accessible, we have $\langle FA, Fc \rangle$, $\langle GA, Gc \rangle$ as their colimiting cocones, respectively. Thus, they form a colimiting cocone in $\mathcal{C}^{\mathcal{S}}$. For every $i \in I$, we have

$$\begin{array}{ccc}
 FDj & \xrightarrow{Fc_i} & FA \\
 m_{Di} \downarrow & m \text{ naturality} & \downarrow m_A \\
 & = & \\
 GDj & \xrightarrow{Gc_i} & FC^1
 \end{array}$$

Thus $m_A = \text{Colim } D_m$. ■

5.4 Countably presentable domains

We return to characterise the countably presentable domains and establish the local presentability of $\omega\mathbf{CPO}$. The following syntactic characterisation of the countably presentable domains is based on a technique from Bridge's thesis [Bri12]. Bridge uses this technique to present a direct proof for the equivalence between essentially algebraic categories and locally presentable categories [AR94, Theorem 3.36]. We will not present this technique, and refer interested readers to the thesis [Bri12, Section 2.2].

Definition 5.23. A domain presentation \mathcal{P} consist of a triple $\langle B, I, \sqsubseteq \rangle$ where:

- B is a poset, which we call the basis;
- I is a countable set of ω -chains over B , whose elements we call constrained chains; and
- \sqsubseteq is a relation over I , whose elements we call constraints.

We say that a domain presentation is countable when the basis B and the set of constrained chains I are countable.

Plotkin, in unpublished work, uses a different form of constraints that relate a single element of the poset with an ω -chain, $w \sqsubseteq \langle v_n \rangle$.

Let $\mathcal{P} = \langle B, I, \sqsubseteq \rangle$ be a domain presentation, and W a domain. We say that a monotone function $f : B \rightarrow W$ satisfies the constraint $\langle b_n \rangle \sqsubseteq \langle b'_n \rangle$ when

$$\bigvee f(b_n) \leq \bigvee f(b'_n)$$

Definition 5.24. Let $\mathcal{P} = \langle B, I, \sqsubseteq \rangle$ be a domain presentation. A \mathcal{P} -model W is a pair $\langle |W|, W[-] \rangle$ consisting of an ω -cpo $|W|$ and a monotone function $W[-] : B \rightarrow |W|$ satisfying all the constraints in \sqsubseteq .

A \mathcal{P} -homomorphism h from W to W' is a Scott-continuous function $h : |W| \rightarrow |W'|$ factoring $W'[-]$ as

$$\begin{array}{ccc} |W| & & \\ \uparrow W[-] & \searrow h & \\ B & \xrightarrow{W'[-]} & |W'| \end{array}$$

We denote by $\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$ the category of \mathcal{P} -models and \mathcal{P} -homomorphisms between them.

Using this terminology, we present our central class of domains:

Definition 5.25. We say an ω -cpo K has a countable presentation if there exists a countable domain presentation \mathcal{P} such that K is the carrier for an initial \mathcal{P} -model.

Example 5-10. Every finite domain W has a countable presentation, with $B := W$, $I := \emptyset$, and $W[-] := \text{id}$. As W is finite, any monotone map is also continuous, hence W is the initial model. \square

More generally, every ω -continuous ω -cpo W has a countable presentation. Recall that a domain W is called ω -continuous if there exists a countable set B , such that for every w in W there exists an ω -chain $\langle b_n \rangle$ in B such that: $\bigvee b_n = w$, and for any ω -chain satisfying $w \leq \bigvee w_m$, for every n there exists an m for which $b_n \leq w_m$.

For example, any finite poset is an ω -continuous domain, as well as any closed interval $[a, b]$ with the arithmetic ordering on real numbers.

We can understand the ω -continuous domains categorically as follows. Consider the forgetful functor $|-| : \omega\mathbf{CPO} \rightarrow \mathbf{Pos}$. This functor has a left adjoint $F \dashv |-|$ (for example, by appeal to Freyd's adjoint functor theorem). The ω -continuous domains are precisely the free objects FB for a countable poset B . The unit $\eta : B \rightarrow |FB|$ induces the countable subset $\eta[B] \subseteq FB$. We sketch the proof as follows.

Consider any continuous poset B . Routine calculation shows that $\text{Cl}\eta[B] = FB$. An additional routine calculation using the Sierpinski space shows that if $\bigvee \eta(b_n) \leq \bigvee \eta(b'_m)$ then for every n there exists an m such that $\eta(b_n) \leq \eta(b'_m)$. This implies that if we have an ω -chain $\langle \bigvee_m \eta(b_{n,m}) \rangle_n$ we can construct an ω -chain $\eta(b_{k,m_k})$ such that $\bigvee_n \bigvee_m \eta(b_{n,m}) = \bigvee_k \eta(b_{k,m_k})$. Therefore, the iterative construction of $\text{Cl}\eta[B]$ stabilises

in the first step:

$$(\eta[B])_2 = \left\{ \bigvee_n \bigvee_m \eta(b_{m,n}) \mid b_{m,n} \in B \right\} = \left\{ \bigvee_k \eta(a_k) \mid a_k \in B \right\} = (\eta[B])_1$$

Consequently, $FB = \text{Cl}\eta[B] = \{\bigvee_k \eta(a_k) \mid a_k \in B\}$. Therefore, for every element w in FB there is a chain ηb_n in $\eta[B]$ such that $w = \bigvee b_n$, and if $w \leq \bigvee_k \bigvee_m b'_{k,m}$, then for every n there exist k, m for which $b_n \leq b_{k,m}$. Thus FB is ω -continuous.

Conversely, consider any ω -continuous domain W , and let B be the countable subset exhibiting it as an ω -continuous domain, considered as a poset. If V is any other domain and $f : B \rightarrow |V|$ any monotone function, extend f to a continuous function $f^\dagger : W \rightarrow V$ by setting

$$f^\dagger : \bigvee b_n \mapsto \bigvee f(b_n)$$

The definition of ω -continuous domains implies f^\dagger is independent of the choice of ω -chain $\langle b_n \rangle$, monotone, and continuous. By fiat, f factors as $f^\dagger \circ \subseteq$, and any other such factorisation $f = g \circ \subseteq$ implies that $g = f^\dagger$. Therefore, $W \cong FB$. Another way to show that $W \cong FB$ is to use a *rounded ideal completion* argument² (see, e.g., Keimel [Kei10] for the definition of rounded ideal completion).

In summary:

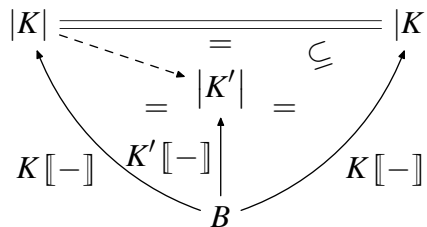
Example 5-11 (Hyland et. al [HPP06, Section 2]). All ω -continuous domains have a countable presentation. Given any FB , the required presentation is $\langle B, \emptyset, \emptyset \rangle$, and the initial model is $\langle FB, \eta \rangle$. □

Initiality implies that all domains with a countable presentation are separable:

Lemma 5.26. *If K has a countable presentation $\mathcal{P} = \langle B, I, \subseteq \rangle$, then $K[-][B]$, the image of B under the map $K[-]$, is dense in $|K|$. Consequently, domains with a countable presentation are separable.*

Proof

Set $B' := K[-][B]$. As B is countable, so is B' . Consider the model K' given by $\text{Cl}B'$ with the restriction of $K[-]$ to a function from B to $\text{Cl}B'$, and obtain by initiality that



Consequently, $\text{Cl}B' = |K'| = |K|$. ■

²Gordon Plotkin, private communication, 2012.

We can always construct a couniversal pair for any domain presentation:

Lemma 5.27. *Every (not necessarily countable) domain presentation has an initial model.*

Proof

Using the forgetful functor $|-| : \omega\mathbf{CPO} \rightarrow \mathbf{Pos}$, note that $\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$ is the full subcategory of the comma category $B \downarrow |-|$, whose objects consist of all the \mathcal{P} -models. We use Freyd's existence theorem [ML98, Theorem V.6.1] to establish that the category $\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$ has an initial object.

Having a left adjoint, $|-|$ is continuous, hence $B \downarrow |-|$ is complete, and the projection functor from $B \downarrow |-|$ to $\omega\mathbf{CPO}$ is continuous.

Let $D : J \rightarrow \mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$ be any small diagram and $\langle W, f \rangle$ its limit as a diagram in $B \downarrow |-|$, with a limiting cone c . The projection W is then a limit of the projected diagram $|D| : J \rightarrow \omega\mathbf{CPO}$ with c limiting cone again. For every constraint $\langle b_n \rangle \sqsubseteq \langle b'_n \rangle$, and for every $j \in J$, we know that Dj satisfies the constraint, hence:

$$c_j(\bigvee f(b_n)) = \bigvee c_j \circ f(b_n) = \bigvee Dj \llbracket b_n \rrbracket \leq \bigvee Dj \llbracket b'_n \rrbracket = c_j(\bigvee f(b'_n))$$

This inequation holds for every $j \in J$, hence from basic properties of limits in $\omega\mathbf{CPO}$ (see Section 4.1), $\bigvee f(b_n) \leq \bigvee f(b'_n)$ in W , thus f satisfies the constraint $\langle b_n \rangle \sqsubseteq \langle b'_n \rangle$. Hence, f satisfies all the constraints, and $\langle W, f \rangle$ is a \mathcal{P} -model. Therefore, the subcategory $\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$ is closed under limits in $B \downarrow |-|$, and thus complete. It is also locally small.

For the solution set condition, let $\mathcal{P} = \langle B, I, \sqsubseteq \rangle$, and take I to be the following set:

$$\left\{ W \in \text{Ob}(\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})) \mid |W| \text{ is a cardinal less than or equal to } \min\{2, |B|\}^{\aleph_0} \right\}$$

For any \mathcal{P} -model W in $\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$, consider the \mathcal{P} -model W' induced by the subdomain $\text{Cl}(W \llbracket - \rrbracket [B])$ and restricting $W \llbracket - \rrbracket$'s codomain. As we noted in Section 5.1, $\text{Cl}(W \llbracket - \rrbracket [B])$ can have at most $\min\{2, |B|\}^{\aleph_0}$ elements, hence it is isomorphic to some $W'' \in I$. Therefore, we have the following \mathcal{P} -homomorphism:

$$\begin{array}{ccccc} W'' & \cong & W' & \subseteq & W \\ & \searrow & \uparrow & \searrow & \nearrow \\ & & B & & \end{array}$$

Thus, the solution set condition is satisfied, hence $\mathbf{Mod}(\mathcal{P}, \omega\mathbf{CPO})$ has an initial object, as required. ■

All domains are countably directed colimits of domains with a countable presentation:

Lemma 5.28. *Let W be any domain.*

1. *The following set is countably directed*

$$I := \{ \langle B, I' \rangle \mid B \subseteq W \text{ a subposet, } I' \subseteq \omega\text{Chains}(B), \text{ and } B, I' \text{ countable} \}$$

where the order is given componentwise by inclusion. Moreover, if, for every $\langle B, I \rangle$, we define $\langle b_n \rangle \sqsubseteq \langle b'_n \rangle$ if and only if $\bigvee b_n \leq \bigvee b'_n$, then $\mathcal{P}_{\langle B, I \rangle} := \langle B, I, \sqsubseteq \rangle$ is a domain presentation.

2. *There is a countably directed diagram $W_- : I \rightarrow \omega\mathbf{CPO}$ where $W_{\langle B, I \rangle}$ is the initial $\mathcal{P}_{\langle B, I \rangle}$ -model, and the morphism map $W_{\langle B, I \rangle \leq \langle B', I' \rangle}$ is the unique $\mathcal{P}_{\langle B, I \rangle}$ -homomorphism satisfying:*

$$\begin{array}{ccc} & W_{\langle B, I \rangle} & \\ & \uparrow & \searrow \\ W_{\langle B, I \rangle} \llbracket - \rrbracket & & W_{\langle B', I' \rangle} \\ & B \subseteq B' & \xrightarrow{W_{\langle B', I' \rangle} \llbracket - \rrbracket} \\ & & \end{array} \quad \begin{array}{c} \text{---} \\ \text{=} \\ \text{---} \end{array} \quad \begin{array}{c} W_{\langle B, I \rangle \leq \langle B', I' \rangle} \\ \text{---} \\ \text{=} \\ \text{---} \end{array}$$

3. *There is a colimiting cocone $\langle W, c \rangle$ induced by the inclusions:*

$$\begin{array}{ccc} & W_{\langle B, I \rangle} & \\ & \uparrow & \searrow \\ W_{\langle B, I \rangle} \llbracket - \rrbracket & & W \\ & B \subseteq & \xrightarrow{c_{\langle B, I \rangle}} \end{array} \quad \begin{array}{c} \text{---} \\ \text{=} \\ \text{---} \end{array}$$

Proof

Given a countable collection $\langle \langle B_n, I_n \rangle \rangle$ in I , the union $\langle \bigcup B_n, \bigcup I_n \rangle$ is also in I . Thus I is countably directed, and we proved 1.

Next, by Lemma 5.27 we can choose an initial $\mathcal{P}_{\langle B, I \rangle}$ -model $W_{\langle B, I \rangle}$ for each $\langle B, I \rangle$ in I . Thus, the object map $W_{\langle B, I \rangle}$ is well-defined. By our choice of the corresponding \sqsubseteq , $W_{\langle B', I' \rangle}$ satisfies all the constraints in \sqsubseteq . Therefore, by initiality, there exists a unique $\mathcal{P}_{\langle B, I \rangle}$ -homomorphism $W_{\langle B, I \rangle \leq \langle B', I' \rangle}$ satisfying

$$\begin{array}{ccc} & W_{\langle B, I \rangle} & \\ & \uparrow & \searrow \\ W_{\langle B, I \rangle} \llbracket - \rrbracket & & W_{\langle B', I' \rangle} \\ & B \subseteq B' & \xrightarrow{W_{\langle B', I' \rangle} \llbracket - \rrbracket} \\ & & \end{array} \quad \begin{array}{c} \text{---} \\ \text{=} \\ \text{---} \end{array} \quad \begin{array}{c} W_{\langle B, I \rangle \leq \langle B', I' \rangle} \\ \text{---} \\ \text{=} \\ \text{---} \end{array}$$

Routine arguments show that $W_- : I \rightarrow \omega\mathbf{CPO}$ is a functor, and we proved 2.

Compatibility of the cocone $\langle W, c \rangle$ follows by routine initiality arguments. Given any $w \in W$, take $\langle \{w\}, \emptyset \rangle \in I$, and calculate:

$$\begin{array}{ccc} & W_{\langle B, I \rangle} \llbracket w \rrbracket & \\ & \uparrow & \swarrow c_{\langle B, I \rangle} \\ W_{\langle B, I \rangle} \llbracket - \rrbracket & \uparrow = & \\ w & \xrightarrow{\subseteq} & w \end{array}$$

Therefore c is collectively epi.

Next, take any i_1, i_2 in I , and $w_1 \in W_{i_1}$, $w_2 \in W_{i_2}$, such that

$$c_{i_1}(w_1) \leq c_{i_2}(w_2)$$

Denote $i_1 = \langle B_1, I_2 \rangle$, and $i_2 = \langle B_2, I_2 \rangle$.

By Lemma 5.26, $W_{i_1} \llbracket - \rrbracket \llbracket B_1 \rrbracket$ is dense in W_{i_1} , hence there exists a sequence $\langle b_n^1 \rangle$ over B_1 such that $\langle W_{i_1} \llbracket b_n^1 \rrbracket \rangle$ is an ω -chain, and $\bigvee W_{i_1} \llbracket b_n^1 \rrbracket = w_1$. But then

$$\langle c_{i_1}(W_{i_1} \llbracket b_n^1 \rrbracket) \rangle = \langle b_n^1 \rangle$$

is an ω -chain in W , hence $\langle b_n^1 \rangle$ is an ω -chain in B_1 . Similarly, there exists an ω -chain $\langle b_n^2 \rangle$ in B_2 such that $\bigvee W_{i_2} \llbracket b_n^2 \rrbracket = w_2$.

Choose

$$j := \langle B_1 \cup B_2, I_1 \cup I_2 \cup \{ \langle b_n^1 \rangle, \langle b_n^2 \rangle \} \rangle \in I$$

We have that $j \geq i_1, i_2$, and moreover, that $\langle b_n^1 \rangle \sqsubseteq_j \langle b_n^2 \rangle$, as:

$$\bigvee b_n^1 = c_{i_1}(\bigvee W_{i_1} \llbracket b_n^1 \rrbracket) = c_{i_1}(w_1) \leq c_{i_2}(w_2) = \bigvee b_n^2$$

But W_j satisfies the constraint $\langle b_n^1 \rangle \sqsubseteq_j \langle b_n^2 \rangle$, hence:

$$W_{j \geq i_1}(w_1) = \bigvee W_{i_1} \llbracket b_n^1 \rrbracket \leq \bigvee W_{i_2} \llbracket b_n^2 \rrbracket = W_{j \geq i_2}(w_2)$$

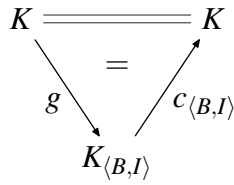
Therefore, by Lemma 5.4, $\langle W, c \rangle = \text{Colim } W_-$, and we proved 3. ■

We next establish our characterisation of the countably presentable domains:

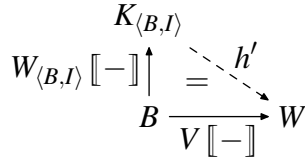
Proposition 5.29. *Every countably presentable domain has a countable presentation.*

Proof

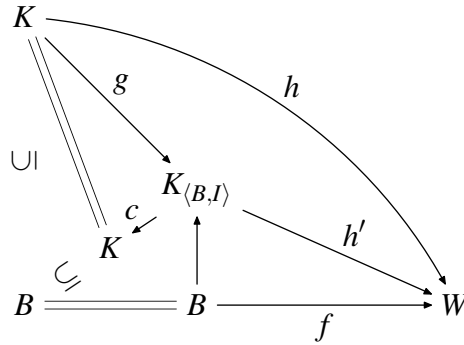
Let K be any countably presentable domain. By Lemma 5.28, we present K as a colimit $\langle K, c \rangle = \text{Colim } W_-$. As K is countably presentable, we can factorise the identity over K :



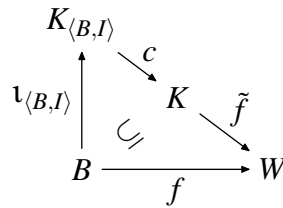
By definition, K with the inclusion $B \subseteq K$ is a $\mathcal{P}_{\langle B, I \rangle}$ -model. Given any other model V satisfying the constraints, initiality yields the following diagram:



Take $h : K \xrightarrow{g} K_{\langle B, I \rangle} \xrightarrow{h'} W$, then the following diagram commutes:



Given any other homomorphism $h'' : K \rightarrow W$, pre-composition with c yields the following commuting diagram



By initiality, we deduce that $h'' \circ c = h'$. Pre-composing with g yields:

$$h'' = h' \circ g = h$$

Therefore, h is unique, and $\langle K, \subseteq \rangle$ is initial. ■

Proposition 5.30. *A domain that has a countable presentation is countably presentable.*

Proof

Let K be any domain that has a countable presentation $\mathcal{P} = \langle B, I, \sqsubseteq \rangle$. Let $\langle C, c \rangle$ be any colimiting cocone of a countably directed diagram $D : I \rightarrow \omega\mathbf{CPO}$, and $f : K \rightarrow C$ any continuous function.

Take $B' := f \circ K \llbracket - \rrbracket \llbracket B \rrbracket \subseteq C$. By Lemma 5.4, c is collectively epi, hence for every $b' \in B'$ there exists some $i_{b'} \in I$ and $d_{b'} \in Di_{b'}$ such that $c_{i_{b'}}(d_{b'}) = b'$. By the same Lemma, for all $b' \leq b''$ in B' there exists some $i'_{b' \leq b''} \geq i_{b'}, i_{b''}$ such that:

$$D(i'_{b' \leq b''} \geq i_{b''})(d_{b''}) \geq D(i'_{b' \leq b''} \geq i_{b'})(d_{b'})$$

Note that, as B' is countable, $\left\{ i_{b'}, i'_{b' \leq b''} \mid b', b'' \in B' \right\}$ is also countable. As I is countably directed, there exists some upper bound $i \in I$.

Set $\tilde{f} : |B| \rightarrow |Di|$ by $\tilde{f}(b) := D(i \geq i_{f \circ K \llbracket b \rrbracket})(d_{f \circ K \llbracket b \rrbracket})$. Note that, for any $b \in B$, if we set $b' := f(K \llbracket b \rrbracket)$, we have:

$$c_i \circ \tilde{f}(b) = c_i \circ D(i \geq i_{b'})(d_{b'}) = c_{i_{b'}}(d_{b'}) = b' = f(K \llbracket b \rrbracket)$$

Therefore, $c_j \circ \tilde{f} = f \circ K \llbracket - \rrbracket$. Also note that, from i 's choice, \tilde{f} is monotone. Indeed, for any $b_1 \leq b_2$, set $b'_i := f(K \llbracket b_i \rrbracket)$, and calculate:

$$\begin{aligned} \tilde{f}(b_2) &= D(i \geq i_{b'_2})(d_{b'_2}) = D(i \geq i'_{b'_2 \geq b'_1}) \circ D(i'_{b'_2 \geq b'_1} \geq i_{b'_2})(d_{b'_2}) \\ &\geq D(i \geq i'_{b'_2 \geq b'_1}) \circ D(i'_{b'_2 \geq b'_1} \geq i_{b'_1})(d_{b'_1}) = \tilde{f}(b_1) \end{aligned}$$

For any constraint $\langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle$, we have:

$$c_i(\bigvee \tilde{f}(b_n^1)) = \bigvee c_i \circ \tilde{f}(b_n^1) = \bigvee f(K \llbracket b_n^1 \rrbracket) \leq \bigvee f(K \llbracket b_n^2 \rrbracket) = c_i(\bigvee \tilde{f}(b_n^2))$$

K satisfies all constraints
and f is monotone
 \downarrow

Therefore, again by Lemma 5.4, there exists some $j_{\langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle}$ in I such that:

$$D(j_{\langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle} \geq i)(\bigvee \tilde{f}(b_n^2)) \geq D(j_{\langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle} \geq i)(\bigvee \tilde{f}(b_n^1))$$

As \sqsubseteq is countable, there is some j in I bounding $\{i\} \cup \left\{ j_{\langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle} \mid \langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle \right\}$. Define \check{f} as the post-composition

$$\check{f} : B \xrightarrow{\tilde{f}} Di \xrightarrow{D(i \leq j)} Dj$$

This function also satisfies $c_j \circ \check{f} = f \circ K \llbracket - \rrbracket$, is also monotone, and, moreover, satisfies all constraints $\langle b_n^1 \rangle \sqsubseteq \langle b_n^2 \rangle$. Therefore, by initiality of K , there exists a unique continuous function $g : K \rightarrow Dj$ such that

$$\begin{array}{ccc}
 & K & \\
 & \uparrow & \searrow g \\
 K[-] & & D_j \\
 & \uparrow \text{=} & \\
 & B & \xrightarrow{\check{f}} D_j
 \end{array}$$

Therefore,

$$\begin{array}{ccccc}
 K & \xrightarrow{g} & D_j & \xrightarrow{c_j} & C \\
 & \searrow \text{=} & \uparrow \check{f} & \text{=} & \nearrow \\
 & & B & & \\
 & \nearrow K[-] & & \searrow f \circ K[-] &
 \end{array}$$

By couniversality, $c_j \circ g = f$, hence c_j factors through f .

Let $K \xrightarrow{g'} D_j' \xrightarrow{c_j'} C$ be any other factorisation. Then for any $k \in K[-][B] \subseteq K$, we have:

$$c_j'(g'(k)) = f(k) = c_j(g(k))$$

By Lemma 5.4, there exists some $\hat{i}_k \geq j', j$ such that

$$D(\hat{i}_k \geq j')(g'(k)) = D(\hat{i}_k \geq j)(g(k))$$

As $K[-][B]$ is countable, there is some \hat{i} in I bounding $\{j, j'\} \cup \{\hat{i}_k | k \in \mathfrak{t}[B]\}$. We then have that $D(\hat{i} \geq j) \circ g$ and $D(\hat{i} \geq j') \circ g'$ agree on $K[-][B]$. By Lemma 5.26, $\mathfrak{t}[B]$ is dense in K , hence $D(\hat{i} \geq j) \circ g = D(\hat{i} \geq j') \circ g'$.

We have shown that D_j factorises essentially uniquely through f . Therefore, K is countably presentable. ■

Thus, we have characterised the countably presentable domains:

Theorem 5.31. *A domain is countably presentable if and only if it has a countable presentation.*

More explicitly, an ω -cpo K is countably presentable if and only if there exists a countable subset $B \subseteq K$, and a countable binary relation \sqsubseteq between pairs of ω -chains from B , such that:

- for every pair $\langle b_n \rangle \sqsubseteq \langle b'_n \rangle$, $\bigvee b_n \leq \bigvee b'_n$; and
- for any ω -cpo W and monotone function $f : B \rightarrow W$, such that, for all $\langle b_n \rangle \sqsubseteq \langle b'_n \rangle$, $\bigvee f(b_n) \leq \bigvee f(b'_n)$, there exist a unique Scott-continuous function $\hat{f} : K \rightarrow W$ satisfying

$$\begin{array}{ccc}
 K & & \\
 \cup \downarrow & \hat{f} \searrow & \\
 B & \xrightarrow{f} & W
 \end{array}$$

Proof

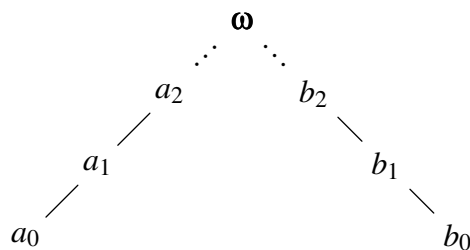
From Propositions 5.29, 5.30 follows that the countably presentable domains are precisely the domains with a countable presentation. For the explicit formulation, note that, for any presentation \mathcal{P} , every initial \mathcal{P} -model K is also initial for the presentation $\langle K[-][B], K[-][I], K[-] \times K[-][\sqsubseteq] \rangle$, but with the inclusion replacing $K[-]$. ■

Note that this universal characterisation of countably presentable domains is still unsatisfactory. A predicative characterisation that does not quantify over all other domains would be more satisfying. Nevertheless, this characterisation does shed more light on the nature of the countably presentable domains, and suffices for our purposes.

In particular, we corroborate Hyland et al.'s observation that all ω -continuous ω -cpo's are countably presentable [HPP06]. Thus, the closed interval $[0, 1]$ with the arithmetic ordering shows that there are countably presentable domains with uncountably many elements, refuting Adámek and Rosický's claim that the countably presentable domains are precisely the domains with countably many elements [AR94].

We summarise our findings in the Venn diagram in Figure 5.1. To complete the picture, we add the following example:

Example 5-12. Consider the constraining data given as follows. The basis is the disjoint union of two copies of ω , i.e., $\{a_0 \leq a_1 \leq a_2 \leq \dots\} \cup \{b_0 \leq b_1 \leq b_2 \leq \dots\}$. The constrained chains are $\langle a_n \rangle, \langle b_n \rangle$. The two constraints are $\langle a_n \rangle \sqsubseteq \langle b_n \rangle, \langle b_n \rangle \sqsubseteq \langle a_n \rangle$. The couniversal domain for this data is given pictorially as



This domain is countable and countably presentable, yet not ω -continuous. □

The category $\omega\mathbf{CPO}$ is locally countably presentable. We first establish the existence of a set \mathcal{A} of isomorphism classes of countably presentable domains:

Theorem 5.32. *Up to isomorphism, $\omega\mathbf{CPO}$ has a set of countably presentable domains.*

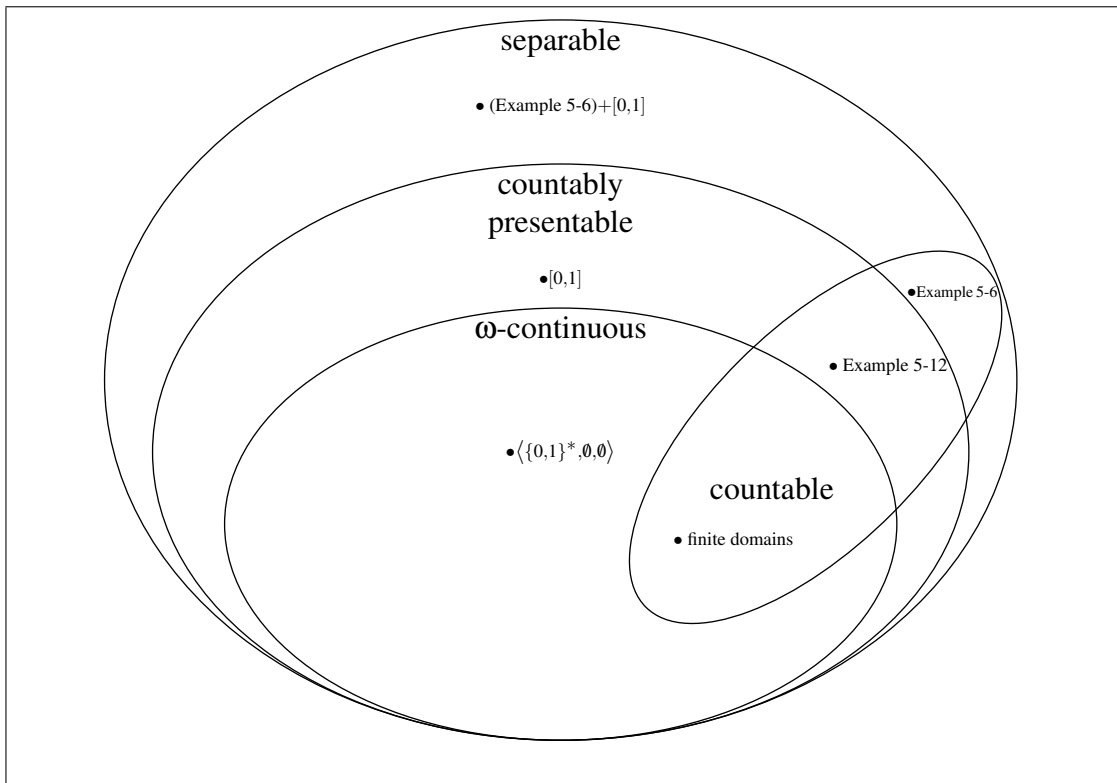


Figure 5.1: characterisation summary

Proof

Let \mathcal{A} be the following set of countably presentable objects K whose carrier set is a cardinal, that are couniversal for some $\langle B, I, \sqsubseteq \rangle$, where B is a poset whose carrier is a countable cardinal, and I is a countable set of ω -chains over B . As $\iota[B]$ is dense in K and countable, \mathcal{A} is indeed a set. Any countably presentable object is then isomorphic to one of \mathcal{A} 's elements. ■

Putting together Corollary 5.5, and Theorems 5.31 and 5.32, we corroborate the following known fact [Mes81]:

Theorem 5.33. *The category ωCPO is locally countably presentable.*

And in particular, we obtain another proof for the following known fact (see, e.g., Meseguer [Mes81], or Barr and Wells [BW95, Exercise 14.5.7]):

Corollary 5.34. *The category ωCPO is cocomplete.*

Example 5-13. When $\lambda \geq \aleph_1$ and $|\text{Char}| < \lambda$, the domain theoretic monads for modelling recursion, I/O and exceptions are λ -ranked. □

To summarise, we reviewed the basic properties of locally presentable categories.

Chapter 6

Lawvere theories

Better fish are in the sea

Is not the theory for me

—Ella Fitzgerald



In this chapter we present the main technical apparatus of Plotkin and Power’s algebraic theory of effects: *enriched Lawvere theories*. Each enriched Lawvere theory gives rise to a ranked strong monad, and these include many programming language models [PP02]. We then define a subclass of our CBPV models with effects that arise through Lawvere theories.

First, in Section 6.1, we recall some concepts from enriched category theory, in particular *powers* and *copowers*. Then, in Section 6.2, following Power [Pow00], we define enriched Lawvere theories and recall the relevant results and constructions. Next, in Section 6.3, we define algebraic operations for a Lawvere theory. Finally, in Section 6.4, we present algebraic models of CBPV.

6.1 Enriched powers and copowers

We present the concepts enriched in a *symmetric monoidal closed category* \mathcal{V} , given by the data $\langle |\mathcal{V}|, \mathbb{1}, \otimes, \multimap, \text{unit}_{\text{left}}, \text{unit}_{\text{right}}, \text{assoc}, \text{symm} \rangle$. As we are mainly interested in monoidal categories arising from a cartesian closed structure, the extra generality of the monoidal setting is not necessary for our purposes. However, the monoidal notation is useful, as it highlights the use of the enriching structure, with no notational overhead. We denote the bijection given in the closed structure by

$$\lambda_{V.-} : \mathcal{V}(W \otimes V, U) \cong \mathcal{V}(W, V \multimap U)$$

its inverse by $(\lambda V)^{-1} \cdot -$, and set

$$\text{eval} : (V \multimap W) \otimes V \xrightarrow{(\lambda V)^{-1} \cdot \text{id}_{V \multimap W}} W$$

Given a \mathcal{V} -category C , we denote its objects by \underline{A} , \underline{B} , etc., its hom-objects by $C(\underline{A}, \underline{B})$, its identities by $\text{id}_{\underline{A}} : \mathbb{1} \rightarrow C(\underline{A}, \underline{A})$, and its composition by

$$\circ_{\underline{A}, \underline{B}, \underline{C}} : C(\underline{B}, \underline{C}) \otimes C(\underline{A}, \underline{B}) \rightarrow C(\underline{A}, \underline{C})$$

Using the closed structure, \mathcal{V} is then self-enriched. We also get a map:

$$C(\underline{A}, -) : C(\underline{B}, \underline{C}) \xrightarrow{\lambda_{C(\underline{A}, \underline{B})} \cdot \circ_{\underline{A}, \underline{B}, \underline{C}}} \mathcal{V}(C(\underline{A}, \underline{B}), C(\underline{A}, \underline{C}))$$

By varying \underline{B} and \underline{C} over all C -objects, the morphisms $C(\underline{A}, -)$ collate to the *post-composition*, or *representable*, \mathcal{V} -functor $C(\underline{A}, -) : C \rightarrow \mathcal{V}$. Dually, using the symmetric structure of \mathcal{V} , we obtain the contravariant *pre-composition*, or *co-representable*, \mathcal{V} -functor $C(-, \underline{C}) : C^{\text{op}} \rightarrow \mathcal{V}$. We denote by $|C|_0$ the ordinary category underlying C , and similarly for enriched functors and natural transformations.

Definition 6.1. *Let \mathcal{V} be a symmetric monoidal closed category, C a \mathcal{V} -category, V a \mathcal{V} -object, and \underline{A} a C -object. A power of \underline{A} by V is a pair $\langle \prod_V \underline{A}, \langle - : - \rightarrow \underline{A} \rangle_V \rangle$ consisting of a C -object $\prod_V \underline{A}$ and a \mathcal{V} -natural isomorphism*

$$\langle - : - \rightarrow \underline{A} \rangle_{\mathcal{V}} : V(V, C(-, \underline{A})) \rightarrow C(-, \prod_V \underline{A})$$

Given a power $\langle \prod_V \underline{A}, \langle - \rangle \rangle$, we define the counit as the composite

$$\pi_- : V \xrightarrow{\cong} \mathbb{1} \otimes V \xrightarrow{(\lambda V)^{-1} \cdot (\text{id} \xrightarrow{\text{id}} C(\prod_V \underline{A}, \prod_V \underline{A})) \xrightarrow{\langle - : \prod_V \underline{A} \rightarrow \underline{A} \rangle_V^{-1}} V \multimap C(\prod_V \underline{A}, \underline{A})} C(\prod_V \underline{A}, \underline{A})$$

Kelly's notation for the power is $V \mathfrak{h} \underline{A}$, while Power's notation for the power is \underline{A}^V . We chose the non-standard product notation because the universal arrow $\langle - : - \rightarrow \underline{A} \rangle_{\mathcal{V}}$ is more similar to the universal arrow of products, i.e., tupling, than the universal arrow of exponentials, i.e., evaluation.

Example 6-1 (Kelly [Kel82a, Section 3.7]). If we consider the self-enrichment of \mathcal{V} , then all powers exist. The power object $\prod_V W$ is given by $V \multimap W$, and the natural isomorphism by

$$\langle - : U \rightarrow W \rangle_V : (V \multimap (U \multimap W)) \xrightarrow{\cong} (U \multimap (V \multimap W))$$

whose existence follows from symmetry. The counit is given by

$$\pi_- : V \xrightarrow{\lambda_{V \multimap W} \cdot (\text{eval} \circ \text{symm})} ((V \multimap W) \multimap W)$$

Thus, when the monoidal structure is cartesian, the self-enriched power coincides with the exponential. \square

Example 6-2 (Kelly [Kel82a, Section 3.7]). Consider the case $\mathcal{V} = \mathbf{Set}$ with the cartesian closed structure. Thus we consider locally small categories \mathcal{C} . Let X be a set and \underline{A} any \mathcal{C} -object. The power $\prod_X \underline{A}$ exists if and only if the small product $\prod_{x \in X} \underline{A}$ exists. The required natural isomorphism is given by universality of the product:

$$\begin{aligned} \langle - : \underline{B} \rightarrow \underline{A} \rangle_X : (\mathcal{C}(\underline{B}, \underline{A}))^X &\rightarrow \mathcal{C}(\underline{B}, \prod_{x \in X} \underline{A}) \\ \langle - : \underline{B} \rightarrow \underline{A} \rangle_X : \lambda x. f_x &\mapsto \langle f_x \rangle_{x \in X} \end{aligned}$$

The counit is given by the projections:

$$\begin{aligned} \pi_- : X &\rightarrow \mathcal{C}(\prod_{x \in X} \underline{A}, \underline{A}) \\ \pi_- : x &\mapsto \pi_x \end{aligned}$$

This example explains our choice of notation. \square

We now examine the case $\mathcal{V} = \omega\mathbf{CPO}$ with the cartesian closed structure. The following notion will make the characterisation more succinct:

Definition 6.2. Let \mathcal{C} be an $\omega\mathbf{CPO}$ -enriched category, and W an ω -cpo. A W -compatible family from \underline{A} to \underline{B} in \mathcal{C} is a family of \mathcal{C} -arrows $f_w : \underline{A} \rightarrow \underline{B}$ indexed by $w \in W$, such that, for all $w \leq w'$ in W , $f_w \leq f_{w'}$, and, for all ω -chains $\langle w_n \rangle$ in W , $\bigvee f_{w_n} = f_{\bigvee w_n}$.

Thus, the elements of the ω -cpo $(\mathcal{C}(\underline{A}, \underline{B}))^W$ are precisely the W -compatible families from \underline{A} to \underline{B} .

Proposition 6.3. Let \mathcal{C} be an $\omega\mathbf{CPO}$ -enriched category, W an ω -cpo, and \underline{A} any \mathcal{C} -object. An object $\prod_W \underline{A}$ in \mathcal{C} is a power of \underline{A} by W if and only if there is a W -compatible family $\pi_w : \prod_W \underline{A} \rightarrow \underline{A}$ such that, for all W -compatible families f_w from every \underline{B} to \underline{A} there exists a unique morphism $\langle f_w \rangle_{w \in W}$ satisfying, for all $w \in W$:

$$\begin{array}{ccc} \underline{B} & & \\ \langle f_w \rangle_{w \in W} \downarrow & \searrow f_w & \\ \prod_W \underline{A} & \xrightarrow{\pi_w} & \underline{A} \end{array} \quad =$$

and the following evident map is Scott-continuous:

$$\langle - \rangle_W : (\mathcal{C}(\underline{B}, \underline{A}))^W \rightarrow \mathcal{C}(\underline{B}, \prod_W \underline{A})$$

Proof

First, note that given an $\omega\mathbf{CPO}$ -enriched power $\langle \prod_W \underline{A}, \langle - \rangle \rangle$, the counit is given by

$$\pi_- : w \xrightarrow{\cong} \langle \star, w \rangle \xrightarrow{(\lambda_W)^{-1} \cdot \lambda_\star \cdot \langle \text{id}_{\prod_W \underline{A}} \rangle^{-1}} \langle \text{id}_{\prod_W \underline{A}} \rangle^{-1}(w)$$

Next, note that $\langle - \rangle$'s $\omega\mathbf{CPO}$ -naturality amounts to the following diagram chasing for any $h : \underline{B} \rightarrow \underline{C}$ in \mathcal{C} , and W -compatible family f_w from \underline{B} to \underline{A} :

$$\begin{array}{ccc} \lambda_w \cdot \underline{B} \xrightarrow{f_w} \underline{A} & \xrightarrow{\langle - \rangle} & \left(\underline{B} \xrightarrow{\langle \lambda_w \cdot f_w \rangle} \prod_W \underline{A} \right) \\ \downarrow (C(h, \underline{A}))^W & & \downarrow C\left(h, \prod_W \underline{A}\right) \\ \lambda_w \cdot \left(\underline{C} \xrightarrow{h} \underline{B} \xrightarrow{f_w \circ h} \underline{A} \right) & \xrightarrow{\langle - \rangle} & \left(\underline{C} \xrightarrow{h} \underline{B} \xrightarrow{\langle \lambda_w \cdot f_w \rangle} \prod_W \underline{A} \right) \\ & & \Downarrow \\ \lambda_w \cdot \left(\underline{C} \xrightarrow{h} \underline{B} \xrightarrow{f_w \circ h} \underline{A} \right) & \xrightarrow{\langle - \rangle} & \left(\underline{C} \xrightarrow{\langle \lambda_w \cdot (f_w \circ h) \rangle} \prod_W \underline{A} \right) \end{array}$$

Inverting $\langle - \rangle$ then yields, for all $g : \underline{B} \rightarrow \prod_W \underline{A}$:

$$\langle g \circ h \rangle^{-1} = \lambda_w \cdot (\langle g \rangle^{-1}(w) \circ h) \quad (6.1)$$

For necessity, assume that the power $\langle \prod_W \underline{A}, \langle - \rangle \rangle$ exists. Choose π_w as the W -compatible family, and for any W -compatible family, choose $\langle f_w \rangle_{w \in W}$ as $\langle \lambda_w \cdot f_w \rangle$. We then have, for all w :

$$\begin{array}{c} \text{Equation (6.1)} \\ \downarrow \\ \pi_w \circ \langle \lambda_w \cdot f_w \rangle = \langle \text{id} \rangle^{-1}(w) \circ \langle \lambda_w \cdot f_w \rangle \stackrel{\downarrow}{=} \langle \text{id} \circ \langle \lambda_w \cdot f_w \rangle \rangle^{-1} w = \lambda_w \cdot f_w w = f_w \end{array}$$

As $\langle - \rangle$ is a Scott continuous bijection, we are done.

Conversely, assume the existence of the W -compatible family π_w and of $\langle - \rangle$, as in the Proposition's statement. Scott-continuity of $\langle - \rangle$ ensures we have a candidate morphism $\langle - \rangle$ for the natural isomorphism. Define:

$$\langle g \rangle^{-1} := \lambda_w \cdot (\pi_w \circ g)$$

Continuity of post-composition in the $\omega\mathbf{CPO}$ -enriched category \mathcal{C} , together with π_w being W -compatible, ensures $\langle g \rangle^{-1}$ is a Scott-continuous function, hence in $(C(\underline{B}, \underline{A}))^W$.

Continuity of pre-composition then ensures $\langle - \rangle^{-1}$ is a Scott-continuous function from $C(\underline{B}, \prod_W \underline{A})$ to $(C(\underline{B}, \underline{A}))^W$. Direct calculation, using the assumed commutative triangle, validates that $\langle - \rangle$ and $\langle - \rangle^{-1}$ are inverse to each other. Therefore, for each \underline{B} , $\langle - : \underline{B} \rightarrow \underline{A} \rangle$ is an isomorphism. Post-composing with π_w yields:

$$\pi_w \circ \langle \lambda_w.f_w \rangle \circ h = f_w \circ h = \pi_w \circ \langle \lambda_w.(f_w \circ h) \rangle$$

Thus, universality implies $\langle \lambda_w.f_w \rangle \circ h = \langle \lambda_w.(f_w \circ h) \rangle$, and $\langle - \rangle$ is natural. Therefore, $\prod_W \underline{A}$ is the power of \underline{A} by W . Moreover, direct calculation shows that the counit then satisfies $\pi_-(w) = \pi_w$. ■

Note that when W is discrete, the power degenerates back to the **Set**-enriched case.

The discussion dualises in a straightforward manner:

Definition 6.4. Let \mathcal{V} be a symmetric monoidal closed category, C a \mathcal{V} -category, V a \mathcal{V} -object, and \underline{A} a C -object. A copower of \underline{A} by V is a pair $\langle \sum_V \underline{A}, [- : \underline{A} \rightarrow -]_V \rangle$ consisting of a C -object $\sum_V \underline{A}$ and a \mathcal{V} -natural isomorphism

$$[- : \underline{A} \rightarrow -]_V : \mathcal{V}(V, C(\underline{A}, -)) \rightarrow C(\sum_V \underline{A}, -)$$

Given a copower $\langle \sum_V \underline{A}, [-] \rangle$, we define the counit as the composite

$$\iota_- : V \xrightarrow{\cong} \mathbb{1} \otimes V \xrightarrow{(\lambda_V)^{-1} \cdot (\mathbb{1} \xrightarrow{\text{id}} C(\sum_V \underline{A}, \sum_V \underline{A})) \xrightarrow{[- : \sum_V \underline{A} \rightarrow \underline{A}]_V^{-1}} V \multimap C(\underline{A}, \sum_V \underline{A})} C(\underline{A}, \sum_V \underline{A})$$

Both Kelly and Power denote the copower by $V \otimes \underline{A}$. As for powers, we prefer to use the coproduct notation as the universal arrow $[- : \underline{A} \rightarrow -]_V$ is more similar to the coproduct universal arrow, i.e., cotupling, than to the product universal arrow, i.e., tupling.

Example 6-3 (Kelly [Kel82a, Section 3.7]). If we consider the self-enrichment of \mathcal{V} , then all copowers exist. The copower object $\sum_V W$ is given by $V \otimes W$, and the natural isomorphism by

$$[- : W \rightarrow U]_V : (V \multimap (W \multimap U)) \xrightarrow{\cong} ((V \otimes W) \multimap U)$$

whose existence follows from the closed structure. The counit is given by

$$\iota_- : V \xrightarrow{\lambda_W \cdot \text{id}} (W \multimap (V \otimes W))$$

Thus, when the monoidal structure is cartesian, the self-enriched copower coincides with the binary product. □

Example 6-4 (Kelly [Kel82a, Section 3.7]). Consider the case $\mathcal{V} = \mathbf{Set}$ with the cartesian closed structure. Thus we consider locally small categories \mathcal{C} . Let X be a set and \underline{A} any \mathcal{C} -object. The copower $\sum_X \underline{A}$ exists if and only if the small coproduct $\sum_{x \in X} \underline{A}$ exists. The required natural isomorphism is given by universality of the product:

$$\begin{aligned} [- : \underline{B} \rightarrow \underline{A}]_X &: (\mathcal{C}(\underline{A}, \underline{B}))^X \rightarrow \mathcal{C}(\sum_{x \in X} \underline{A}, \underline{B}) \\ [- : \underline{A} \rightarrow \underline{B}]_X &: \lambda x. f_x \quad \mapsto [f_x]_{x \in X} \end{aligned}$$

The counit is given by the injections::

$$\begin{aligned} \iota_- &: X \rightarrow \mathcal{C}(\underline{A}, \sum_{x \in X} \underline{A}) \\ \iota_- &: x \mapsto \iota_x \end{aligned}$$

This example explains our choice of notation. \square

We now examine the case $\mathcal{V} = \omega\mathbf{CPO}$ with the cartesian closed structure.

Proposition 6.5. *Let \mathcal{C} be an $\omega\mathbf{CPO}$ -enriched category, W an ω -cpo, and \underline{A} any \mathcal{C} -object. An object $\prod_W \underline{A}$ in \mathcal{C} is a copower of \underline{A} by W if and only if there is a W -compatible family $\iota_w : \underline{A} \rightarrow \sum_W \underline{A}$ such that, for all W -compatible families f_w from \underline{A} to every \underline{B} there exists a unique morphism $[f_w]_{w \in W}$ satisfying, for all $w \in W$:*

$$\begin{array}{ccc} & \underline{B} & \\ & \uparrow [f_w]_{w \in W} & \\ & \sum_W \underline{A} & \xleftarrow{\iota_w} \underline{A} \\ & & \swarrow f_w \\ & & \underline{A} \end{array} \quad \begin{array}{c} \\ \\ \\ = \end{array}$$

and the following evident map is Scott-continuous:

$$[-]_W : (\mathcal{C}(\underline{A}, \underline{B}))^W \rightarrow \mathcal{C}(\sum_W \underline{A}, \underline{B})$$

Proof

By duality, i.e., choose $\mathcal{C} := \mathcal{C}^{\text{op}}$ and apply Proposition 6.3. \blacksquare

Finally, we discuss preservation of powers and copowers:

Definition 6.6. *Let $F : \mathcal{B} \rightarrow \mathcal{C}$ be a \mathcal{V} -functor.*

1. *Let $\langle \prod_V \underline{B}, \langle - \rangle \rangle$ be a power in \mathcal{B} . We say that F preserves the power $\langle \prod_V \underline{B}, \langle - \rangle \rangle$ if $F(\prod_V \underline{B})$ is the power $\prod_V (F\underline{B})$ in \mathcal{C} , and the corresponding counit π_- is given by:*

$$F \circ \pi_- : V \xrightarrow{\pi_-} \mathcal{B} \left(\prod_V \underline{B}, \underline{B} \right) \xrightarrow{F} \mathcal{C} \left(F \left(\prod_V \underline{B} \right), F\underline{B} \right)$$

2. Dually, let $\langle \sum_V \underline{B}, [-] \rangle$ be a copower in \mathcal{B} . We say that F preserves the copower $\langle \sum_V \underline{B}, [-] \rangle$ if $F(\sum_V \underline{B})$ is the copower $\sum_V (F\underline{B})$ in \mathcal{C} , and the corresponding counit ι_- is given by:

$$F \circ \iota_- : V \xrightarrow{\iota_-} \mathcal{B}(\underline{B}, \sum_V \underline{B}) \xrightarrow{F} \mathcal{C}(F(\sum_V \underline{B}), F\underline{B})$$

Example 6-5. In the case $\mathcal{V} = \mathbf{Set}$ with the cartesian closed structure, an ordinary functor $F : \mathcal{B} \rightarrow \mathcal{C}$ preserves the power $\prod_{x \in X} \underline{B}$ if and only if $F(\prod_{x \in X} \underline{B})$ is the product $\prod_{x \in X} F\underline{B}$ in \mathcal{C} exhibited by $F\pi_x$ as the projection to the x component, for every $x \in X$. In other words, F preserves the power $\prod_X \underline{B}$ if and only if it preserves the product $\prod_{x \in X} \underline{B}$ in the usual sense. Similarly, a functor preserves the copower $\sum_X \underline{B}$ if and only if it preserves the coproduct $\sum_{x \in X} \underline{A}$ in the usual sense. \square

Example 6-6. In the case $\mathcal{V} = \omega\mathbf{CPO}$ with the cartesian closed structure, an $\omega\mathbf{CPO}$ -enriched functor $F : \mathcal{B} \rightarrow \mathcal{C}$ preserves the power $\prod_W \underline{A}$ if and only if the W -compatible family $F\pi_w : \prod_W \underline{B} \rightarrow \underline{B}$ exhibits $F(\prod_W \underline{A})$ as the power $\prod_W (F\underline{B})$. Similarly, F preserves the copower $\sum_W \underline{A}$ if and only if the W -compatible family $F\iota_w : \underline{B} \rightarrow \sum_W \underline{B}$ exhibits $F(\sum_W \underline{A})$ as the copower $\sum_W (F\underline{B})$. \square

Finally, we note that powers are a special case of *indexed limits*, also known as *enriched limits*, and *cylindrical limits*, and copowers are a special case of *indexed colimits*. Therefore, powers are preserved by enriched right-adjoint enriched functors, and copowers are preserved by enriched left-adjoint enriched functors.

Example 6-7 (Enriched Kleisli category). Let T be a monad on \mathcal{C} . Recall the Kleisli category \mathcal{C}_T , whose objects are the \mathcal{C} -objects, and morphisms f from V to W are the morphisms $f : A \rightarrow TA$ in \mathcal{C} . When both \mathcal{C} and T are \mathcal{V} -enriched, this structure can be enriched to form a \mathcal{V} -category \mathcal{C}_T , by setting:

$$\begin{aligned} \mathcal{C}_T(\underline{A}, \underline{B}) &:= \mathcal{C}(\underline{A}, T\underline{B}) \\ \text{id}_{\underline{A}} &: \mathbb{1} \xrightarrow{\eta_{\underline{A}}} \mathcal{C}(\underline{A}, T\underline{A}) \\ \circ_{\underline{A}, \underline{B}, \underline{C}} &: \mathcal{C}(\underline{B}, T\underline{C}) \otimes \mathcal{C}(\underline{A}, T\underline{B}) \xrightarrow{T \otimes \text{id}} \mathcal{C}(T\underline{B}, T^2\underline{C}) \otimes \mathcal{C}(\underline{A}, T\underline{B}) \xrightarrow{\circ^{\mathcal{C}}} \mathcal{C}(\underline{A}, T^2\underline{C}) \xrightarrow{\mathcal{C}(\underline{A}, \mu)} \mathcal{C}(\underline{A}, T\underline{C}) \end{aligned}$$

where $\mathcal{C}(\underline{A}, \underline{\mu})$ is the application of the ordinary functor $| \mathcal{C}(\underline{A}, -) |_{\circ}$ underlying the post-composition functor.

Linton [Lin69] showed that \mathcal{C}_T is indeed a \mathcal{V} -category, even when \mathcal{V} is neither

closed nor symmetric. Moreover, he showed that by setting $F\underline{A} := \underline{A}$, $U\underline{A} := T\underline{A}$, and

$$F : C(\underline{A}, \underline{B}) \xrightarrow{C(\underline{A}, \underline{\eta})} C(\underline{A}, T\underline{B})$$

$$U : C(\underline{A}, T\underline{B}) \xrightarrow{T} C(T\underline{A}, T^2\underline{B}) \xrightarrow{C(T\underline{A}, \underline{\mu})} C(T\underline{A}, T\underline{B})$$

we obtain a \mathcal{V} -adjunction $F \dashv U : C_T \rightarrow C$ that resolves the \mathcal{V} -monad T .

Therefore, if C has the copower $\sum_V \underline{A}$, then C_T also has the copower of \underline{A} by V , with both copower objects coinciding, and the counit given by:

$$F\underline{\iota}_- : V \xrightarrow{\underline{\iota}_-} C(\underline{A}, \sum_V \underline{A}) \xrightarrow{C(\underline{A}, \underline{\eta})} C(\underline{A}, T \sum_V \underline{A})$$

Let T' be any other \mathcal{V} -monad over C , and $m : T \rightarrow T'$ a \mathcal{V} -monad morphism. Define the following \mathcal{V} -functor:

$$M : C_T \longrightarrow C_{T'}$$

$$M : \underline{A} \longmapsto \underline{A}$$

$$M : C(\underline{A}, T\underline{B}) \xrightarrow{C(\underline{A}, m_B)} C(\underline{A}, T'\underline{B})$$

Then M maps the copower $\sum_V \underline{A}$ in C_T to the copower $\sum_V \underline{A}$ in $C_{T'}$. Furthermore, M preserves the copower $\sum_V \underline{A}$, as we have:

$$\begin{array}{ccc}
 & & C(\underline{A}, T \sum_V \underline{A}) \\
 & \nearrow^{F\underline{\iota}_-} & \\
 V & \xrightarrow{\underline{\iota}_-} & C(\underline{A}, \sum_V \underline{A}) \\
 & \searrow_{F'\underline{\iota}_-} & \\
 & & C(\underline{A}, T' \sum_V \underline{A})
 \end{array}
 \begin{array}{l}
 \text{copower} \\
 \text{preservation} \\
 \underline{=} \\
 C(\underline{A}, \underline{\eta}) \\
 \text{enriched} \\
 \text{monad} \\
 \text{morphism} \\
 \underline{=} \\
 C(\underline{A}, m)
 \end{array}$$

By considering the self-enrichment on \mathcal{V} , we have that, for every \mathcal{V} -enriched monad T , \mathcal{V}_T has all copowers, given by $V \otimes W$, and counits:

$$V \xrightarrow{\lambda W \cdot \text{id}} W \multimap (V \otimes W) \xrightarrow{W \multimap \underline{\eta}} W \multimap T(V \otimes W)$$

That is, $\lambda W \cdot \eta_{V \otimes W}$ as the counit. □

Finally, we state a few technical results about powers and copowers. The calculations below become clearer through string diagrams (see, for example, Baez and

Stay [BS11]). However, we keep to commuting diagrams to avoid the overhead imposed by additional notation. In the calculations below, given a morphism $f : W \rightarrow V$ in a symmetric monoidal closed category \mathcal{V} , we denote by $\underline{f} : \mathbb{1} \rightarrow \mathcal{V}(W, V)$ the corresponding \mathcal{V} -arrow $\underline{f} := \lambda W. f \circ \text{unit}_{\text{left}}$.

Lemma 6.7. *Let $F : \mathcal{B} \rightarrow \mathcal{C}$ be a \mathcal{V} -enriched functor preserving the power $\prod_V \underline{B}$. Then*

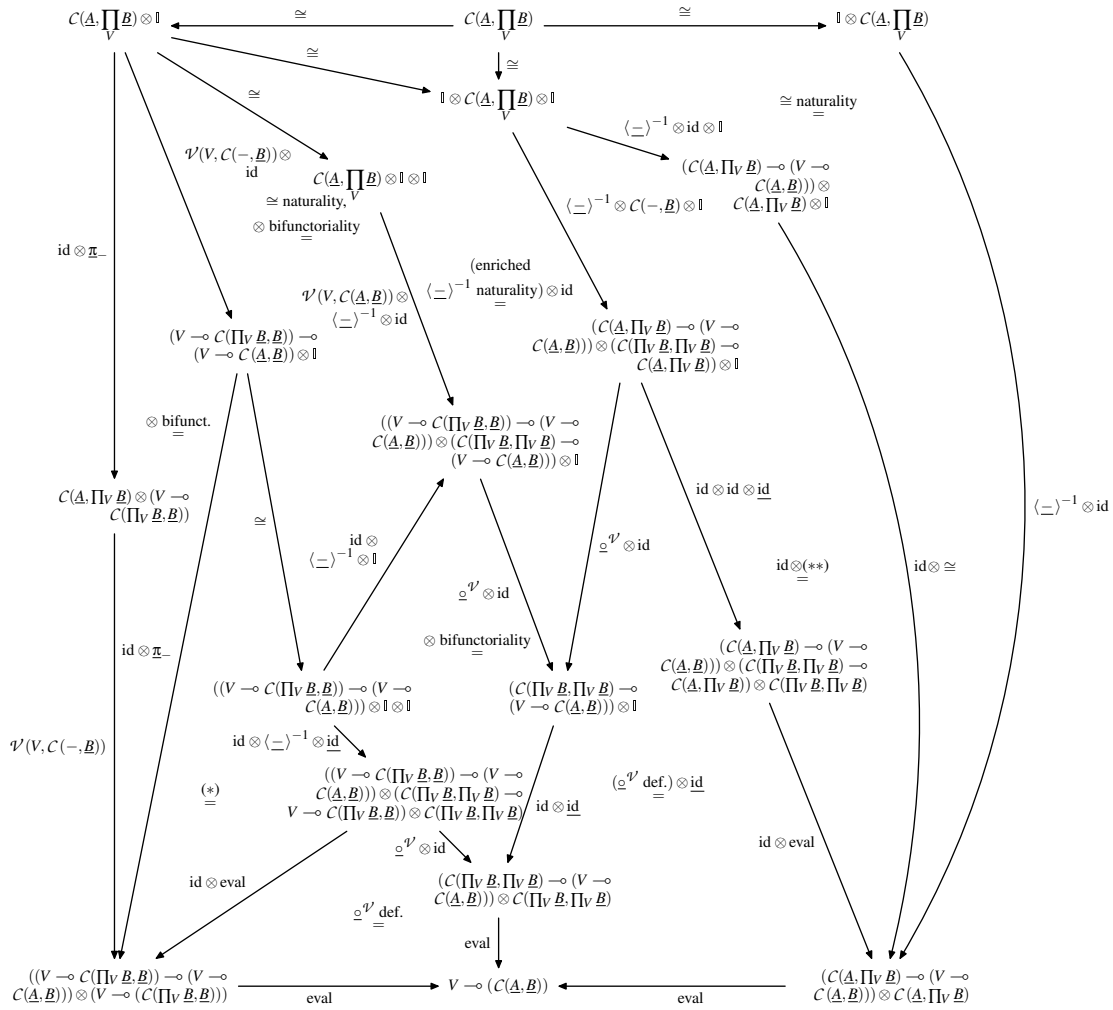
$$\begin{array}{ccc}
 V \multimap \mathcal{B}(-, \underline{B}) & \xrightarrow{\langle - \rangle_{\mathcal{B}}} & \mathcal{B}(-, \prod_V \underline{B}) \\
 \downarrow V \multimap F & = & \downarrow F \\
 V \multimap \mathcal{C}(-, F\underline{B}) & \xrightarrow{\langle - \rangle_{\mathcal{C}}} & \mathcal{C}(-, F \prod_V \underline{B})
 \end{array}$$

Proof

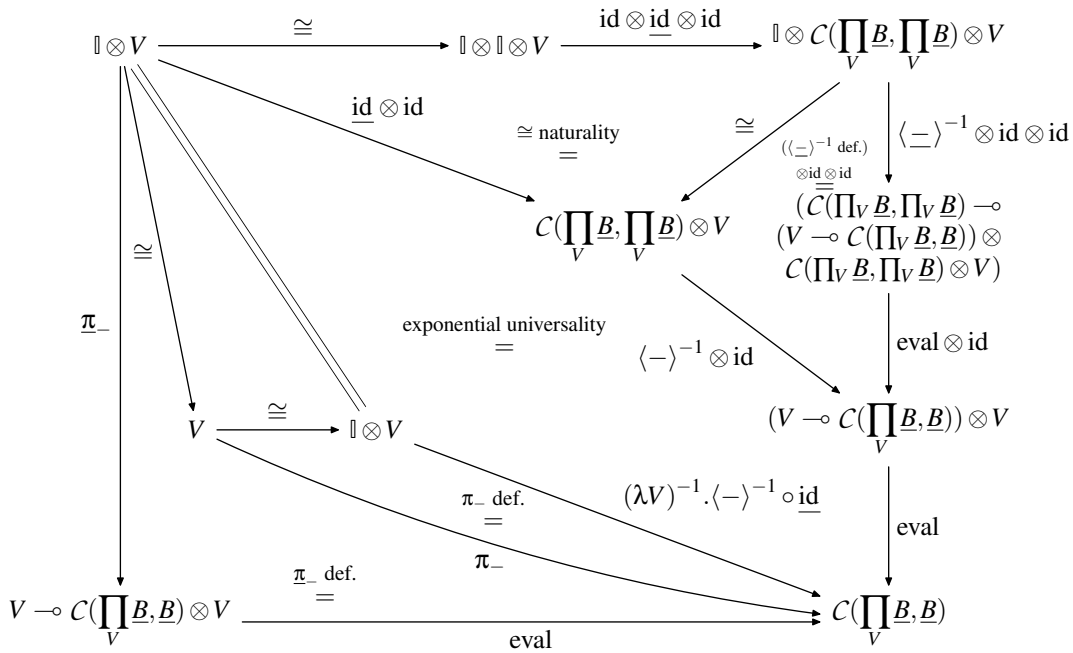
First, we prove that, for every power, the isomorphism $\langle - \rangle^{-1}$ is uniquely determined by the counit π_- :

$$\begin{array}{ccc}
 \mathcal{C}(-, \prod_V \underline{B}) & \xrightarrow{\langle - \rangle^{-1}} & V \multimap (\mathcal{C}(-, \underline{B})) \\
 \cong \swarrow & & \searrow \text{eval} \\
 \mathcal{C}(-, \prod_V \underline{B}) \otimes \mathbb{1} & = & ((V \multimap \mathcal{C}(\prod_V \underline{B}, \underline{B})) \multimap (V \multimap \mathcal{C}(-, \underline{B}))) \otimes (V \multimap (\mathcal{C}(\prod_V \underline{B}, \underline{B}))) \\
 \text{id} \otimes \pi_- \searrow & & \nearrow \mathcal{V}(V, \mathcal{C}(-, \underline{B})) \otimes \text{id} \\
 \mathcal{C}(-, \prod_V \underline{B}) \otimes (V \multimap \mathcal{C}(\prod_V \underline{B}, \underline{B})) & &
 \end{array} \tag{6.2}$$

Calculate:



where (*) follows from



and (**) follows from

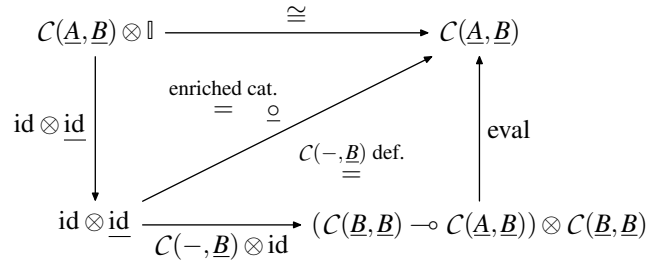
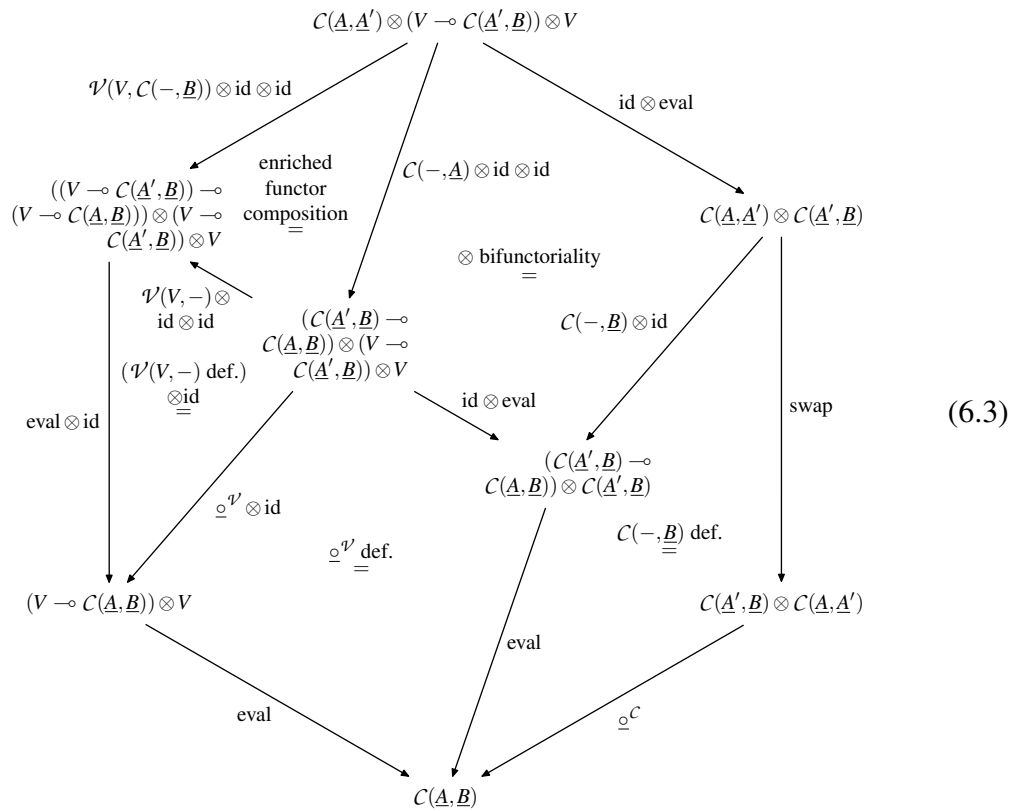
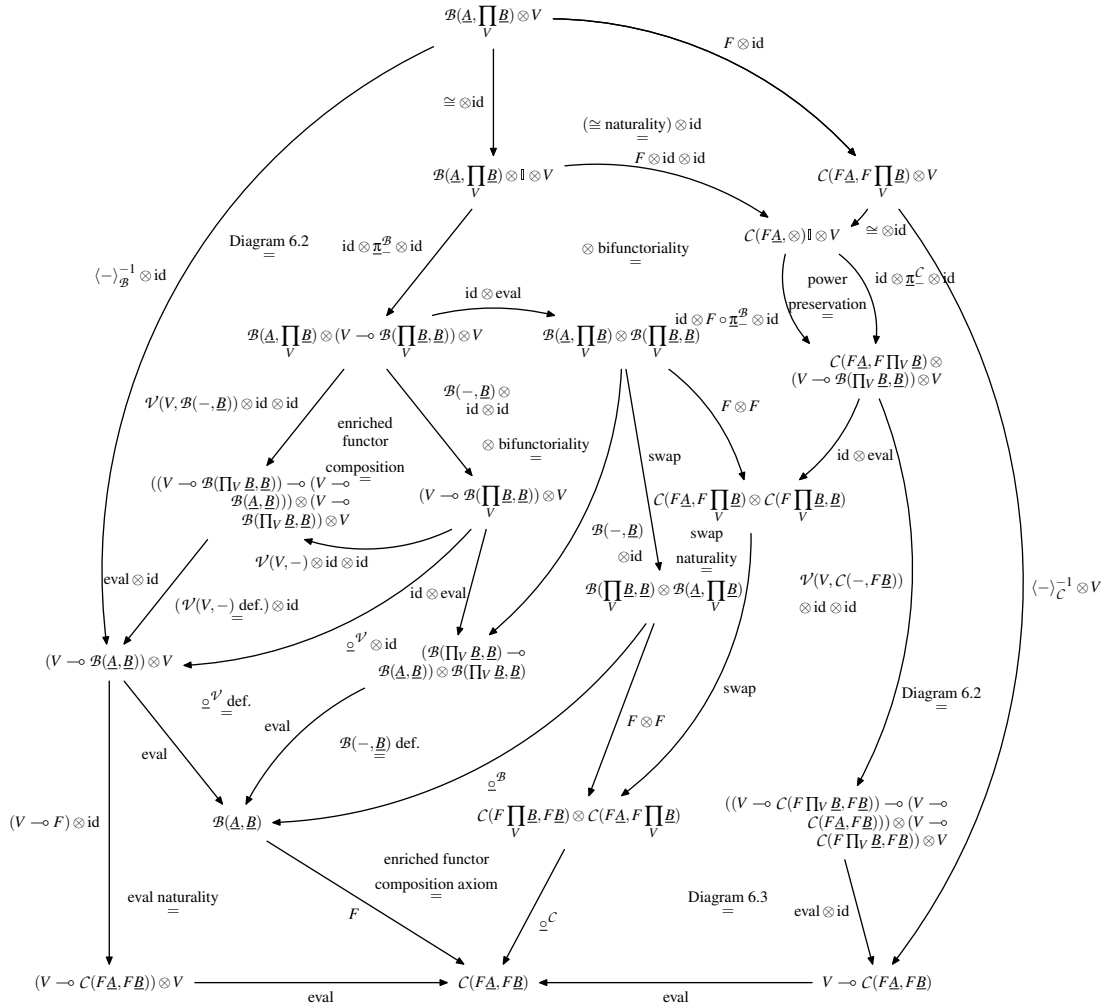


Diagram 6.2 then follows by noting that $\langle - \rangle^{-1} = \text{eval} \circ \langle _ \rangle^{-1} \otimes \text{id} \circ \cong$.

We will also use the following identity:

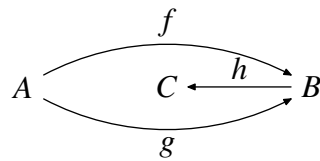


Finally, we calculate:

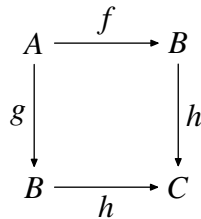


The desired equation follows by the universal property of the closed structure. ■

In the following, we will prove commutativity by post-composing with an arrow and showing the resulting diagram commutes, which misshapes the original diagram whose commutativity we want to establish. Therefore, we introduce the following notation. We say that a diagram of the shape



commutes when the following diagram commutes



We will use the following lemma to construct enriched categories with powers from other categories.

Lemma 6.8. *Let \mathcal{V} be a symmetric monoidal category, and \mathcal{C} a \mathcal{V} -category. Assume as given a class $\text{Ob}(\mathcal{B})$, and that, for every \mathcal{B} -members $\underline{A}, \underline{B}, \underline{C}$, assume as given:*

- a choice $\mathcal{B}(\underline{A}, \underline{B})$ of objects in \mathcal{V} ;
- a map F assigning to each \underline{A} in $\text{Ob}(\mathcal{B})$ an object $F\underline{A}$ in \mathcal{C} ;
- a choice $F_{\underline{A}, \underline{B}}: \mathcal{B}(\underline{A}, \underline{B}) \rightarrow \mathcal{C}(F\underline{A}, F\underline{B})$ of arrows in \mathcal{V} ;
- a choice $\underline{\text{id}}_{\underline{A}}^{\mathcal{B}}: \mathbb{1} \rightarrow \mathcal{B}(\underline{A}, \underline{A})$ of arrows in \mathcal{V} , satisfying

$$\begin{array}{ccc} & \underline{\text{id}}^{\mathcal{B}} & \mathcal{B}(\underline{A}, \underline{A}) \\ & \searrow & \downarrow F \\ \mathbb{1} & & \mathcal{C}(F\underline{A}, F\underline{A}) \\ & \swarrow & \\ & \underline{\text{id}}^{\mathcal{C}} & \end{array} =$$

- and a choice $\underline{\circ}^{\mathcal{B}}: \mathcal{B}(\underline{B}, \underline{C}) \otimes \mathcal{B}(\underline{A}, \underline{B}) \rightarrow \mathcal{B}(\underline{A}, \underline{C})$ of arrows in \mathcal{V} satisfying

$$\begin{array}{ccc} \mathcal{B}(\underline{B}, \underline{C}) \otimes \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\underline{\circ}^{\mathcal{B}}} & \mathcal{B}(\underline{A}, \underline{C}) \\ F \otimes F \downarrow & = & \downarrow F \\ \mathcal{C}(F\underline{B}, F\underline{C}) \otimes \mathcal{C}(F\underline{A}, F\underline{B}) & \xrightarrow{\underline{\circ}^{\mathcal{C}}} & \mathcal{C}(F\underline{A}, F\underline{C}) \end{array}$$

1. The following diagrams commute:

$$\begin{array}{ccccc} (\mathcal{B}(\underline{C}, \underline{D}) \otimes \mathcal{B}(\underline{B}, \underline{C})) \otimes \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\cong} & \mathcal{B}(\underline{C}, \underline{D}) \otimes (\mathcal{B}(\underline{B}, \underline{C}) \otimes \mathcal{B}(\underline{A}, \underline{B})) & & \\ \downarrow \underline{\circ}^{\mathcal{B}} \otimes \text{id} & & \downarrow \text{id} \otimes \underline{\circ}^{\mathcal{B}} & & \\ \mathcal{B}(\underline{B}, \underline{D}) \otimes \mathcal{B}(\underline{A}, \underline{B}) & & \mathcal{B}(\underline{C}, \underline{D}) \otimes \mathcal{B}(\underline{A}, \underline{C}) & & \\ & \searrow \underline{\circ}^{\mathcal{B}} & \swarrow \underline{\circ}^{\mathcal{B}} & & \\ & & \mathcal{B}(\underline{A}, \underline{D}) & & \\ & & \uparrow F & & \\ & & \mathcal{C}(\underline{A}, \underline{D}) & & \\ & & \downarrow F & & \\ \mathbb{1} \otimes \mathcal{B}(\underline{A}, \underline{B}) & & \mathcal{B}(\underline{A}, \underline{B}) \otimes \mathbb{1} & & \\ \downarrow \underline{\text{id}}_{\underline{B}}^{\mathcal{B}} \otimes \text{id} & & \downarrow \text{id} \otimes \underline{\text{id}}_{\underline{A}}^{\mathcal{B}} & & \\ \mathcal{B}(\underline{B}, \underline{B}) \otimes \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\underline{\circ}^{\mathcal{B}}} & \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\underline{\circ}^{\mathcal{B}}} & \mathcal{B}(\underline{A}, \underline{B}) \otimes \mathcal{B}(\underline{A}, \underline{A}) \\ \downarrow \underline{\text{id}}_{\underline{B}}^{\mathcal{B}} \otimes \text{id} & & \downarrow \underline{\text{id}}_{\underline{B}}^{\mathcal{B}} \otimes \text{id} & & \downarrow \text{id} \otimes \underline{\text{id}}_{\underline{A}}^{\mathcal{B}} \\ \mathcal{C}(\underline{A}, \underline{B}) & \xrightarrow{F} & \mathcal{C}(\underline{A}, \underline{B}) & \xrightarrow{F} & \mathcal{C}(\underline{A}, \underline{B}) \end{array}$$

Assume further that this data makes \mathcal{B} a \mathcal{V} -category, and $F : \mathcal{B} \rightarrow \mathcal{C}$ a \mathcal{V} -functor.

2. Assume that for some object V in \mathcal{V} , and \underline{B} in $\text{Ob}(\mathcal{B})$, there is some $\prod_V \underline{B}$ in $\text{Ob}(\mathcal{B})$ such that $F \prod_V \underline{B}$ is the power $\prod_V F \underline{B}$ in \mathcal{C} exhibited by $\langle - \rangle_{\mathcal{C}}$. Given a choice of maps

$$\begin{aligned} \langle - : \underline{A} \rightarrow \underline{B} \rangle_{\mathcal{B}} : V \multimap \mathcal{B}(\underline{A}, \underline{B}) &\rightarrow \mathcal{B}(\underline{A}, \prod_V \underline{B}) \\ \langle - : \underline{A} \rightarrow \underline{B} \rangle_{\mathcal{B}}^{-1} : V \multimap \mathcal{B}(\underline{A}, \underline{B}) &\leftarrow \mathcal{B}(\underline{A}, \prod_V \underline{B}) \end{aligned}$$

satisfying

$$\begin{array}{ccc} V \multimap \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\langle - \rangle_{\mathcal{B}}} & \mathcal{B}(\underline{A}, \prod_V \underline{B}) \\ \downarrow V \multimap F & = & \downarrow F \\ V \multimap C(F\underline{A}, F\underline{B}) & \xrightarrow{\langle - \rangle_{\mathcal{C}}} & C(F\underline{A}, F \prod_V \underline{B}) \end{array} \quad \begin{array}{ccc} V \multimap \mathcal{B}(\underline{A}, \underline{B}) & \xleftarrow{\langle - \rangle_{\mathcal{B}}^{-1}} & \mathcal{B}(\underline{A}, \prod_V \underline{B}) \\ \downarrow V \multimap F & = & \downarrow F \\ V \multimap C(F\underline{A}, F\underline{B}) & \xleftarrow{\langle - \rangle_{\mathcal{C}}^{-1}} & C(F\underline{A}, F \prod_V \underline{B}) \end{array}$$

then the following diagrams commute:

$$\begin{array}{ccccc} & & \mathcal{B}(\underline{A}, \prod_V \underline{B}) & \xlongequal{\quad} & \mathcal{B}(\underline{A}, \prod_V \underline{B}) \\ & \nearrow \langle - \rangle_{\mathcal{B}} & \searrow \langle - \rangle_{\mathcal{B}}^{-1} & & \nearrow \langle - \rangle_{\mathcal{B}} \\ & & C(F\underline{A}, F \prod_V \underline{B}) & & \\ & \nearrow V \multimap C(F\underline{A}, F\underline{B}) & \searrow F & & \nearrow F \\ V \multimap \mathcal{B}(\underline{A}, \underline{B}) & \xlongequal{\quad} & V \multimap \mathcal{B}(\underline{A}, \underline{B}) & & \end{array}$$

$$\begin{array}{ccc} & & ((V \multimap \mathcal{B}(\underline{A}, \underline{B})) \multimap \mathcal{B}(\underline{A}, \prod_V \underline{B})) \otimes \\ & & ((V \multimap \mathcal{B}(\underline{A}', \underline{B})) \multimap (V \multimap \mathcal{B}(\underline{A}, \underline{B}))) \otimes \\ \langle - \rangle_{\mathcal{B}} \otimes \mathcal{V}(V, \mathcal{B}(-, \underline{B})) \otimes \text{id} & \nearrow & (V \multimap \mathcal{B}(\underline{A}, \underline{B})) \\ \cong \otimes \text{id} & \nearrow & \downarrow \circ^{\mathcal{B}} \otimes \text{id} \\ \mathcal{B}(\underline{A}, \underline{A}') \otimes & & ((V \multimap \mathcal{B}(\underline{A}, \underline{B})) \multimap \\ (V \multimap \mathcal{B}(\underline{A}, \underline{B})) & \xleftarrow{F} & \mathcal{B}(\underline{A}, \prod_V \underline{B}) \xleftarrow{\text{eval}} & \mathcal{B}(\underline{A}, \prod_V \underline{B})) \otimes \\ & & & (V \multimap \mathcal{B}(\underline{A}, \underline{B})) \\ \cong \otimes \text{id} & \nearrow & & \downarrow \circ^{\mathcal{B}} \otimes \text{id} \\ \mathcal{B}(\underline{A}, \underline{A}') \otimes \mathbb{1} \otimes & & & (\mathcal{B}(\underline{A}', \prod_V \underline{B}) \multimap \mathcal{B}(\underline{A}, \prod_V \underline{B})) \otimes \\ (V \multimap \mathcal{B}(\underline{A}, \underline{B})) & \xrightarrow{\langle - \rangle_{\mathcal{B}}} & & ((V \multimap \mathcal{B}(\underline{A}', \underline{B})) \multimap \mathcal{B}(\underline{A}', \prod_V \underline{B})) \otimes \\ & & & (V \multimap \mathcal{B}(\underline{A}, \underline{B})) \end{array}$$

$$\begin{array}{ccc}
 & \mathcal{B}(\prod_V \underline{B}, \underline{B}) & \\
 \pi_-^{\mathcal{B}} \nearrow & \downarrow F & \\
 V & & \\
 \pi_-^{\mathcal{C}} \searrow & \mathcal{C}(F \prod_V \underline{B}, F\underline{B}) &
 \end{array}$$

where $\pi_-^{\mathcal{B}}$ is a formal counit as defined as for powers, i.e.,

$$\pi_- : V \xrightarrow{\cong} \mathbb{1} \otimes V \xrightarrow{(\lambda V)^{-1} \cdot (\mathbb{1} \xrightarrow{\text{id}} \mathcal{C}(\prod_V \underline{A}, \prod_V \underline{A}))} \mathcal{C}(\prod_V \underline{A}, \underline{A}) \xrightarrow{\langle -, \prod_V \underline{A} \rightarrow \underline{A} \rangle_V^{-1}} \mathcal{C}(\prod_V \underline{A}, \underline{A})$$

3. Assume a given \mathcal{V} -functor $G : \mathcal{A} \rightarrow \mathcal{C}$ such that F factors through G 's object map, i.e. there is a map H from $\text{Ob}(\mathcal{A})$ to $\text{Ob}(\mathcal{B})$ such that $G = F \circ H$, and a choice $H_{\underline{A}, \underline{B}} : \mathcal{A}(\underline{A}, \underline{B}) \rightarrow \mathcal{B}(H\underline{A}, H\underline{B})$ of arrows in \mathcal{V} , satisfying:

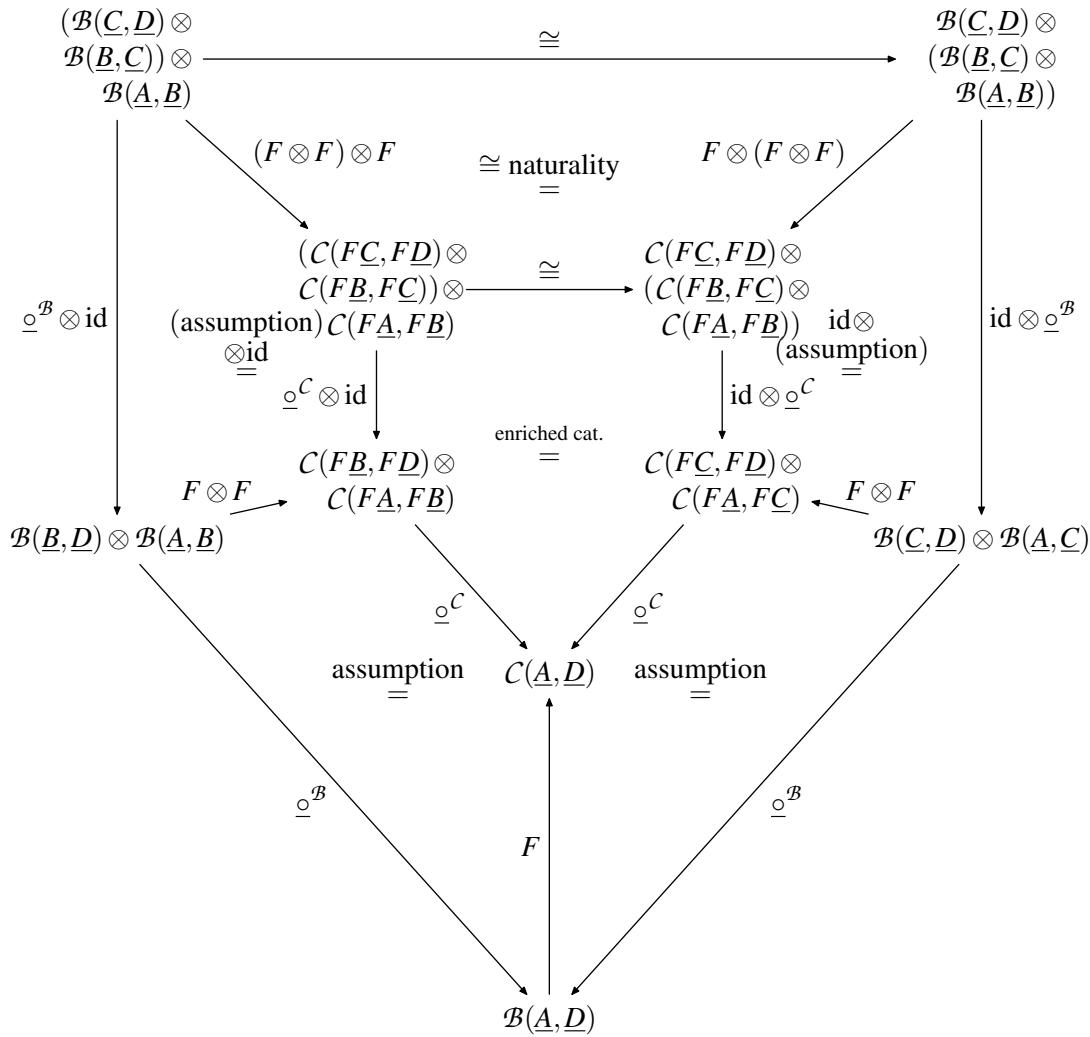
$$\begin{array}{ccc}
 & \mathcal{B}(H\underline{A}, H\underline{B}) & \\
 H \nearrow & \downarrow F & \\
 \mathcal{A}(\underline{A}, \underline{B}) & = & \\
 G \searrow & \mathcal{C}(F\underline{A}, F\underline{B}) &
 \end{array}$$

then the following diagrams commute:

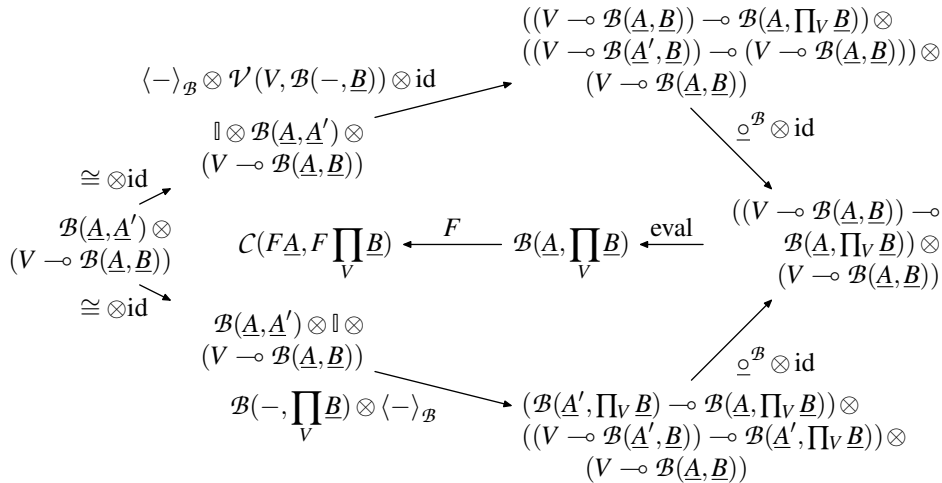
$$\begin{array}{ccccc}
 & \mathcal{A}(\underline{A}, \underline{A}) & \mathcal{A}(\underline{B}, \underline{C}) \otimes \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\circ_{\mathcal{A}}} & \mathcal{A}(\underline{A}, \underline{C}) \\
 \mathbb{1} \nearrow \text{id}^{\mathcal{A}} & \downarrow H & \downarrow H \otimes H & & \downarrow H \\
 & \mathcal{C}(F\underline{A}, F\underline{A}) & & \mathcal{C}(F\underline{A}, F\underline{C}) & \\
 & \downarrow F & & \downarrow F & \\
 \mathbb{1} \searrow \text{id}^{\mathcal{B}} & \mathcal{B}(H\underline{A}, H\underline{A}) & \mathcal{B}(H\underline{B}, H\underline{C}) \otimes \mathcal{B}(F\underline{A}, F\underline{B}) & \xrightarrow{\circ_{\mathcal{C}}} & \mathcal{B}(H\underline{A}, H\underline{C})
 \end{array}$$

Proof

With one notable exception in part 2, the proofs are straightforward calculations. These calculations reduce each diagram to the corresponding property of the enriched concept. For example, the proof of the first identity reduces the diagram to the associativity axiom of the enriched category \mathcal{C} .



The exception is the following diagram from part 2:



In this case too we reduce to the naturality of the isomorphism $\langle - \rangle_C$ in \mathcal{C} . However, this time the reduction requires some care. First, we note some properties involved in this proof.

Recall that the action of the contravariant functor $(-) \multimap W$ on morphisms $f : V \rightarrow V'$ is defined as:

$$f \multimap W := \lambda V. \left((V' \multimap W) \otimes V \xrightarrow{\text{id} \otimes f} (V' \multimap W) \otimes V' \xrightarrow{\text{eval}} W \right)$$

and satisfies the following naturality condition (see Mac Lane [ML98, Theorem IV.7.3]):

$$\lambda V. g \circ (\text{id} \otimes f) = (f \multimap W) \circ (\lambda V'. g) \quad (6.4)$$

Using the bifunctor $(-) \multimap (-)$ we can reformulate our assumption on $\langle - \rangle_{\mathcal{B}}$ as follows:

$$\begin{array}{ccc} \Downarrow & \xrightarrow{\langle - \rangle_{\mathcal{B}}} & (V \multimap \mathcal{B}(\underline{A}, \underline{B})) \otimes (\mathcal{B}(\underline{A}, \prod_V \underline{B})) \\ \langle - \rangle_{\mathcal{C}} \downarrow & = & \downarrow \text{id} \multimap F \\ (V \multimap \mathcal{C}(F\underline{A}, F\underline{B})) \otimes (\mathcal{C}(F\underline{A}, F \prod_V \underline{B})) & & (V \multimap \mathcal{B}(\underline{A}, \underline{B})) \otimes (\mathcal{C}(F\underline{A}, F \prod_V \underline{B})) \end{array} \quad (6.5)$$

$\xrightarrow{(V \multimap F) \multimap \text{id}}$

The proof is a straightforward calculation:

$$\begin{aligned} (\text{id} \multimap F) \circ \langle - \rangle_{\mathcal{B}} &= (\text{id} \multimap F) \circ (\lambda V \multimap \mathcal{B}(\underline{A}, \underline{B}). (\langle - \rangle_{\mathcal{B}} \circ \text{unit}_{\text{left}})) \\ &\stackrel{\lambda \text{ naturality}}{\downarrow} \\ &= \lambda V \multimap \mathcal{B}(\underline{A}, \underline{B}). (F \circ \langle - \rangle_{\mathcal{B}} \circ \text{unit}_{\text{left}}) \\ &\stackrel{\text{assumption}}{\downarrow} \\ &= \lambda V \multimap \mathcal{B}(\underline{A}, \underline{B}). (\langle - \rangle_{\mathcal{C}} \circ (V \multimap F) \circ \text{unit}_{\text{left}}) \\ &\stackrel{\text{unit}_{\text{left}} \text{ naturality}}{\downarrow} \\ &= \lambda V \multimap \mathcal{B}(\underline{A}, \underline{B}). (\langle - \rangle_{\mathcal{C}} \circ \text{unit}_{\text{left}} \circ (\text{id} \otimes (V \multimap F))) \\ &\stackrel{\lambda \text{ naturality (6.4)}}{\downarrow} \\ &= (V \multimap F) \circ \lambda V \multimap \mathcal{B}(\underline{A}, \underline{B}). (\langle - \rangle_{\mathcal{C}} \circ \text{unit}_{\text{left}}) \\ &= (V \multimap F) \circ \langle - \rangle_{\mathcal{C}} \end{aligned}$$

The contravariant hom-object \mathcal{V} functor $\mathcal{B}(-, \underline{B})$ satisfies

$$\begin{array}{ccc}
 \mathcal{B}(\underline{A}, \underline{A}') & \xrightarrow{F} & C(\underline{FA}, \underline{FA}') \\
 \downarrow \mathcal{B}(-, \underline{B}) & & \downarrow C(-, \underline{FB}) \\
 \mathcal{B}(\underline{A}', \underline{B}) \multimap \mathcal{B}(\underline{A}, \underline{B}) & \xrightarrow{\text{id} \multimap F} & \mathcal{B}(\underline{A}', \underline{B}) \multimap C(\underline{FA}, \underline{FB}) \\
 & & \downarrow F \multimap \text{id} \\
 & & \mathcal{B}(\underline{A}', \underline{B}) \multimap C(\underline{FA}, \underline{FB})
 \end{array}
 \quad (6.6)$$

Indeed,

$$\begin{aligned}
 & \downarrow C(-, \underline{FB}) \text{ def.} \\
 (F \rightarrow \text{id}) \circ C(-, \underline{FB}) \circ F &= (F \rightarrow \text{id}) \circ \left(\lambda C(\underline{FA}', \underline{FB}).(\underline{\circ}^C \circ \text{swap}) \right) \circ F \\
 & \downarrow \lambda \text{ naturality (6.4)} \\
 &= \lambda C(\underline{FA}', \underline{FB}).(\underline{\circ}^C \circ \text{swap} \circ (\text{id} \otimes F)) \circ F \\
 & \downarrow \lambda \text{ naturality} \\
 &= \lambda C(\underline{FA}', \underline{FB}).(\underline{\circ}^C \circ \text{swap} \circ (\text{id} \otimes F) \circ (F \otimes \text{id})) \\
 & \downarrow \otimes \text{ bifunctionality, swap naturality} \\
 &= \lambda C(\underline{FA}', \underline{FB}).(\underline{\circ}^C \circ F \otimes F \circ \text{swap}) \\
 & \downarrow \text{enriched functor composition axiom} \\
 &= \lambda C(\underline{FA}', \underline{FB}).(F \circ \underline{\circ}^{\mathcal{B}} \circ \text{swap}) \\
 & \downarrow \lambda \text{ naturality} \\
 &= (\text{id} \multimap F) \circ \left(\lambda C(\underline{FA}', \underline{FB}).(\underline{\circ}^{\mathcal{B}} \circ \text{swap}) \right) \\
 & \downarrow C(-, \underline{FB}) \text{ def.} \\
 &= (\text{id} \multimap F) \circ C(-, \underline{FB})
 \end{aligned}$$

Similar calculations establish the following three properties of the interaction between the composition $\underline{\circ}^{\mathcal{V}}$ in \mathcal{V} and \multimap for any morphism f in \mathcal{V} :

$$(\text{id} \multimap f) \circ \underline{\circ}^{\mathcal{V}} = \underline{\circ}^{\mathcal{V}} \circ ((\text{id} \multimap f) \otimes f) \quad (6.7a)$$

$$\underline{\circ}^{\mathcal{V}} \circ ((f \multimap \text{id}) \otimes \text{id}) = \underline{\circ}^{\mathcal{V}} \circ (\text{id} \otimes (\text{id} \multimap f)) \quad (6.7b)$$

$$\underline{\circ}^{\mathcal{V}} \circ (\text{id} \otimes (f \multimap \text{id})) = (f \multimap \text{id}) \circ \underline{\circ}^{\mathcal{V}} \quad (6.7c)$$

Another such calculation using these identities shows that:

$$\begin{array}{ccc}
 \mathcal{B}(\underline{A}, \underline{A}') & \xrightarrow{F} & C(\underline{FA}, \underline{FA}') \\
 \downarrow V \multimap \mathcal{B}(-, \underline{B}) & & \downarrow V \multimap C(-, \underline{FB}) \\
 (V \multimap \mathcal{B}(\underline{A}', \underline{B})) \multimap & = & (V \multimap C(\underline{FA}', \underline{FB})) \multimap \\
 (V \multimap \mathcal{B}(\underline{A}, \underline{B})) & \xrightarrow{\text{id} \multimap (\text{id} \multimap F)} & (V \multimap C(\underline{FA}, \underline{FB})) \\
 & & \downarrow (V \multimap F) \multimap \text{id} \\
 & & (V \multimap \mathcal{B}(\underline{A}', \underline{B})) \multimap \\
 & & (V \multimap C(\underline{FA}, \underline{FB}))
 \end{array} \tag{6.8}$$

The calculation on page 132 completes the proof. ■

6.2 Enriched Lawvere theories

We recall the relevant notions regarding *enriched Lawvere theories*. Our account follows most closely Power's account [Pow00]. A key difference is that we keep to a symmetric closed enrichment, rather than the more general biclosed structure Power considers. The symmetric account dates further back to Kelly [Kel82b].

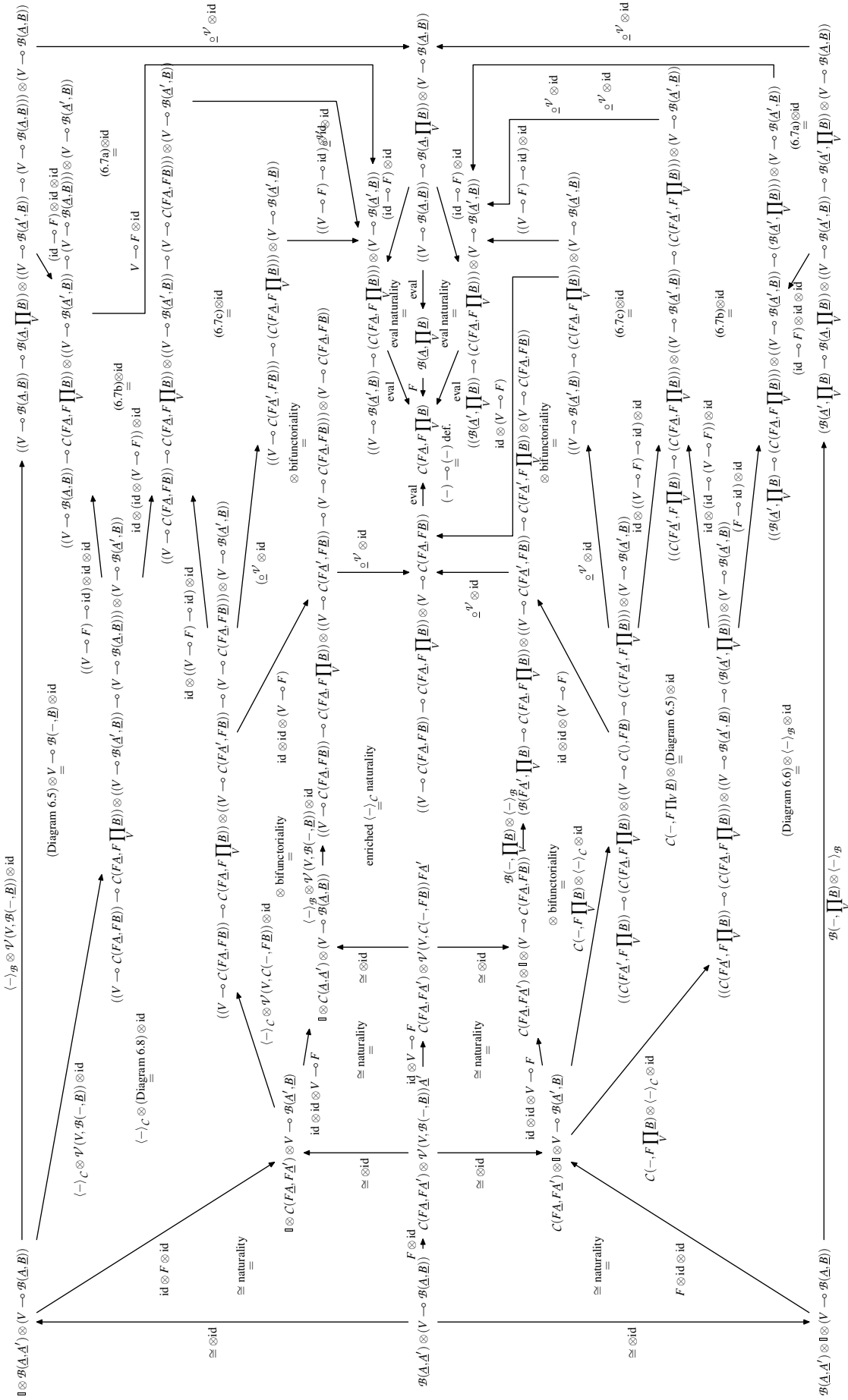
First, we assume the enriching category has the following structure (see Power's account [Pow00, Section 2]).

Definition 6.9. *Let λ be a regular cardinal. A λ -Power category \mathcal{V} is a symmetric monoidal closed category $\langle |\mathcal{V}|, \mathbb{1}, \otimes, \multimap, \text{unit}_{\text{left}}, \text{unit}_{\text{right}}, \text{assoc}, \text{symm} \rangle$ such that:*

- $|\mathcal{V}|$ is locally λ -presentable; and
- $\mathbb{1}$ is λ -presentable, and if V, W are λ -presentable, then so is $V \otimes W$.

We call an \aleph_0 -Power category a *finitary* Power category, and similarly we call an \aleph_1 -Power category a *countably infinitary* Power category, or simply a countable Power category (although it is may *not* be a countable category, i.e., have countably many morphisms). We restrict our attention to the following two cases:

Example 6-8. The category **Set** with the cartesian closed structure is a finite Power category. Indeed, in Example 5-7 we saw that **Set** is locally finitely presentable. As $\mathbb{1} = \{*\}$ is finite, and the product of finite sets is finite as well, **Set** is a finite Power category. □



Example 6-9. The category $\omega\mathbf{CPO}$ with the cartesian closed structure is a countable Power category. Indeed, Theorem 5.33 shows that $\omega\mathbf{CPO}$ is locally countably presentable. As $\mathbb{1} = \{\star\}$ is finite, it is countably presentable by Example 5-5. Let V, W be two ω -cpos with countable domain presentations $\mathcal{P}_V = \langle A, I, \sqsubseteq_V \rangle, \mathcal{P}_W = \langle B, J, \sqsubseteq_W \rangle$, respectively. Define:

$$\begin{aligned} C &:= A \times B \\ K &:= \{ \langle \langle a_n, b \rangle \rangle_n \mid \langle a_n \rangle \in I, b \in B \} \cup \{ \langle \langle a, b_n \rangle \rangle_n \mid a \in A, \langle b_n \rangle \in J \} \\ \sqsubseteq_U &:= \{ \langle \langle a_n, b \rangle \rangle \sqsubseteq \langle \langle a'_n, b \rangle \rangle \mid \langle a_n \rangle \sqsubseteq_V \langle a'_n \rangle, b \in B \} \cup \\ &\quad \{ \langle \langle a, b_n \rangle \rangle \sqsubseteq \langle \langle a, b'_n \rangle \rangle \mid a \in A, \langle b_n \rangle \sqsubseteq_W \langle b'_n \rangle \} \end{aligned}$$

then $\mathcal{P}_U = \langle C, K, \sqsubseteq_U \rangle$ is a countable domain presentation, and $U := V \times W$ is the initial \mathcal{P}_U -model. Indeed, setting $U \llbracket \langle a, b \rangle \rrbracket := \langle V \llbracket a \rrbracket, W \llbracket b \rrbracket \rangle$ exhibits U as a \mathcal{P}_U -model. For initiality, take any other model U' . For each $a \in A$, due to the presence of constraints of the form $\langle \langle a, b_n \rangle \rangle \sqsubseteq \langle \langle a, b'_n \rangle \rangle$, the function $g_a := \lambda b. U' \llbracket \langle a, b \rangle \rrbracket$ exhibits a \mathcal{P}_W -model $\langle U', g_a \rangle$, hence we have a unique Scott-continuous $f_a : W \rightarrow U'$ factoring $W \llbracket - \rrbracket$ through g_a . Due to constraints of the form $\langle \langle a_n, b \rangle \rangle \sqsubseteq \langle \langle a'_n, b \rangle \rangle$, the map $g' := \lambda a. f_a$ from A to U' is monotone, hence we have a unique Scott-continuous $h' : V \rightarrow (U')^V$ factoring $V \llbracket - \rrbracket$ through g' . Uncurrying then yields the unique $h : V \times W \xrightarrow{\lambda V. h'} U'$ factoring $U \llbracket - \rrbracket$ through $U' \llbracket - \rrbracket$. Thus $V \times W$ has a countable presentation, and $\omega\mathbf{CPO}$ is a countable Power category. \square

When a category has all (co)powers of objects by λ -presentable objects, we say it has all λ -(co)powers. Denote by $\mathbf{Pres}_\lambda \mathcal{V}$ the full subcategory of \mathcal{V} consisting of the λ -presentable objects. When \mathcal{V} is a λ -Power category, $\mathbf{Pres}_\lambda \mathcal{V}$ contains $\mathbb{1}$ and $V \otimes W$ for every λ -presentable V and W . Thus, $\mathbf{Pres}_\lambda \mathcal{V}$ has all λ -copowers, hence the opposite \mathcal{V} -category $\mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}$ has all λ -powers. Note that, in light of Proposition 5.11, $\mathbf{Pres}_\lambda \mathcal{V}$ is essentially small, i.e., it is \mathcal{V} -equivalent to a small category. Note that Power [Pow00] prefers to work with small categories, hence instead uses the skeleton of our $\mathbf{Pres}_\lambda \mathcal{V}$. As we will see later, it will be useful to have all λ -presentable objects rather than a set of representatives, although it will complicate some proofs.

We say that a \mathcal{V} -functor $F : \mathcal{B} \rightarrow \mathcal{C}$ is *identity-on-objects* if $\text{Ob}(\mathcal{B}) = \text{Ob}(\mathcal{C})$ and $F \underline{B} = \underline{B}$ for all \mathcal{B} -objects \underline{B} . For example, the \mathcal{V} -functor F from \mathcal{V} to the Kleisli \mathcal{V} -category \mathcal{V}_T that we study in Example 6-7 is identity-on-objects.

Definition 6.10. Let \mathcal{V} be a λ -Power category. An enriched λ -Lawvere \mathcal{V} -theory \mathcal{L} is a pair $\langle |\mathcal{L}|, \mathcal{L} \llbracket - \rrbracket \rangle$ where:

- $|\mathcal{L}|$ is a \mathcal{V} category with λ -powers; and
- $\mathcal{L}[-] : \mathbf{Pres}_\lambda^{\text{op}} \mathcal{V} \rightarrow |\mathcal{L}|$ is a \mathcal{V} -functor that is λ -power preserving and identity-on-objects.

Example 6-10 (see Power [Pow00, Construction 3.3]). Let T be a monad enriched in a λ -Power category \mathcal{V} , and F the \mathcal{V} -functor from \mathcal{V} to the Kleisli \mathcal{V} -category \mathcal{V}_T we studied in Example 6-7. Denote by \mathcal{V}_T^λ the \mathcal{V} -category induced by the λ -presentable objects in \mathcal{V} . Then F restricts to an identity-on-objects λ -copower preserving \mathcal{V} -functor $F^\lambda : \mathbf{Pres}_\lambda \mathcal{V} \rightarrow \mathcal{V}_T^\lambda$. By moving to the opposite categories, we obtain an identity-on-objects λ -power preserving \mathcal{V} -functor from $\mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}$ to $(\mathcal{V}_T^\lambda)^{\text{op}}$, and consequently a λ -Lawvere \mathcal{V} -theory which we denote by \mathcal{L}_T .

Explicitly, \mathcal{L}_T is given as follows. The objects of $|\mathcal{L}_T|$, as in any λ -Lawvere \mathcal{V} -theory, are the λ -presentable objects of \mathcal{V} . The hom-objects $|\mathcal{L}_T|(V, W)$ are given by $\mathcal{V}(W, TV)$. Identities are given by the monadic unit η , and the composition by:

$$\circ_{A,B,C} : \mathcal{V}(C, TB) \otimes \mathcal{V}(B, TA) \xrightarrow{\text{id} \otimes T} \mathcal{V}(C, TB) \otimes \mathcal{V}(TB, T^2A) \xrightarrow{\circ_{\mathcal{V}}^{\text{osymm}}} \mathcal{V}(C, T^2A) \xrightarrow{\mathcal{V}(A, \mu)} \mathcal{V}(C, TA)$$

The powers $\prod_V U$ are given as \mathcal{V} copowers, i.e., the tensor product $V \otimes U$.

Finally, the object map of the \mathcal{V} -functor $\mathcal{L}_T[-]$ is the identity map, and its action on hom-objects is given by post-composing with the unit η :

$$\mathcal{L}_T[-] : \mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}(V, W) = \mathcal{V}(W, V) \xrightarrow{\mathcal{V}(W, \eta_V)} \mathcal{V}(W, TV) = |\mathcal{L}_T|(V, W)$$

Thus, we have a λ -Lawvere \mathcal{V} -theory. □

In the sequel, we will present more syntactic and concrete ways to define Lawvere theories.

Definition 6.11. Let \mathcal{L} be a λ -Lawvere \mathcal{V} -Theory and C a \mathcal{V} -category with λ -powers. An \mathcal{L} -model \mathcal{M} in C is a \mathcal{V} -functor $\mathcal{M} : |\mathcal{L}| \rightarrow C$ preserving all λ -powers.

Given two models $\mathcal{M}, \mathcal{M}'$, an \mathcal{L} -homomorphism h is a \mathcal{V} -natural transformation from \mathcal{M} to \mathcal{M}' .

We denote by $\mathbf{Mod}(\mathcal{L}, C)$ the full subcategory of the (ordinary) category of \mathcal{V} -functors from $|\mathcal{L}|$ to C and \mathcal{V} -natural transformations between them consisting of the \mathcal{L} -models.

Let \mathcal{M} be an \mathcal{L} -model. Note that for any λ -presentable V :

$$\begin{array}{ccccc} \text{identity-on-objects} & & \text{power preservation} & & \text{power preservation} \\ \downarrow & & \downarrow & & \downarrow \\ \mathcal{M}(V) = \mathcal{M}(\mathcal{L}[V]) \cong \mathcal{M}(\mathcal{L}[V \otimes \mathbb{1}]) & \cong & \mathcal{M}\left(\prod_V \mathcal{L}[\mathbb{1}]\right) & = & \mathcal{M}\left(\prod_V \mathbb{1}\right) \cong \prod_V \mathcal{M}(\mathbb{1}) \\ & & \uparrow & & \\ & & \text{identity-on-objects} & & \end{array}$$

Therefore, the object map of \mathcal{M} is uniquely determined by its action on $\mathbb{1}$. We call the object $\mathcal{M}(\mathbb{1})$ the *carrier* of \mathcal{M} . In fact, we have the following functor:

$$\begin{aligned} U_{\mathcal{L}} : \mathbf{Mod}(\mathcal{L}, \mathcal{C}) &\rightarrow |\mathcal{C}|_{\circ} \\ U_{\mathcal{L}} : \mathcal{M} &\mapsto \mathcal{M}(\mathbb{1}) \\ U_{\mathcal{L}} : (\mathcal{M} \xrightarrow{h} \mathcal{M}') &\mapsto (\mathbb{1} \xrightarrow{h_{\mathbb{1}}} \mathcal{C}(\mathcal{M}(\mathbb{1}), \mathcal{M}'(\mathbb{1}))) \end{aligned}$$

Note how we regarded $\mathbf{Mod}(\mathcal{L}, \mathcal{C})$ as an ordinary category. In fact, $\mathbf{Mod}(\mathcal{L}, \mathcal{C})$ is a \mathcal{V} -category [Pow00, Definition 3.2]. To explicitly describe this structure, we need to introduce enriched functor categories, which requires us to review additional notions from enriched category theory. As we will not explicitly use this enriched structure in what follows, we treat $\mathbf{Mod}(\mathcal{L}, \mathcal{C})$ merely as an ordinary category.

Power [Pow00] generalised the usual bijection between finitary monads and Lawvere theories to the enriched case in the following manner:

Theorem 6.12 ([Pow00, Construction 3.3 and Theorems 3.4,4.1]). *Let \mathcal{V} be a λ -Power category. For every λ -Lawvere \mathcal{V} -theory \mathcal{L} , the functor $U_{\mathcal{L}} : \mathbf{Mod}(\mathcal{L}, \mathcal{V}) \rightarrow \mathcal{V}$ is monadic, i.e., $U_{\mathcal{L}}$ has a left adjoint $F_{\mathcal{L}} : \mathcal{V} \rightarrow \mathbf{Mod}(\mathcal{L}, \mathcal{V})$, and the Eilenberg-Moore category for the resulting monad $T_{\mathcal{L}} = U_{\mathcal{L}}F_{\mathcal{L}}$ over \mathcal{V} is equivalent to $\mathbf{Mod}(\mathcal{L}, \mathcal{V})$. Furthermore, $T_{\mathcal{L}}$ is \mathcal{V} -enriched and λ -ranked. We call $F_{\mathcal{L}}$ the free model functor for \mathcal{L} , and $T_{\mathcal{L}}$ the free model monad for \mathcal{L} .*

Using Theorem 5.19, we deduce the following:

Theorem 6.13 ([HPP06]). *For every λ -Lawvere \mathcal{V} -theory \mathcal{L} , the category of \mathcal{L} -models $\mathbf{Mod}(\mathcal{L}, \mathcal{V})$ is locally λ -presentable.*

Plotkin and Power, in unpublished work, use this theorem to show indirectly that $\omega\mathbf{CPO}$ is locally countably presentable by exhibiting it as a category of \mathcal{L} -models for a suitable countable Lawvere \mathbf{Pos} -enriched theory.

Next, we follow Power [Pow00] and construct a category of enriched Lawvere theories:

Definition 6.14. *Let $\mathcal{L}, \mathcal{L}'$ be two λ -Lawvere \mathcal{V} -theories. A morphism \mathfrak{T} from \mathcal{L} to \mathcal{L}' is a \mathcal{V} functor $\mathfrak{T} : |\mathcal{L}| \rightarrow |\mathcal{L}'|$ that is identity-on-objects, λ -power preserving, and satisfies*

$$\begin{array}{ccc} & \mathbf{Pres}_{\lambda}^{\text{op } \mathcal{V}} & \\ \mathcal{L}[-] & \begin{array}{c} \nearrow \\ = \\ \searrow \end{array} & \mathcal{L}'[-] \\ & |\mathcal{L}| \xrightarrow{\mathfrak{T}} |\mathcal{L}'| & \end{array}$$

Taking λ -Lawvere \mathcal{V} -theories as objects and their morphisms with the usual functor composition and identities yields an ordinary category $\mathbf{Law}_\lambda \mathcal{V}$.

Example 6-11. Let $m : T \rightarrow T'$ be a \mathcal{V} -monad morphism over a λ -Power category \mathcal{V} . We saw in Example 6-7 that m induces an identity-on-objects, copower preserving \mathcal{V} -functor $M : \mathcal{V}_T \rightarrow \mathcal{V}_{T'}$. Therefore, by restricting to the λ -presentable objects and taking the opposite categories, this \mathcal{V} -functor restricts to a morphism of λ -Lawvere \mathcal{V} -theories \mathcal{L}_m from the theory \mathcal{L}_T to the theory $\mathcal{L}_{T'}$, presented in Example 6-10.

Denote by $\mathcal{V}\text{-Monads}$ the (ordinary) category of \mathcal{V} -monads and \mathcal{V} -monad morphisms. Then the construction \mathcal{L}_- is an ordinary functor from the category $\mathcal{V}\text{-Monads}$ to the category $\mathbf{Law}_\lambda \mathcal{V}$. \square

Denote by $\mathcal{V}\text{-Monads}_\lambda$ the full subcategory of $\mathcal{V}\text{-Monads}$ consisting of the λ -ranked monads. The functor \mathcal{L}_- restricts to a functor from $\mathcal{V}\text{-Monads}_\lambda$ to $\mathbf{Law}_\lambda \mathcal{V}$.

Theorem 6.15 ([Pow00, Theorem 4.3]). *Let \mathcal{V} be a λ -Power category. The functor $\mathcal{L}_- : \mathcal{V}\text{-Monads}_\lambda \rightarrow \mathbf{Law}_\lambda \mathcal{V}$ is an equivalence of categories. Its adjoint weak inverse from $\mathbf{Law}_\lambda \mathcal{V}$ to $\mathcal{V}\text{-Monads}_\lambda$ maps every λ -Lawvere \mathcal{V} -theory \mathcal{L} to the monad $T_{\mathcal{L}}$ from Theorem 6.12. In particular:*

- for every λ -Lawvere \mathcal{V} -theory \mathcal{L} , we have $\mathcal{L}_{T_{\mathcal{L}}} \cong \mathcal{L}$; and
- for every λ -ranked \mathcal{V} -monad T over \mathcal{V} , we have $T_{\mathcal{L}_T} \cong T$.

In the following we occasionally need the explicit description of T_- on a theory \mathcal{L} . Therefore, we recount a known technique to reconstruct, up to isomorphism, the functor T_- from the definition of \mathcal{L}_- , using the previous theorem. Denote by $\theta : \mathcal{L}_{T_-} \rightarrow \text{id}$ the counit of the equivalence.

Given a \mathcal{V} -category \mathcal{C} and arrows

$$\begin{aligned} \underline{f} : \mathbb{1} &\multimap \mathcal{C}(\underline{A}, \underline{B}) \\ \underline{g} : \mathbb{1} &\multimap \mathcal{C}(\underline{C}, \underline{D}) \end{aligned}$$

define the composition arrow (cf. Kelly [Kel82a, Section 1.6]) as follows:

$$\begin{aligned} \mathcal{C}(\underline{f}, \underline{g}) : \mathcal{C}(\underline{B}, \underline{C}) &\xrightarrow{\cong} \mathbb{1} \otimes \mathcal{C}(\underline{B}, \underline{C}) \otimes \mathbb{1} \xrightarrow{g \otimes \text{id} \otimes f} \mathcal{C}(\underline{B}, \underline{C}) \otimes \mathcal{C}(\underline{B}, \underline{C}) \otimes \mathcal{C}(\underline{A}, \underline{B}) \\ &\xrightarrow{\text{id} \otimes \circ} \mathcal{C}(\underline{B}, \underline{C}) \otimes \mathcal{C}(\underline{A}, \underline{C}) \xrightarrow{\circ} \mathcal{C}(\underline{A}, \underline{D}) \end{aligned}$$

For a Lawvere theory \mathcal{L} , and any arrow $f : A \leftarrow B$ in $\mathbf{Pres}_\lambda \mathcal{V}$, we define the arrow:

$$\mathcal{L}[\underline{f}] : \mathbb{1} \xrightarrow{f} \mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}(A, B) \xrightarrow{\mathcal{L}[\underline{-}]} |\mathcal{L}|(A, B)$$

Straightforward calculations verify the following identities:

$$\mathcal{L}[\text{id}_A] = \underline{\text{id}}_A^{|\mathcal{L}|} \quad (6.9a)$$

$$|\mathcal{L}|(\mathcal{L}[\underline{f}], \mathcal{L}[\underline{h}]) \circ \mathcal{L}[\underline{g}] = \mathcal{L}[\underline{f} \circ \underline{g} \circ \underline{h}] \quad (6.9b)$$

$$\begin{array}{ccc} C(\underline{B}, \underline{C}) \otimes C(\underline{A}, \underline{B}) & \xrightarrow{\underline{\circ}} & C(\underline{A}, \underline{C}) \\ \downarrow C(\underline{\text{id}}_{\underline{B}}, \underline{g}) \otimes C(\underline{f}, \underline{\text{id}}_{\underline{B}}) & = & \downarrow C(\underline{f}, \underline{g}) \\ C(\underline{B}, \underline{C}') \otimes C(\underline{A}', \underline{B}) & \xrightarrow{\underline{\circ}} & C(\underline{A}', \underline{C}') \end{array} \quad (6.9c)$$

$$\begin{array}{ccc} C(\underline{B}, \underline{C}) \otimes C(\underline{A}, \underline{B}) & \xrightarrow{\text{id} \otimes C(\underline{\text{id}}_{\underline{A}}, \underline{f})} & C(\underline{B}, \underline{C}) \otimes C(\underline{A}, \underline{B}') \\ \downarrow C(\underline{f}, \underline{\text{id}}_{\underline{C}}) \otimes \text{id} & = & \downarrow \underline{\circ} \\ C(\underline{B}', \underline{C}) \otimes C(\underline{A}, \underline{B}) & \xrightarrow{\underline{\circ}} & C(\underline{A}, \underline{C}) \end{array} \quad (6.9d)$$

$$\begin{array}{ccc} |\mathcal{L}|(\underline{A}, \underline{B}) & \xrightarrow{|\mathcal{L}|(\mathcal{L}[\underline{f}], \mathcal{L}[\underline{g}])} & |\mathcal{L}|(\underline{A}', \underline{B}') \\ \downarrow \mathfrak{T} & = & \downarrow \mathfrak{T} \\ |\widehat{\mathcal{L}}|(\underline{A}, \underline{B}) & \xrightarrow{|\widehat{\mathcal{L}}|(\widehat{\mathcal{L}}[\underline{f}], \mathcal{L}[\underline{g}])} & |\widehat{\mathcal{L}}|(\underline{A}', \underline{B}') \end{array} \quad (6.9e)$$

Let \mathcal{L} be any Lawvere theory.

First, we recover the action of $T_{\mathcal{L}}$ on objects. Consider any \mathcal{V} -object A , and a λ -directed diagram $D : I \rightarrow \mathcal{V}$, whose vertices are λ -presentable, for which $A = \text{Colim} D$.

We would like to calculate as follows:

$$\begin{aligned}
& \lambda\text{-ranked monad} \\
& \downarrow \\
T_{\mathcal{L}}A &= \text{Colim } T_{\mathcal{L}} \circ D \cong \text{Colim}(\mathbb{1} \multimap T_{\mathcal{L}}D(-)) = \text{Colim } \mathbf{Pres}_{\lambda}\mathcal{V}(\mathbb{1}, T_{\mathcal{L}}D(-)) \\
& \quad \text{Theorem 6.15} \\
& \quad \downarrow \\
&= \text{Colim } |\mathcal{L}_{T_{\mathcal{L}}}|(D(-), \mathbb{1}) \cong \text{Colim } |\mathcal{L}|(D(-), \mathbb{1})
\end{aligned}$$

However, we need to take some care with the last two transitions, and clarify what we mean by $|\mathcal{L}|(D(-), \mathbb{1})$ when the diagram is applied to morphisms in I .

Let \mathcal{L} be any Lawvere theory. We define an ordinary functor

$$|\mathcal{L}|(\mathcal{L}[-], \mathbb{1}) : \mathbf{Pres}_{\lambda}\mathcal{V} \rightarrow \mathcal{V}$$

The object map maps every λ -presentable object A to $|\mathcal{L}|(A, \mathbb{1})$. Given any morphism $f : A \rightarrow A'$ between λ -presentable objects, we have a \mathcal{V} -morphism $\mathcal{L}[[f]] : \mathbb{1} \rightarrow |\mathcal{L}|(A', A)$. Thus, we define the morphism map by the enriched precomposition with $\mathcal{L}[[f]]$:

$$|\mathcal{L}|(\mathcal{L}[[f]], \mathbb{1}) : |\mathcal{L}|(A, \mathbb{1}) \rightarrow |\mathcal{L}|(A', \mathbb{1})$$

In light of (6.9a) and (6.9b), $|\mathcal{L}|(\mathcal{L}[-], \mathbb{1})$ is a functor. For any λ -presentable object A , define an isomorphism $\alpha_A : T_{\mathcal{L}}A \xrightarrow{\cong} |\mathcal{L}|(A, \mathbb{1})$ by:

$$\alpha_A : T_{\mathcal{L}}A \cong \mathbb{1} \multimap T_{\mathcal{L}}A = \mathbf{Pres}_{\lambda}\mathcal{V}(\mathbb{1}, T_{\mathcal{L}}A) = |\mathcal{L}_{T_{\mathcal{L}}}|(A, \mathbb{1}) \xrightarrow{\theta} |\mathcal{L}|(A, \mathbb{1})$$

Because θ is a morphism of Lawvere theories, it follows by Diagram 6.9e that θ is natural in A . Consequently, α is a natural isomorphism between the restriction of $T_{\mathcal{L}}$ to the λ -presentable objects, i.e., as a functor from $\mathbf{Pres}_{\lambda}\mathcal{V}$ to \mathcal{V} , and the functor $|\mathcal{L}|(\mathcal{L}[-], \mathbb{1})$.

Consider any diagram $D : I \rightarrow \mathcal{V}$, whose vertices are λ -presentable, for which $A = \text{Colim } D$. Let $D_{\mathcal{L}} : I \rightarrow \mathcal{V}$ be the diagram $|\mathcal{L}|(\mathcal{L}[-], \mathbb{1}) \circ D$. Then precomposing α with D yields an isomorphism $\alpha : T_{\mathcal{L}} \circ D \xrightarrow{\cong} D_{\mathcal{L}}$. Thus we have:

$$T_{\mathcal{L}}A = T_{\mathcal{L}} \text{Colim } D = \text{Colim } T_{\mathcal{L}}D \cong \text{Colim } D_{\mathcal{L}}$$

Similarly, to recover the action of $T_{\mathcal{L}}$ on morphisms, we use Proposition 5.21. Consider any λ -directed diagram $D : I \rightarrow \mathcal{V}^{\mathbb{S}}$ whose vertices are all λ -presentable. The naturality of α yields a natural isomorphism $\alpha^{\mathbb{S}} : T^{\mathbb{S}} \rightarrow |\mathcal{L}|(\mathcal{L}[-], \mathbb{1})^{\mathbb{S}}$. Define $D_{\mathcal{L}}$ as the composition $|\mathcal{L}|(\mathcal{L}[-], \mathbb{1})^{\mathbb{S}} \circ D$. We then have:

$$\begin{aligned}
& \text{Proposition 5.21} \\
& \downarrow \\
T_{\mathcal{L}}f &\cong \text{Colim } T_{\mathcal{L}}^{\mathbb{S}} \circ D \cong \text{Colim } D_{\mathcal{L}}^{\mathbb{S}}
\end{aligned}$$

Thus we recover the object map of \mathcal{L}_- . Let $\mathfrak{T} : \mathcal{L} \rightarrow \widehat{\mathcal{L}}$ be a morphism of Lawvere theories.

Consider any \mathcal{V} -object A , and a λ -directed diagram $D : I \rightarrow \mathcal{V}$, whose vertices are λ -presentable, for which $A = \text{Colim} D$. We will invoke Proposition 5.22 on $T_{\mathfrak{T}}$. Thus we construct $D_{T_{\mathfrak{T}}}$ as in Proposition 5.22:

$$\begin{aligned} D_{T_{\mathfrak{T}}}i &: T_{\mathcal{L}}Di \xrightarrow{(T_{\mathfrak{T}})Di} T_{\widehat{\mathcal{L}}}Di & i \in \text{Ob}(I) \\ D_{T_{\mathfrak{T}}}f &= \left\langle T_{\mathcal{L}}Di \xrightarrow{T_{\mathcal{L}}Df} T_{\mathcal{L}}Dj, T_{\widehat{\mathcal{L}}}Di \xrightarrow{T_{\widehat{\mathcal{L}}}Df} T_{\widehat{\mathcal{L}}}Dj \right\rangle & f : i \rightarrow j \text{ in } I \end{aligned}$$

The components of \mathfrak{T} form a natural transformation $\mathfrak{T}_{A, \mathbb{1}} : |\mathcal{L}|(A, \mathbb{1}) \rightarrow |\widehat{\mathcal{L}}|(A, \mathbb{1})$. Thus we construct the diagram $D_{\mathfrak{T}}$ as in Proposition 5.22. I.e., for $i \in \text{Ob}(I)$,

$$D_{\mathfrak{T}}i : |\mathcal{L}|(Di, \mathbb{1}) \xrightarrow{\mathfrak{T}} |\widehat{\mathcal{L}}|(Di, \mathbb{1}) \quad (6.10)$$

and for $f : i \rightarrow j$ in I :

$$D_{\mathfrak{T}}f = \left\langle |\mathcal{L}|(Di, \mathbb{1}) \xrightarrow{|\mathcal{L}|(\mathcal{L}[Df], \mathbb{1})} |\mathcal{L}|(Dj, \mathbb{1}), |\widehat{\mathcal{L}}|(Di, \mathbb{1}) \xrightarrow{|\widehat{\mathcal{L}}|(\widehat{\mathcal{L}}[Df], \mathbb{1})} |\widehat{\mathcal{L}}|(Dj, \mathbb{1}) \right\rangle$$

The two isomorphisms $\alpha_{\mathcal{L}} : T_{\mathcal{L}} \xrightarrow{\cong} |\mathcal{L}|(\mathcal{L}[-], \mathbb{1})$ and $\alpha_{\widehat{\mathcal{L}}} : T_{\widehat{\mathcal{L}}} \xrightarrow{\cong} |\widehat{\mathcal{L}}|(\widehat{\mathcal{L}}[-], \mathbb{1})$ then induce an isomorphism $\hat{\alpha}$ from $D_{T_{\mathfrak{T}}}$ to $D_{\mathfrak{T}}$. Therefore:

$$\begin{array}{c} \text{Proposition 5.22} \\ \downarrow \\ (T_{\mathfrak{T}})_A \cong \text{Colim } D_{T_{\mathfrak{T}}} \cong \text{Colim } D_{\mathfrak{T}} \end{array}$$

Thus we reconstructed the functor T_- .

The category $\mathbf{Law}_{\lambda} \mathcal{V}$ is locally presentable¹, but the proof lies beyond the scope of this thesis. From its local presentability we deduce the following.

Theorem 6.16 (for a special case, see Hyland et. al [HPP06, Theorem 6]). *The category $\mathbf{Law}_{\lambda} \mathcal{V}$ is cocomplete.*

Therefore, we also know that $\mathbf{Law}_{\lambda} \mathcal{V}$ is complete. However, in the sequel we will make use of the fact that the limits in $\mathbf{Law}_{\lambda} \mathcal{V}$ are given *componentwise*:

Theorem 6.17. *Let $D : J \rightarrow \mathbf{Law}_{\lambda} \mathcal{V}$ be a small diagram. Every limiting cone $\langle \mathcal{L}, F \rangle$ is uniquely determined by a collection of component limiting cones $\langle |\mathcal{L}|(\underline{A}, \underline{B}), F_{\underline{A}, \underline{B}} \rangle$ for the component diagrams $|D(-)|(\underline{A}, \underline{B})$.*

The remainder of the Lawvere theory structure is given as the unique maps satisfying, for all $\underline{A}, \underline{B}$:

¹John Power, private communication, 2012.

$$\begin{array}{ccc}
\begin{array}{c} \text{id}^{|L|} \\ \swarrow \\ |L|(A, A) \\ \searrow \\ \text{id}^{|Di|} \end{array} & \begin{array}{c} |L|(A, A) \\ \downarrow F^i \\ |Di|(F^i A, F^i A) \end{array} & \\
= & & \\
\begin{array}{c} |L|(B, C) \otimes |L|(A, B) \\ \downarrow F^i \otimes F^i \\ |Di|(F^i B, F^i C) \otimes |Di|(F^i A, F^i B) \end{array} & \begin{array}{c} \xrightarrow{\circ^{|L|}} \\ |L|(A, C) \\ \downarrow F^i \\ |Di|(F^i A, F^i C) \end{array} & \\
& = & \\
& \begin{array}{c} \xrightarrow{\circ^{|Di|}} \\ |Di|(F^i A, F^i C) \end{array} &
\end{array}$$

$$\begin{array}{ccc}
& & |L|(A, B) \\
& \swarrow & \downarrow \\
\text{Pres}_\lambda^{\text{op}} \mathcal{V}(A, B) & = & F^i \\
& \searrow & \downarrow \\
& & |Di|(F^i A, F^i B)
\end{array}$$

$$\begin{array}{ccc}
V \multimap |L|(A, B) & \xrightarrow{\langle - \rangle_{|L|}} & |L|(A, \prod_V B) \\
\downarrow V \multimap F^i & = & \downarrow F^i \\
V \multimap |Di|(A, B) & \xrightarrow{\langle - \rangle_{|Di|}} & |Di|(A, \prod_V B)
\end{array}
\qquad
\begin{array}{ccc}
V \multimap |L|(A, B) & \xleftarrow{\langle - \rangle_{|L|}^{-1}} & |L|(A, \prod_V B) \\
\downarrow V \multimap F^i & = & \downarrow F^i \\
V \multimap |Di|(A, B) & \xleftarrow{\langle - \rangle_{|Di|}^{-1}} & |Di|(A, \prod_V B)
\end{array}$$

Proof

First, assume a choice of limiting cones $\langle |L|(A, B), F_{A, B} \rangle$ for the component diagrams $|D(-)| (A, B)$. Straightforward calculation shows that we indeed have, for every A, B , cones for each of the diagrams above. For example, for every $f : i \rightarrow j$, we have

$$\begin{array}{ccccc}
& & V \multimap |Di|(A, B) & \xrightarrow{\langle - \rangle_{|Di|}} & |Di|(A, \prod_V B) \\
& \nearrow V \multimap F^i & \downarrow & \downarrow V \multimap Df & \downarrow Df \\
V \multimap |L|(A, B) & \xrightarrow{V \multimap (\text{cone})} & & \xrightarrow{\text{Lemma 6.7}} & \\
& \searrow V \multimap F^j & V \multimap |Dj|(A, B) & \xrightarrow{\langle - \rangle_{|Dj|}} & |Dj|(A, \prod_V B)
\end{array}$$

Thus $\langle - \rangle_{|L|}$ is uniquely defined. Using the fact that $\langle - \rangle_{|Di|}$ are isomorphisms, $\langle - \rangle_{|L|}^{-1}$ is uniquely defined. Similar arguments establish the unique existence of the other structure maps.

Lemma 6.8 completes the construction. From Lemma 6.8(1) we have, for all i :

$$\begin{array}{c}
(|\mathcal{L}|(\underline{C}, \underline{D}) \otimes |\mathcal{L}|(\underline{B}, \underline{C})) \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \xrightarrow{\cong} |\mathcal{L}|(\underline{C}, \underline{D}) \otimes (|\mathcal{L}|(\underline{B}, \underline{C}) \otimes |\mathcal{L}|(\underline{A}, \underline{B})) \\
\downarrow \underline{\circ}^{|\mathcal{L}|} \otimes \text{id} \qquad \qquad \qquad \downarrow \text{id} \otimes \underline{\circ}^{|\mathcal{L}|} \\
|\mathcal{L}|(\underline{B}, \underline{D}) \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \qquad \qquad \qquad |\mathcal{L}|(\underline{C}, \underline{D}) \otimes |\mathcal{L}|(\underline{A}, \underline{C}) \\
\downarrow \underline{\circ}^{|\mathcal{L}|} \qquad \qquad \qquad \downarrow \underline{\circ}^{|\mathcal{L}|} \\
\mathbb{1} \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \qquad \qquad \qquad |\mathcal{L}|(\underline{A}, \underline{D}) \qquad \qquad \qquad |\mathcal{L}|(\underline{A}, \underline{B}) \otimes \mathbb{1} \\
\downarrow \underline{\text{id}}_B^{|\mathcal{L}|} \otimes \text{id} \qquad \qquad \qquad \downarrow \underline{\text{id}}_A^{|\mathcal{L}|} \\
|\mathcal{L}|(\underline{B}, \underline{B}) \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \xrightarrow{\underline{\circ}^{|\mathcal{L}|}} |\mathcal{L}|(\underline{A}, \underline{B}) \xleftarrow{\underline{\circ}^{|\mathcal{L}|}} |\mathcal{L}|(\underline{A}, \underline{B}) \otimes |\mathcal{L}|(\underline{A}, \underline{A})
\end{array}$$

$|Di|(\underline{A}, \underline{D})$ (top node)
 $|Di|(\underline{A}, \underline{B})$ (middle nodes)
 F^i (functors between nodes)

Therefore, as $\langle |\mathcal{L}|(\underline{A}, \underline{B}), F_{\underline{A}, \underline{B}} \rangle$ is a limiting cone, we deduce that:

$$\begin{array}{c}
(|\mathcal{L}|(\underline{C}, \underline{D}) \otimes |\mathcal{L}|(\underline{B}, \underline{C})) \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \xrightarrow{\cong} |\mathcal{L}|(\underline{C}, \underline{D}) \otimes (|\mathcal{L}|(\underline{B}, \underline{C}) \otimes |\mathcal{L}|(\underline{A}, \underline{B})) \\
\downarrow \underline{\circ}^{|\mathcal{L}|} \otimes \text{id} \qquad \qquad \qquad \downarrow \text{id} \otimes \underline{\circ}^{|\mathcal{L}|} \\
|\mathcal{L}|(\underline{B}, \underline{D}) \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \qquad \qquad \qquad |\mathcal{L}|(\underline{C}, \underline{D}) \otimes |\mathcal{L}|(\underline{A}, \underline{C}) \\
\downarrow \underline{\circ}^{|\mathcal{L}|} \qquad \qquad \qquad \downarrow \underline{\circ}^{|\mathcal{L}|} \\
\mathbb{1} \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \qquad \qquad \qquad |\mathcal{L}|(\underline{A}, \underline{D}) \qquad \qquad \qquad |\mathcal{L}|(\underline{A}, \underline{B}) \otimes \mathbb{1} \\
\downarrow \underline{\text{id}}_B^{|\mathcal{L}|} \otimes \text{id} \qquad \qquad \qquad \downarrow \underline{\text{id}}_A^{|\mathcal{L}|} \\
|\mathcal{L}|(\underline{B}, \underline{B}) \otimes |\mathcal{L}|(\underline{A}, \underline{B}) \xrightarrow{\underline{\circ}^{|\mathcal{L}|}} |\mathcal{L}|(\underline{A}, \underline{B}) \xleftarrow{\underline{\circ}^{|\mathcal{L}|}} |\mathcal{L}|(\underline{A}, \underline{B}) \otimes |\mathcal{L}|(\underline{A}, \underline{A})
\end{array}$$

$|Di|(\underline{A}, \underline{D})$ (top node)
 $|Di|(\underline{A}, \underline{B})$ (middle nodes)
 F^i (functors between nodes)

Thus $|\mathcal{L}|$ is a \mathcal{V} -category, and F^i become \mathcal{V} -functors from $|\mathcal{L}|$ to $|Di|$.

Therefore, we can invoke Lemma 6.8(2). From

$$\begin{array}{ccc}
& |\mathcal{L}|(\underline{A}, \prod_V \underline{B}) & \xlongequal{\quad} & |\mathcal{L}|(\underline{A}, \prod_V \underline{B}) \\
\langle - \rangle_{|\mathcal{L}|} \nearrow & & & \langle - \rangle_{|\mathcal{L}|} \searrow \\
V \multimap |\mathcal{L}|(\underline{A}, \underline{B}) & \xlongequal{\quad} & V \multimap |\mathcal{L}|(\underline{A}, \underline{B}) & \\
& & \langle - \rangle_{|\mathcal{L}|}^{-1} = & \\
& & & \langle - \rangle_{|\mathcal{L}|}
\end{array}$$

we deduce that $\langle - \rangle_{|\mathcal{L}|}$ is an isomorphism. From

$$\begin{array}{ccc}
\langle - \rangle_{|\mathcal{L}|} \otimes \mathcal{V}(V, |\mathcal{L}|(-, \underline{B})) \otimes \text{id} & \nearrow & ((V \multimap |\mathcal{L}|(\underline{A}, \underline{B})) \multimap |\mathcal{L}|(\underline{A}, \prod_V \underline{B})) \otimes \\
& & ((V \multimap |\mathcal{L}|(\underline{A}', \underline{B})) \multimap (V \multimap |\mathcal{L}|(\underline{A}, \underline{B}))) \otimes \\
& & (V \multimap |\mathcal{L}|(\underline{A}, \underline{B})) \\
\cong \otimes \text{id} \nearrow & & \circlearrowleft_{|\mathcal{L}|} \otimes \text{id} \\
|\mathcal{L}|(\underline{A}, \underline{A}') \otimes & & \\
(V \multimap |\mathcal{L}|(\underline{A}, \underline{B})) & = & |\mathcal{L}|(\underline{A}, \prod_V \underline{B}) \xleftarrow{\text{eval}} ((V \multimap |\mathcal{L}|(\underline{A}, \underline{B})) \multimap \\
& & |\mathcal{L}|(\underline{A}, \prod_V \underline{B})) \otimes \\
& & (V \multimap |\mathcal{L}|(\underline{A}, \underline{B})) \\
\cong \otimes \text{id} \searrow & & \circlearrowleft_{|\mathcal{L}|} \otimes \text{id} \\
|\mathcal{L}|(\underline{A}, \underline{A}') \otimes \mathbb{1} \otimes & & \\
(V \multimap |\mathcal{L}|(\underline{A}, \underline{B})) & & \\
|\mathcal{L}|(-, \prod_V \underline{B}) \otimes \langle - \rangle_{|\mathcal{L}|} & \nearrow & (|\mathcal{L}|(\underline{A}', \prod_V \underline{B}) \multimap |\mathcal{L}|(\underline{A}, \prod_V \underline{B})) \otimes \\
& & ((V \multimap |\mathcal{L}|(\underline{A}', \underline{B})) \multimap |\mathcal{L}|(\underline{A}', \prod_V \underline{B})) \otimes \\
& & (V \multimap |\mathcal{L}|(\underline{A}, \underline{B}))
\end{array}$$

we deduce, by appeal to universality, that $\langle - \rangle_{|\mathcal{L}|}$ is \mathcal{V} -natural, hence $|\mathcal{L}|$ has the power $\prod_V \underline{B}$. From

$$\begin{array}{ccc}
& |\mathcal{L}|(\prod_V \underline{B}, \underline{B}) & \\
\pi_-^{|\mathcal{L}|} \nearrow & & \downarrow F^i \\
V & = & \\
\pi_-^{Di} \searrow & & |\text{Di}|(\prod_V \underline{B}, \underline{B})
\end{array}$$

we deduce that F^i preserves this power.

We now invoke Lemma 6.8(3) for $H := \mathcal{L}[-]$, and deduce that:

6.3 Algebraic operations

We define algebraic operations for Lawvere theories:

Definition 6.18. Let \mathcal{L} be a λ -Lawvere \mathcal{V} -theory, and A, P be λ -presentable objects. An operation op of type $A \langle P \rangle$ in \mathcal{L} is a morphism $\text{op} : A \rightarrow P$ in $|\mathcal{L}|$, i.e., a \mathcal{V} -morphism $\text{op} : \mathbb{1} \rightarrow |\mathcal{L}|(A, P)$.

Let op, op' be algebraic operations of type $A \langle P \rangle$ in $\mathcal{L}, \widehat{\mathcal{L}}$, respectively, and $\mathfrak{T} : \mathcal{L} \rightarrow \widehat{\mathcal{L}}$ a morphism of Lawvere theories. We say that \mathfrak{T} maps op to op' if $\mathfrak{T}(\text{op}) = \text{op}'$, i.e., when

$$\begin{array}{ccc}
 & & |\mathcal{L}|(A, P) \\
 & \text{op} \nearrow & \downarrow \mathfrak{T} \\
 I & & = \\
 & \text{op}' \searrow & \downarrow \\
 & & |\widehat{\mathcal{L}}|(A, P)
 \end{array}$$

Just as in Section 3.1, we write $\text{op} : A \langle P \rangle$ when op is an operation of type $A \langle P \rangle$.

In fact, Plotkin and Power's notion of algebraic operations (see Definition 2.1) arose from the notion of algebraic operations in a Lawvere theory. Thus, the correspondence between algebraic operations and generic effects extends to operations in a Lawvere theory. It is precisely the well-known bijection between arrows in a Lawvere theory and Kleisli arrows:

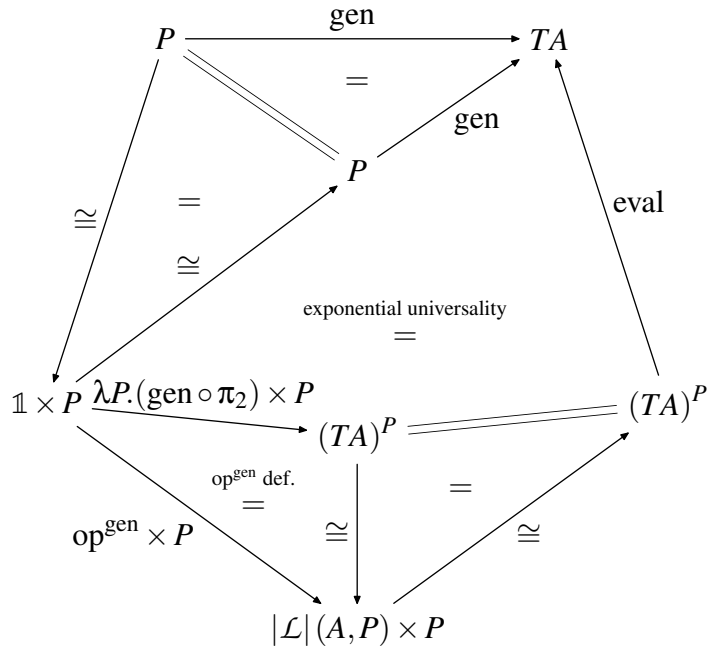
Theorem 6.19. Let \mathcal{V} be a λ -Power category with a cartesian closed structure. The equivalence $\mathbf{Law}_\lambda \mathcal{V} \simeq \mathcal{V}\text{-Monads}_\lambda$ given in Theorem 6.15 induces a bijection between generic effects $\text{gen} : A \langle P \rangle$ for λ -ranked \mathcal{V} -enriched monads and algebraic operations $\text{op} : A \langle P \rangle$ for λ -Lawvere \mathcal{V} -theories, given by:

$$\begin{aligned}
 \text{op}^{\text{gen}} : \mathbb{1} &\xrightarrow{\lambda P. \left(\mathbb{1} \times P \xrightarrow{\pi_2} P \xrightarrow{\text{gen}} TA \right)} (TA)^P = |\mathcal{L}_T|(A, P) \cong |\mathcal{L}|(A, P) \\
 \text{gen}_{\text{op}} : P &\cong \mathbb{1} \times P \xrightarrow{\text{op} \times P} |\mathcal{L}|(A, P) \times P \cong (TA)^P \times P \xrightarrow{\text{eval}} TA
 \end{aligned}$$

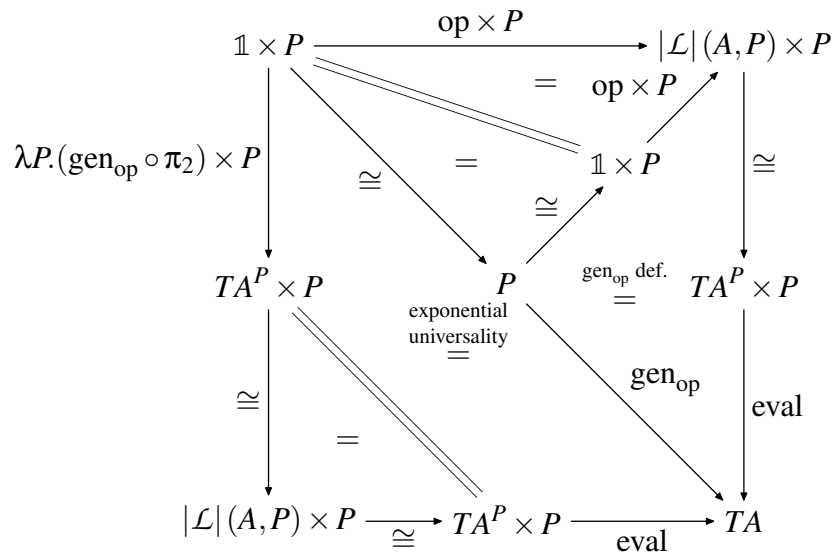
Moreover, this bijection respects the mapping relation between operations: a strong monad morphism m maps gen to gen' if and only if \mathcal{L}_m maps op^{gen} to op'^{gen} .

Proof

The bijection amounts to internalising the Kleisli arrow gen and using the isomorphisms $|\mathcal{L}|(A, P) \cong (TA)^P$. For establishing that $\text{gen}_{\text{op}^{\text{gen}}} = \text{gen}$, we have



Conversely, for $\text{op}^{\text{gen}_{\text{op}}} = \text{op}$ we have, by universality,



Similar calculations show that the commutativity of each of the following two diagrams implies the commutativity of the other:



Therefore, m maps gen to gen' if and only if \mathcal{L}_m maps op^{gen} to $\text{op}^{\text{gen}'}$. ■

Example 6-12. Take \mathcal{V} to be **Set** and λ to be \aleph_0 . Let \mathbb{V} be any finite set with at least two elements denoting storable values. Take $\mathbb{C}\text{hor}$ to be \mathbb{V} . Consider the global state monad $T_{\text{GS}(\mathbb{V})}$, and the monad $T_{\text{I/O}(\mathbb{V})}$ for input/output of ‘characters’ in \mathbb{V} . We define inductively a monad morphism $f : T_{\text{I/O}(\mathbb{V})} \rightarrow T_{\text{GS}(\mathbb{V})}$:

$$\begin{aligned} f_X : T_{\text{I/O}(\mathbb{V})}X &\rightarrow T_{\text{GS}(\mathbb{V})}X \\ f_X : \langle I, \langle t_v \rangle \rangle &\mapsto \lambda v. f_{\{\text{lookup}\}}(t_v)(v) \\ x &\mapsto \lambda v. \langle v, x \rangle \\ \langle O, v_0, t \rangle &\mapsto \lambda v. f(t)(v_0) \end{aligned}$$

Straightforward calculations show f is indeed a monad morphism, and that it maps the input operation to the look-up operation and the output operation to the update operation. Note also that f is surjective, as

$$\lambda v. \langle u_v, x_v \rangle = f(\langle I, \langle \langle O, u_v, x_v \rangle \rangle_{v \in \mathbb{V}} \rangle)$$

Thus, by Theorem 6.19 we have input and output operations for the finitary Lawvere theory $\mathcal{L}_{\text{I/O}(\mathbb{C}\text{hor})}$ corresponding to $T_{\text{I/O}(\mathbb{C}\text{hor})}$, lookup and update operations for the finitary Lawvere theory $\mathcal{L}_{\text{GS}(\mathbb{V})}$, and the monad morphism f maps the input and output operations to the look-up and update operations, respectively. \square

6.4 Algebraic CBPV models

We define a subclass of CBPV models arising from a Lawvere theory.

Definition 6.20. Let λ be a regular cardinal, and \mathcal{V} a locally λ -presentable category, and X a set. A λ -presentable type assignment for X in \mathcal{V} is a type assignment for X in $\mathbf{Pres}_\lambda \mathcal{V}$.

Note that every locally λ -presentable type assignment in \mathcal{V} induces an ordinary type assignment in \mathcal{V} , as $\mathbf{Pres}_\lambda \mathcal{V}$ is a subcategory of \mathcal{V} .

Example 6-13. If \mathbb{V} is a finite set denoting storable values, the type assignment from Example 2-9:

$$\text{lookup} : \mathbb{V} \quad \text{update} : \mathbb{1} \langle \mathbb{V} \rangle$$

is a finitely-presentable type assignment for $\{\text{lookup}, \text{update}\}$ in **Set**. \square

We can now define algebraic models of CBPV:

Definition 2.13⁺. Let Π be a set. An algebraic CBPV Π -model is a quintuple

$$\langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}, \mathcal{L} \llbracket - \rrbracket \rangle$$

where:

- λ is a regular cardinal;
- \mathcal{V} is a λ -Power category with respect to the cartesian closed structure, called the value category;
- type is a λ -presentable type assignment for Π in \mathcal{V} ;
- \mathcal{L} is a λ -Lawvere \mathcal{V} -theory.
- $\mathcal{L} \llbracket - \rrbracket$ assigns to every $\text{op} : A \langle P \rangle$ in Π an algebraic operation $\mathcal{L} \llbracket \text{op} \rrbracket : A \langle P \rangle$ for \mathcal{L} .

To aid the presentation, we use the plus suffix to reflect an algebraic reformulation of a previously visited concept. Thus, Definition 2.13⁺ is an algebraic reformulation of Definition 2.13.

Example 6-14. The global state CBPV model in Example 2-10 can be viewed as an algebraic CBPV model. Take λ to be \aleph_0 , \mathcal{V} to be **Set**, type to be the finitely-presentable type assignment for global state from Example 6-13, \mathcal{L} to be the global state theory $\mathcal{L}_{\text{GS}(\mathbb{V})}$, and $\mathcal{L} \llbracket - \rrbracket$ as the interpretation of look-up and update, as in Example 6-12. \square

More generally, let $\langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}, \mathcal{L} \llbracket - \rrbracket \rangle$ be an algebraic Π -model. Note that we can apply the categorical equivalence $\mathbf{Law}_\lambda \mathcal{V} \simeq \mathcal{V}\text{-Monads}_\lambda$ to \mathcal{L} and obtain a λ -ranked strong monad over \mathcal{V} . In light of Theorem 6.19, we have a Π -model

$$\langle \mathcal{V}, \text{type}, T_\mathcal{L}, \text{op}^{\mathcal{L} \llbracket - \rrbracket} \rangle$$

Conversely, given a Π -model

$$\langle \mathcal{V}, \text{type}, T, O \llbracket - \rrbracket \rangle$$

such that there is a regular cardinal λ which makes \mathcal{V} a λ -Power category with respect to the cartesian structure, type a λ -presentable type assignment, and T be λ -ranked, we obtain an algebraic Σ -model. These constructions are not inverse to each other. However, by composing the two constructions we get a model in which the monad/theory is isomorphic to the monad/theory of the starting model, and this isomorphism preserves

the operations. The proper setting for this fact requires us to introduce morphisms of Π -models. Then, this fact can be phrased as a categorical equivalence between categories of models. However, we have no other interest in morphisms of models and in categorical constructions on models apart from the equivalence we just described here. Therefore, we do not study morphisms of models in this thesis.

To summarise, we presented enriched Lawvere theories and their properties, and specialised our CBPV models to models arising out of some Lawvere theory and its operations.

Chapter 7

Algebraic models

Now the teacher would say to learn your algebra

—Johnny Cash



The semantic structures we considered so far, namely for global state, exceptions, I/O, and recursion, arise from algebraic descriptions using Plotkin and Power’s algebraic theory of effects. We now restrict our attention to such algebraic models. We present a construction that builds an entire hierarchical model from a single CBPV model at the top of the hierarchy. Thus, a refined semantics that takes type-and-effect information into account can be *derived* from an unrefined semantics. Our construction uses *factorisation systems* of categories.

First, in Section 7.1, we present our notion of algebraic models for effect analysis. Next, in Section 7.2, we recall some notions about factorisation systems, and apply them to enriched Lawvere theories. With this machinery at hand, in Section 7.3, we present the categorical conservative restriction construction.

7.1 Categorical algebraic models

Before we present our models, we introduce the following auxiliary definition:

Definition 7.1. *Let Σ be hierarchy (see Definition 3.1). A λ -presentable Σ -type assignment in \mathcal{V} is a λ -presentable type assignment for Π_Σ in \mathcal{V} .*

Without further ado, we define algebraic Σ -models:

Definition 3.2⁺. *Let Σ be a hierarchy. An algebraic Σ -model is a quintuple*

$$\langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}_-, \mathcal{L}_- \llbracket - \rrbracket \rangle$$

where:

- λ is a regular cardinal;
- \mathcal{V} is a λ -Power category with respect to a cartesian closed structure, called the value category;
- type is a λ -presentable Σ -type assignment in \mathcal{V} ;
- \mathcal{L}_- is a functor $\mathcal{E} \rightarrow \mathbf{Law}_\lambda \mathcal{V}$ called the theory hierarchy;
- $\mathcal{L}_- \llbracket - \rrbracket$ assigns to each $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$, $\text{op} : A \langle P \rangle$ an algebraic operation $\mathcal{L}_\varepsilon \llbracket \text{op} \rrbracket$ of type $A \langle P \rangle$ for \mathcal{L}_ε ;

and, for all $\varepsilon \subseteq \varepsilon'$, and $\text{op} \in \varepsilon$, $\mathcal{L}_{\varepsilon \subseteq \varepsilon'}$ maps $\mathcal{L}_\varepsilon \llbracket \text{op} \rrbracket$ to $\mathcal{L}_{\varepsilon'} \llbracket \text{op} \rrbracket$. Thus, $\mathcal{L}_{\varepsilon \subseteq \varepsilon'}$ preserves the operations.

Example 7-1. Let Σ be the hierarchy for global state, and \mathbb{V} a finite set denoting storable values. The global state Σ -model can be viewed as an algebraic Σ -model. We take λ to be \aleph_0 , and \mathcal{V} to be **Set**. For every $\varepsilon \subseteq \{\text{lookup}, \text{update}\}$, we take \mathcal{L}_ε to be the (finitary) Lawvere theory corresponding to the monad $\mathbb{P}\varepsilon$ from Example 3-2, for every $\varepsilon \subseteq \varepsilon'$, $\mathcal{L}_{\varepsilon \subseteq \varepsilon'}$ corresponds to the monad morphism $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ from said example, and for every $\text{op} \in \varepsilon$, $\mathcal{L}_\varepsilon \llbracket \text{op} \rrbracket$ is the algebraic operation corresponding to $O_\varepsilon \llbracket \text{op} \rrbracket$ from that model. By Theorem 6.19, these data constitute an algebraic Σ -model. \square

More generally, as in Section 6.4, every algebraic Σ -model induces a categorical Σ -model where the value category is a λ -Power category with respect to the cartesian structure, the type assignment is λ -presentable, and all monads are λ -ranked, and vice versa. As before, these constructions are only inverse to each other up to isomorphism, hence best studied as equivalences, via a suitable notion of morphisms of Σ -models. As before, we leave such an investigation to further work.

Example 7-2 (benchmark model). Let Σ be an effect hierarchy, and

$$\mathcal{M} = \langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}, \mathcal{L} \llbracket - \rrbracket \rangle$$

an algebraic CBPV Π -model. The *benchmark model* \mathcal{M}_b is given by

$$\mathcal{M}_b := \langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}_-, \mathcal{L}_- \llbracket - \rrbracket \rangle$$

where:

- for all ε , $\mathcal{L}_\varepsilon := \mathcal{L}$;
- for all $\varepsilon \subseteq \varepsilon'$, $\mathcal{L}_{\varepsilon \subseteq \varepsilon'} := \text{id}$; and
- for all ε and $\text{op} \in \varepsilon$, $\mathcal{L}_\varepsilon[\text{op}] := \mathcal{L}[\text{op}]$.

Trivial calculations show that \mathcal{M}_b is indeed an algebraic Σ -model. \square

7.2 Factorisation systems

We recall the notion of a *factorisation system* in a category \mathcal{C} . These are two classes of \mathcal{C} -morphisms, \mathcal{E} and \mathcal{M} conceptually thought of as classes of epi and monic morphisms, respectively, although formally they need not be neither epi nor monic. Given such a pair of classes, we will denote the arrows in \mathcal{E} by $e : A \twoheadrightarrow B$, and the arrows in \mathcal{M} by $m : A \rightarrow B$. The precise definition is as follows:

Definition 7.2. *Let \mathcal{C} be a category. A factorisation system $\langle \mathcal{E}, \mathcal{M} \rangle$ in \mathcal{C} consists of two classes \mathcal{E} and \mathcal{M} of morphisms such that:*

- both \mathcal{E} and \mathcal{M} contain all identity arrows, and are closed under composition with isomorphisms on both sides;
- for every arrow $f : A \rightarrow B$ in \mathcal{C} there exists an \mathcal{E} -morphism $e : A \twoheadrightarrow L$ and an \mathcal{M} -morphism $m : L \rightarrow B$ such that

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ & \searrow e & \nearrow m \\ & L & \end{array}$$

=

and

- for every \mathcal{E} -morphism $e : A \twoheadrightarrow L$, \mathcal{M} -morphism $m : M \rightarrow B$, and every \mathcal{C} arrows $f : A \rightarrow M$, $g : L \rightarrow B$, such that

$$\begin{array}{ccc} A & \xrightarrow{e} & L \\ f \downarrow & = & \downarrow g \\ M & \xrightarrow{m} & B \end{array}$$

there exists a unique C -morphism $h : L \rightarrow M$ filling in the diagonal:

$$\begin{array}{ccc}
 A & \xrightarrow{e} & L \\
 f \downarrow & \searrow \scriptstyle = h & \downarrow g \\
 M & \xrightarrow{m} & B
 \end{array}$$

Sets and functions have the following well-known factorisation system:

Example 7-3. In **Set**, taking \mathcal{E} to be the surjective functions (i.e., the epis) and \mathcal{M} to be the injective functions (i.e., the monos), yields a factorisation system.

Every function $f : X \rightarrow Y$ can be factored as a surjection onto the image $f[X]$:

$$\begin{array}{ccc}
 X & \xrightarrow{f} & Y \\
 f \searrow & \scriptstyle = & \nearrow \subseteq \\
 & f[X] &
 \end{array}$$

□

Example 7-4 (see Meseguer¹ [Mes81]). In $\omega\mathbf{CPO}$, let \mathcal{E} be the class of *dense* functions, these are the Scott-continuous functions $e : V \twoheadrightarrow W$ such that $\text{Cl}(e[V]) = W$, and let \mathcal{M} be the class of *full monos*, these are the Scott-continuous functions $m : V \hookrightarrow W$ such that if $f(v) \leq f(v')$ in W , then $v \leq v'$ in V . Meseguer [Mes81] showed that $\langle \mathcal{E}, \mathcal{M} \rangle$ is a factorisation system in $\omega\mathbf{CPO}$. Indeed, both classes contain all identities and are closed under precomposition with Scott-continuous bijections. Every Scott-continuous function can be factored as the inclusion of the closure of its image:

$$\begin{array}{ccc}
 V & \xrightarrow{f} & W \\
 f \searrow & \scriptstyle = & \nearrow \subseteq \\
 & \text{Cl}(f[V]) &
 \end{array}$$

Finally, consider a commuting square:

$$\begin{array}{ccc}
 V & \xrightarrow{e} & L \\
 f \downarrow & \scriptstyle = & \downarrow g \\
 M & \xrightarrow{m} & W
 \end{array}$$

¹Meseguer's '78 manuscript is rare, but Meseguer describes the results in a later paper [Mes80].

Define U to be the following subset of L :

$$U := \{x \in L \mid \text{there is some } y \in M \text{ such that } m(y) = g(x)\}$$

Consider an arbitrary ω -chain $\langle x_n \rangle$ in U . Then, by U 's definition, we have a (unique) sequence $\langle y_n \rangle$ in M satisfying, for all n , $m(y_n) = g(x_n)$. Because f is monotone and m is full, this sequence is an ω -chain in M . Thus we calculate:

$$m(\bigvee y_n) = \bigvee m(y_n) = \bigvee g(x_n) = g(\bigvee x_n)$$

Therefore, $\bigvee x_n \in U$, hence U is a subdomain of L . We have $e[V] \subseteq U$, as every $v \in V$ satisfies $m(f(v)) = g(e(v))$. Thus, by the closure's definition:

$$L = \text{Cl}(e[V]) \subseteq U \subseteq L$$

hence $L = U$.

Define a function $h : L \rightarrow M$, taking each $x \in L$ to a $y \in M$ satisfying $m(y) = g(x)$. Because m is injective, there is a unique such x . Thus, by definition, we have that h is the unique fill-in morphism:

$$\begin{array}{ccc} V & \xrightarrow{e} & L \\ f \downarrow & \dashrightarrow^{h} & \downarrow g \\ M & \xrightarrow{m} & W \end{array}$$

Thus we have a dense epi-full mono factorisation of $\omega\mathbf{CPO}$. □

We recall a few concepts and results regarding factorisation systems.

Definition 7.3. Let $e : A \rightarrow L$, $m : M \rightarrow B$ be two morphisms. We say that e is left orthogonal to m , and write $e \perp m$, if, for every two morphisms $f : A \rightarrow M$ and $g : L \rightarrow B$ satisfying

$$\begin{array}{ccc} A & \xrightarrow{e} & L \\ f \downarrow & = & \downarrow g \\ M & \xrightarrow{m} & B \end{array}$$

there exists a unique morphism $h : L \rightarrow M$ filling-in the diagonal:

$$\begin{array}{ccc}
 A & \xrightarrow{e} & L \\
 \downarrow f & \dashrightarrow h & \downarrow g \\
 M & \xrightarrow{m} & B
 \end{array}$$

Given any class of morphisms \mathcal{M} , its left orthogonality class ${}^{\perp}\mathcal{M}$ is the class of all morphisms e satisfying $e \perp m$, for every $m \in \mathcal{M}$.

Thus, in a factorisation system $\langle \mathcal{E}, \mathcal{M} \rangle$, $\mathcal{E} \subseteq {}^{\perp}\mathcal{M}$. In fact, the converse inclusion also holds, and \mathcal{E} is completely determined by \mathcal{M} , provided that $\langle {}^{\perp}\mathcal{M}, \mathcal{M} \rangle$ is in fact a factorisation system (see, for example, Adámek et al. [AHS90, Proposition 14.6(3)]).

Factorisation systems can be constructed using an appropriate solution set condition:

Theorem 7.4 (Bousfield's factorisation theorem² [Bou77, Theorem 3.1]). *Let \mathcal{C} be a complete category, and \mathcal{M} a class of maps in \mathcal{C} . Then $\langle {}^{\perp}\mathcal{M}, \mathcal{M} \rangle$ is a factorization system in \mathcal{C} if and only if \mathcal{M} satisfies the following five closure conditions:*

1. every isomorphism is in \mathcal{M} ;
2. \mathcal{M} is closed under composition;
3. if $m \circ f \in \mathcal{M}$ and $m \in \mathcal{M}$, then $f \in \mathcal{M}$;
4. \mathcal{M} is stable under pullbacks: for every pullback square

$$\begin{array}{ccc}
 P & \xrightarrow{f^*(m)} & B \\
 \downarrow & \lrcorner & \downarrow f \\
 A & \xrightarrow{m} & C
 \end{array}$$

with $m \in \mathcal{M}$, we also have $f^*(m) \in \mathcal{M}$; and

5. \mathcal{M} is closed under limits: for every small diagram $D : J \rightarrow \mathcal{C}^{\mathbb{S}}$ in the arrow category $\mathcal{C}^{\mathbb{S}} := \mathcal{C} \downarrow \mathcal{C}$, for which, for all $j \in J$, $Dj \in \mathcal{M}$, the limit arrow is also in \mathcal{M} ,

and the following solution set condition:

²Bousfield states the dual theorem, dealing with right orthogonality classes.

- for every map $f : A \rightarrow B$ in C there is a set S_f of factorisations $f : A \xrightarrow{g} L \xrightarrow{m} B$, with $m \in \mathcal{M}$ such that for every factorisation $f : A \xrightarrow{g} L \xrightarrow{m} B$ there exists some factorisation $f : A \xrightarrow{\hat{g}} \hat{L} \xrightarrow{\hat{m}} B$ in S_f and an arrow $h : \hat{L} \rightarrow L$ such that

$$\begin{array}{ccccc}
 & & \hat{L} & & \\
 & \hat{g} & \nearrow & \hat{m} & \\
 A & & & & B \\
 & g & \searrow & m & \\
 & & L & &
 \end{array}
 \quad
 \begin{array}{c}
 \\
 = \\
 h \\
 = \\
 \\
 \end{array}$$

A category may have multiple factorisation systems. We chose our two factorisation systems of interest, i.e., injections for **Set** and full monos for $\omega\mathbf{CPO}$, seemingly arbitrarily. We would very much like to axiomatise categorically the required properties of our factorisations to make this choice canonical. Category theory has several well-studied notions of monomorphisms, with the following inclusion properties (see, for example, Adámek, Herrlich and Strecker [AHS90, Remark 7.76(2)]):

$$\begin{array}{c}
 \text{section} \\
 \cap \\
 \text{regular monomorphism} \\
 \cap \\
 \text{strict monomorphism} \\
 \cap \\
 \text{strong monomorphism} \\
 \cap \\
 \text{extremal monomorphism} \\
 \cap \\
 \text{monomorphism}
 \end{array}$$

In particular, restricting our attention to *extremal* monos seems appealing, as every locally presentable category has an $\langle \text{epi}, \text{extremal mono} \rangle$ factorisation [AR94, Proposition 1.61]. Unfortunately, the full monomorphisms in $\omega\mathbf{CPO}$ do not coincide with neither the extremal monos nor with the monomorphisms. Rather, they lie between the latter two notions in the above hierarchy. Indeed, recall that an *extremal monomorphism* is a monomorphism $m : A \rightarrow B$ such that the only epimorphisms it can be factored

through are isomorphisms. More explicitly: every epi e for which $m : A \xrightarrow{e} L \xrightarrow{g} B$ is an isomorphism.³ Lehmann [Leh80, Theorem 3] showed that in $\omega\mathbf{CPO}$:

$$\text{extremal monomorphism} \subseteq \text{full monomorphism} \subseteq \text{monomorphism}$$

These inclusions are in fact proper. Lehmann and Pasztor [LP82] exhibit a proper full mono inclusion $m : V \hookrightarrow W$ that is also an epimorphism of $\omega\mathbf{CPO}$. Therefore, this full mono is not an extremal monomorphism. The inclusion of the discrete domain with two elements in the Sierpinski space $2 \subseteq \mathbb{S}$ is a mono that is not full. Therefore,

$$\text{extremal mono} \subset \text{full mono} \subset \text{mono}$$

Lehmann and Pasztor [LP82] give a categorical characterisation of the full monos as extremal monos of a larger category. However, we cannot justify switching from $\omega\mathbf{CPO}$ to this category for no other reason. Therefore, we do not commit to a particular choice of factorisation system. The following folklore result⁴ is a central construction in our account.

Theorem 7.5. *Let \mathcal{V} be a λ -Power category, and $\langle \perp\mathcal{M}, \mathcal{M} \rangle$ a factorisation system in \mathcal{V} . If \mathcal{M} is a subclass of the monomorphisms in \mathcal{V} , then $\mathbf{Law}_\lambda \mathcal{V}$ has a factorisation system $\langle \perp(\mathcal{M}^{\text{law}}), \mathcal{M}^{\text{law}} \rangle$, where \mathcal{M}^{law} consists of all the Lawvere theory morphisms \mathfrak{M} for which, for every λ -presentable A , the component $\mathfrak{M}_{A, \mathbb{1}}$ is in \mathcal{M} .*

Proof

We invoke Bousfield's factorisation theorem (Theorem 7.4). The closure conditions follow from their counterparts in each component:

1. If \mathfrak{T} is a Lawvere theory isomorphism, then its components $\mathfrak{T}_{A, \mathbb{1}}$ are isomorphisms. Thus, \mathfrak{T} 's $A, \mathbb{1}$ components lie in \mathcal{M} , hence \mathfrak{T} is in \mathcal{M}^{law} .
2. Similarly, as composition of Lawvere theory morphisms is defined component-wise, we deduce that \mathcal{M}^{law} is closed under composition.
3. If $\mathfrak{M} \circ \mathfrak{T} \in \mathcal{M}^{\text{law}}$, and $\mathfrak{M} \in \mathcal{M}^{\text{law}}$, then for each $A, \mathbb{1}$ component we have that $\mathfrak{M}_{A, \mathbb{1}} \circ \mathfrak{T}_{A, \mathbb{1}}$ and $\mathfrak{M}_{A, \mathbb{1}}$ are in \mathcal{M} . Thus, $\mathfrak{M}_{A, \mathbb{1}}$ is in \mathcal{M} , and we deduce that \mathfrak{T} is in \mathcal{M}^{law} .
4. Consider a pullback square

³In a locally presentable category, the inclusion strong mono \subseteq extremal mono becomes an equality [AR94, Proposition 1.61].

⁴John Power, private communication, 2012.

$$\begin{array}{ccc}
\widehat{\mathcal{L}} & \xrightarrow{\mathfrak{T}^*(\mathfrak{M})} & \mathcal{L}_2 \\
\downarrow & \lrcorner & \downarrow \mathfrak{T} \\
\mathcal{L}_1 & \xrightarrow{\mathfrak{M}} & \mathcal{L}
\end{array}$$

with \mathfrak{M} in \mathcal{M}^{law} . As limits of Lawvere theories are taken componentwise (Theorem 6.17), we have a pullback square

$$\begin{array}{ccc}
|\widehat{\mathcal{L}}|(A, \mathbb{1}) & \xrightarrow{(\mathfrak{T}^*(\mathfrak{M}))_{A, \mathbb{1}}} & |\mathcal{L}_2|(A, \mathbb{1}) \\
\downarrow & \lrcorner & \downarrow \mathfrak{T}_{A, \mathbb{1}} \\
|\mathcal{L}_1|(A, \mathbb{1}) & \xrightarrow{\mathfrak{M}_{A, \mathbb{1}}} & |\mathcal{L}|(A, \mathbb{1})
\end{array}$$

with $\mathfrak{M}_{A, B}$ in \mathcal{M} . Thus, $\mathfrak{T}^*(\mathfrak{M})$ is in \mathcal{M}^{law} .

5. A similar appeal to Theorem 6.17 shows that \mathcal{M}^{law} is closed under limits.

Some care is required to establish the solution set condition. The crux of the argument lies in the fact that a locally λ -presentable category is locally small (see Corollary 5.13), well-powered (see Proposition 5.14), has a set \mathbb{P} of non-isomorphic representative objects of the λ -presentable objects (see Proposition 5.11), and that the hom-objects are determined up to isomorphism by the hom-objects $|\mathcal{L}|(A, \mathbb{1})$ (due to preservation of powers). A slight complication arises from our choice to define Lawvere theories as consisting of a large class of objects, rather than as a small skeleton \mathbb{P} . We deal with this difficulty by noting that, as there is only a set \mathbb{P} of λ -presentable objects up to isomorphism, the full inclusion $\mathbb{P} \subseteq \mathbf{Pres}_\lambda \mathcal{V}$ forms an equivalence of categories [ML98, Proposition IV.3.2], and has an adjoint $\partial : \mathbf{Pres}_\lambda \mathcal{V} \rightarrow \mathbb{P}$ and a specified isomorphism $\eta^\partial : \text{id} \xrightarrow{\cong} \partial$.

Let $\mathfrak{T} : \mathcal{L}_1 \rightarrow \mathcal{L}_2$ be any morphism of Lawvere theories. Let $I_{\mathfrak{T}}$ be the following set of sextuples

$$\left\langle T_{\mathcal{L}} -, \underline{\text{id}}_{\mathcal{L}}, \circ_{\mathcal{L}}^{\mathcal{L}}, \mathcal{L} \llbracket - \rrbracket_{\langle -, - \rangle}, \mathfrak{T}_{\langle -, - \rangle}^g, \mathfrak{M}_{\langle -, - \rangle} \right\rangle$$

where:

- $T_{\mathcal{L}} -$ assigns to each object A in \mathbb{P} a subobject $T_{\mathcal{L}} A$ from the set $\text{Sub}(|\mathcal{L}_2|(A, \mathbb{1}))$;

Using $T_{\mathcal{L}}$, for each pair of objects A, B in \mathbb{P} , denote by $|\mathcal{L}|(A, B)$ the object $B \multimap T_{\mathcal{L}} A$;

- $\underline{\text{id}}_{\mathcal{L}}$ assigns to each object A in \mathbb{P} a morphism

$$\underline{\text{id}}_A^{\mathcal{L}} : \mathbb{1} \rightarrow |\mathcal{L}|(A, A)$$

- $\circ_{\langle -, -, - \rangle}^{\mathcal{L}}$ assigns to each triple of objects A , B , and C in \mathbb{P} a morphism

$$\circ_{A, B, C}^{|\mathcal{L}|} : |\mathcal{L}|(B, C) \otimes |\mathcal{L}|(A, B) \rightarrow |\mathcal{L}|(A, C)$$

- $\mathcal{L} \llbracket - \rrbracket_{\langle -, - \rangle}$ assigns to each pair of objects A and B in \mathbb{P} a morphism

$$\mathcal{L} \llbracket - \rrbracket_{A, B} : \mathbf{Pres}_{\lambda}^{\text{op}} \mathcal{V}(A, B) \rightarrow |\mathcal{L}|(A, B)$$

- $\mathfrak{T}_{\langle -, - \rangle}^g$ assigns to each pair of objects A and B in \mathbb{P} a morphism

$$\mathfrak{T}_{A, B}^g : |\mathcal{L}_1|(A, B) \rightarrow |\mathcal{L}|(A, B)$$

- $\mathfrak{M}_{\langle -, - \rangle}$ assigns to each pair of objects A and B in \mathbb{P} an \mathcal{M} -morphism

$$\mathfrak{M}_{A, B} : |\mathcal{L}|(A, B) \rightarrow |\mathcal{L}_2|(A, B)$$

Because our assignments only range over the set \mathbb{P} and assign elements from sets, $I_{\mathfrak{T}}$ is indeed a set.

Denote by $S_{\mathfrak{T}}$ the set of all factorisations $\mathfrak{T} : \mathcal{L}_1 \xrightarrow{\mathfrak{T}^g} \mathcal{L} \xrightarrow{\mathfrak{M}} \mathcal{L}_2$ with \mathfrak{M} in \mathcal{M}^{law} , together with a specified family $\langle T_{\mathcal{L}} A \rangle_{A \in \mathbb{P}}$, satisfying, for all λ -presentable objects A , B , C , and V :

- $T_{\mathcal{L}} \partial A \in \text{Sub}(|\mathcal{L}_2|(\partial A, \mathbb{1}))$;
- $|\mathcal{L}|(A, B) = \partial B \multimap T_{\mathcal{L}} \partial A$;
- $\underline{\text{id}}_A^{\mathcal{L}} : \mathbb{1} \rightarrow |\mathcal{L}|(A, A) = \underline{\text{id}}_{\partial A}^{\mathcal{L}} : \mathbb{1} \rightarrow |\mathcal{L}_2|(\partial A, \partial A)$;
- $\circ_{A, B, C}^{|\mathcal{L}|} = \circ_{\partial A, \partial B, \partial C}^{|\mathcal{L}|}$;
- $\mathcal{L} \llbracket - \rrbracket_{A, B} = \mathcal{L} \llbracket - \rrbracket_{\partial A, \partial B}$;
- $\mathfrak{T}_{A, B}^g = \mathfrak{T}_{\partial A, \partial B}^g$; and
- $\mathfrak{M}_{A, B} = \mathfrak{M}_{\partial A, \partial B}$

Thus we have an injection from $S_{\mathfrak{T}}$ into the set $I_{\mathfrak{T}}$, and $S_{\mathfrak{T}}$ is indeed a set.

Let $\mathfrak{T} : \mathcal{L}_1 \xrightarrow{\mathfrak{T}^g} \mathcal{L} \xrightarrow{\mathfrak{M}} \mathcal{L}_2$ be any factorisation of \mathfrak{T} . As $\mathfrak{M}_{\partial A, \mathbb{1}} : |\mathcal{L}|(\partial A, \mathbb{1}) \rightarrow |\mathcal{L}_2|(\partial A, \mathbb{1})$ is an \mathcal{M} -morphism, it is also a monomorphism. Therefore, there exists a unique sub-object $T_{\widehat{\mathcal{L}}}\partial A \in \text{Sub}(|\mathcal{L}_2|(\partial A, \mathbb{1}))$ and a unique isomorphism

$$F_A : T_{\widehat{\mathcal{L}}}\partial A \xrightarrow{\cong} |\mathcal{L}|(\partial A, \mathbb{1})$$

satisfying:

$$\begin{array}{ccc} T_{\widehat{\mathcal{L}}}\partial A & & \\ \downarrow F_A & \searrow \hat{m}_A & \\ |\mathcal{L}|(\partial A, \mathbb{1}) & \xrightarrow[\mathfrak{M}_{\partial A, \mathbb{1}}]{\text{def}} & |\mathcal{L}_2|(\partial A, \mathbb{1}) \end{array}$$

Note that \hat{m}_A , and F_A depend only on ∂A .

For every λ -presentable objects A, B , we have isomorphisms:

$$\eta_A^\partial : A \xrightarrow{\cong} \partial A, \quad \eta_B^\partial : B \xrightarrow{\cong} \partial B, \quad \text{unit}_{\text{right}} : \partial B \otimes \mathbb{1} \xrightarrow{\cong} \partial B$$

We therefore have \mathcal{V} -isomorphisms:

$$\begin{aligned} \mathcal{L} \left[\left(\eta_A^\partial \right)^{-1} \right] & : \mathbb{1} \xrightarrow{(\eta_A^\partial)^{-1}} \mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}(A, \partial A) \xrightarrow{\mathcal{L}[-]} |\mathcal{L}|(A, \partial A) \\ \mathcal{L} \left[\eta_B^\partial \right] & : \mathbb{1} \xrightarrow{\eta_B^\partial} \mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}(\partial B, B) \xrightarrow{\mathcal{L}[-]} |\mathcal{L}|(\partial B, B) \\ \mathcal{L} \left[\cong \right] & : \mathbb{1} \xrightarrow{\cong} \mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}(\partial B, \partial B \otimes \mathbb{1}) \xrightarrow{\mathcal{L}[-]} |\mathcal{L}|(\partial B, \partial B \otimes \mathbb{1}) \\ & = |\mathcal{L}|(\partial B, \prod_{\partial B} \mathbb{1}) \end{aligned}$$

Define, for every A, B , $\widehat{\mathcal{L}}(A, B) := \partial B \multimap T_{\widehat{\mathcal{L}}}\partial A$. We have an isomorphism

$$\begin{aligned} G_{A,B} : \widehat{\mathcal{L}}(A, B) & = \partial B \multimap T_{\widehat{\mathcal{L}}}\partial A \xrightarrow{\text{id} \multimap F_A} \partial B \multimap |\mathcal{L}|(\partial A, \mathbb{1}) \\ & \downarrow \langle - \rangle \\ & \widehat{\mathcal{L}}(\partial A, \prod_{\partial B} \mathbb{1}) \xrightarrow{|\mathcal{L}|(\text{id}, \mathcal{L}[\cong])} |\mathcal{L}|(\partial A, \partial B) \end{aligned}$$

Note that $G_{A,B}$ only depends on $\partial A, \partial B$. Thus, we obtain an isomorphism:

$$\mathfrak{T}_{A,B}^h : \widehat{\mathcal{L}}(A, B) \xrightarrow{G_{A,B}} |\mathcal{L}|(\partial A, \partial B) \xrightarrow{|\mathcal{L}|(\mathcal{L}[(\eta_A^\partial)^{-1}], \mathcal{L}[\eta_B^\partial])} |\mathcal{L}|(A, B)$$

Define:

$$\begin{array}{ccc}
 \mathbb{1} \begin{array}{l} \xrightarrow{\underline{\text{id}}|\widehat{\mathcal{L}}|} \\ \xrightarrow{\underline{\text{id}}|\mathcal{L}|} \end{array} & \begin{array}{c} |\widehat{\mathcal{L}}|(A,A) \\ \xrightarrow{\text{def}} \\ |\mathcal{L}|(A,A) \end{array} & \begin{array}{c} \xrightarrow{\uparrow(\mathfrak{T}^h)^{-1}} \\ \xrightarrow{\uparrow(\mathfrak{T}^h)^{-1}} \end{array} \\
 & & \\
 & \begin{array}{c} |\widehat{\mathcal{L}}|(B,C) \otimes |\widehat{\mathcal{L}}|(A,B) \\ \xrightarrow{\mathfrak{T}^h \otimes \mathfrak{T}^h \downarrow} \\ |\mathcal{L}|(B,C) \otimes |\mathcal{L}|(A,B) \end{array} & \begin{array}{c} \xrightarrow{\underline{\circ}|\widehat{\mathcal{L}}|} \\ \xrightarrow{\underline{\circ}|\mathcal{L}|} \end{array} & \begin{array}{c} |\widehat{\mathcal{L}}|(A,C) \\ \xrightarrow{\text{def}} \\ |\mathcal{L}|(A,C) \end{array} \\
 & & \\
 & & \begin{array}{c} \xrightarrow{\uparrow(\mathfrak{T}^h)^{-1}} \\ \xrightarrow{\uparrow(\mathfrak{T}^h)^{-1}} \end{array}
 \end{array}$$

$$\begin{array}{ccc}
 V \multimap |\widehat{\mathcal{L}}|(A,B) & \xrightarrow{\langle - \rangle|\widehat{\mathcal{L}}|} & |\widehat{\mathcal{L}}|(A, \prod_V B) & V \multimap |\widehat{\mathcal{L}}|(A,B) & \xleftarrow{\langle - \rangle|\widehat{\mathcal{L}}|^{-1}} & |\widehat{\mathcal{L}}|(A, \prod_V B) \\
 \downarrow V \multimap \mathfrak{T}^h & & \uparrow \text{def } (\mathfrak{T}^h)^{-1} & \downarrow V \multimap \mathfrak{T}^h & & \uparrow \text{def } (\mathfrak{T}^h)^{-1} \\
 V \multimap |\mathcal{L}|(A,B) & \xrightarrow{\langle - \rangle|\mathcal{L}|} & |\mathcal{L}|(A, \prod_V B) & V \multimap |\mathcal{L}|(A,B) & \xleftarrow{\langle - \rangle|\mathcal{L}|^{-1}} & |\mathcal{L}|(A, \prod_V B) \\
 & & \uparrow \text{def } (\mathfrak{T}^h)^{-1} & & & \downarrow \mathfrak{T}^h
 \end{array}$$

$$\begin{array}{ccc}
 & \widehat{\mathcal{L}}[-] & \xrightarrow{\quad} & |\widehat{\mathcal{L}}|(A,B) \\
 & \uparrow & \text{def} & \uparrow (\mathfrak{T}^h)^{-1} \\
 |\mathbf{Pres}_\lambda^{\text{op}} \mathcal{V}|(A,B) & & & \\
 & \downarrow & & \downarrow \\
 & \mathcal{L}[-] & \xrightarrow{\quad} & |\mathcal{L}|(A,B)
 \end{array}$$

Just as in the proof of Theorem 6.17, we use Lemma 6.8 to show that these data define a Lawvere theory $\widehat{\mathcal{L}}$ and a Lawvere theory isomorphism $\mathfrak{T}^h : \widehat{\mathcal{L}} \xrightarrow{\cong} \mathcal{L}$. We therefore obtain a factorisation $\mathfrak{T} : \mathcal{L}_1 \xrightarrow{\widehat{\mathfrak{T}}^g} \widehat{\mathcal{L}} \xrightarrow{\widehat{\mathfrak{M}}} \mathcal{L}_2$ by setting

$$\begin{array}{ccccc}
 & & \widehat{\mathcal{L}} & & \\
 & \widehat{\mathfrak{T}}^g \nearrow & \uparrow & \widehat{\mathfrak{M}} \searrow & \\
 \mathcal{L}_1 & & \text{def } (\mathfrak{T}^h)^{-1} & & \mathcal{L}_2 \\
 & \widehat{\mathfrak{T}} \searrow & \downarrow \mathfrak{T}^h & \text{def} \searrow & \\
 & & \mathcal{L} & & \\
 & & \uparrow \widehat{\mathfrak{M}} & & \\
 & & \mathcal{L} & &
 \end{array}$$

As \mathfrak{T}^h is an isomorphism, $\widehat{\mathfrak{M}}$ is in \mathcal{M}^{law} .

It remains to show that this factorisation belongs to the solution set $S_{\widehat{\mathfrak{T}}}$. The specified family is $\langle T_{\widehat{\mathcal{L}}} A \rangle_{A \in \mathbb{P}}$. By fiat, $|\widehat{\mathcal{L}}|(A, B) = \partial B \multimap T_{\widehat{\mathcal{L}}} A$ is independent of our choice of representatives A, B . Straightforward, yet tedious, calculations show that $\underline{\text{id}}_A^{|\widehat{\mathcal{L}}|}$, $\underline{\circ}_{A,B,C}^{|\widehat{\mathcal{L}}|}$, $\widehat{\mathcal{L}}[-]_{A,B}$, $\widehat{\mathfrak{T}}_{A,B}^g$, and $\widehat{\mathfrak{M}}_{A,B}$ are independent of their representatives. For ex-

ample, to show that $\underline{\text{id}}_A^{|\widehat{\mathcal{L}}|}$ only depends on ∂A , note that, by definition:

$$\begin{aligned} \underline{\text{id}}_A^{|\widehat{\mathcal{L}}|} &= G_{A,A}^{-1} \circ |\mathcal{L}|(\mathcal{L}[\eta^\partial], \mathcal{L}[(\eta^\partial)^{-1}]) \circ \underline{\text{id}}_A^{|\mathcal{L}|} \\ &\stackrel{(6.9a)}{\downarrow} = G_{A,A}^{-1} \circ |\mathcal{L}|(\mathcal{L}[\eta^\partial], \mathcal{L}[(\eta^\partial)^{-1}]) \circ \mathcal{L}[\text{id}_A] \stackrel{(6.9b)}{\downarrow} = G_{A,A}^{-1} \circ \mathcal{L}[\eta^\partial \circ \text{id}_A \circ (\eta^\partial)^{-1}] \\ &= G_{A,A}^{-1} \circ \mathcal{L}[\text{id}_{\partial A}] \end{aligned}$$

As $G_{A,A}^{-1}$ is independent of the representative A , we deduce that $\underline{\text{id}}_A^{|\widehat{\mathcal{L}}|}$ is independent of the representative. Similar arguments using Diagrams 6.9c and 6.9d show $\circlearrowleft_{A,B,C}^{|\widehat{\mathcal{L}}|}$ is independent of the choice of representatives, and finally Diagram 6.9e is used thrice (separately) to show that $\widehat{\mathcal{L}}[-]_{A,B}$, $\mathfrak{T}_{A,B}^g$, and $\widehat{\mathfrak{M}}_{A,B}$ are independent of their representatives.

Therefore, the solution set condition is satisfied. From Bousfield's factorisation theorem we deduce that $\langle \perp(\mathcal{M}^{\text{law}}), \mathcal{M}^{\text{law}} \rangle$ is a factorisation system in $\mathbf{Law}_\lambda \mathcal{V}$. ■

This theorem explicitly describes the \mathcal{M} -class of the factorisation system, but leaves the \mathcal{E} -class unspecified. We do not know a general characterisation for this class. The following partial description will suffice for our purposes:

Proposition 7.6. *Let \mathcal{V} be a λ -Power category, and $\langle \mathcal{E}, \mathcal{M} \rangle$ a factorisation system on \mathcal{V} . Denote by \mathcal{M}^{law} the class of all Lawvere theory morphisms \mathfrak{M} whose components $\mathfrak{M}_{A,\emptyset}$ are in \mathcal{M} , as in Theorem 7.5. Denote by $\mathcal{E}_*^{\text{law}}$ the class of all Lawvere theory morphisms \mathfrak{E} whose components $\mathfrak{E}_{A,\emptyset}$ are in \mathcal{E} . If every morphism in \mathcal{M} is a monomorphism in \mathcal{V} , then every morphism in $\mathcal{E}_*^{\text{law}}$ is left-orthogonal to \mathcal{M}^{law} :*

$$\mathcal{E}_*^{\text{law}} \subseteq \perp(\mathcal{M}^{\text{law}})$$

Proof

Let \mathfrak{E} be an $\mathcal{E}_*^{\text{law}}$ -morphism. Consider an arbitrary commuting square:

$$\begin{array}{ccc} \mathcal{L}_1 & \xrightarrow{\mathfrak{E}} & \mathcal{L}_2 \\ \mathfrak{T} \downarrow & & \downarrow \widehat{\mathfrak{T}} \\ \mathcal{L}_3 & \xrightarrow{\mathfrak{M}} & \mathcal{L}_4 \end{array} \quad =$$

For every λ -presentable A , we have a commuting square:

$$\begin{array}{ccc}
 |\mathcal{L}_1|(A, \mathbb{1}) & \xrightarrow{\mathfrak{E}} & |\mathcal{L}_2|(A, \mathbb{1}) \\
 \downarrow \mathfrak{T} & = & \downarrow \hat{\mathfrak{T}} \\
 |\mathcal{L}_3|(A, \mathbb{1}) & \xrightarrow{\mathfrak{M}} & |\mathcal{L}_4|(A, \mathbb{1})
 \end{array}$$

As $\langle \mathcal{E}, \mathcal{M} \rangle$ is a factorisation system, we have a unique fill-in morphism:

$$\begin{array}{ccc}
 |\mathcal{L}_1|(A, \mathbb{1}) & \xrightarrow{\mathfrak{E}} & |\mathcal{L}_2|(A, \mathbb{1}) \\
 \downarrow \mathfrak{T} & \begin{array}{c} = \\ \nearrow F_A \end{array} & \downarrow \hat{\mathfrak{T}} \\
 |\mathcal{L}_3|(A, \mathbb{1}) & \xrightarrow{\mathfrak{M}} & |\mathcal{L}_4|(A, \mathbb{1})
 \end{array}$$

Define, for all λ -presentable A, B :

$$\begin{array}{ccccc}
 |\mathcal{L}_2|(A, B) & \xrightarrow{|\mathcal{L}_2|(\underline{\text{id}}, \mathcal{L}_2[\cong])} & |\mathcal{L}_2|(A, B \otimes \mathbb{1}) & \xrightarrow{\langle - \rangle_2^{-1}} & B \multimap |\mathcal{L}_2|(A, \mathbb{1}) \\
 \downarrow \mathfrak{F} & & \stackrel{\text{def}}{=} & & \downarrow B \multimap F \\
 |\mathcal{L}_3|(A, B) & \xleftarrow{|\mathcal{L}_2|(\underline{\text{id}}, \mathcal{L}_3[\cong])} & |\mathcal{L}_3|(A, B \otimes \mathbb{1}) & \xleftarrow{\langle - \rangle_3} & B \multimap |\mathcal{L}_3|(A, \mathbb{1})
 \end{array}$$

Thus, \mathfrak{F} is a diagonal fill-in:

$$\begin{array}{ccc}
 \mathcal{L}_1 & \xrightarrow{\mathfrak{E}} & \mathcal{L}_2 \\
 \mathfrak{T} \downarrow & \searrow^{\mathfrak{F}} & \downarrow \hat{\mathfrak{T}} \\
 \mathcal{L}_3 & \xrightarrow{\mathfrak{M}} & \mathcal{L}_4
 \end{array}$$

Conversely, let $\hat{\mathfrak{F}}$ be any diagonal fill-in:

$$\begin{array}{ccc}
 \mathcal{L}_1 & \xrightarrow{\mathfrak{E}} & \mathcal{L}_2 \\
 \mathfrak{T} \downarrow & \searrow^{\hat{\mathfrak{F}}} & \downarrow \hat{\mathfrak{T}} \\
 \mathcal{L}_3 & \xrightarrow{\mathfrak{M}} & \mathcal{L}_4
 \end{array}$$

For every λ -presentable A , we have

$$\begin{array}{ccc}
 |\mathcal{L}_1|(A, \mathbb{1}) & \xrightarrow{\mathfrak{E}} & |\mathcal{L}_2|(A, \mathbb{1}) \\
 \mathfrak{T} \downarrow & \searrow^{\hat{\mathfrak{F}}} & \downarrow \hat{\mathfrak{T}} \\
 |\mathcal{L}_3|(A, \mathbb{1}) & \xrightarrow{\mathfrak{M}} & |\mathcal{L}_4|(A, \mathbb{1})
 \end{array}$$

thus, as $\langle \mathfrak{E}, \mathfrak{M} \rangle$ is a factorisation system, necessarily $\hat{\mathfrak{F}}_{A, \mathbb{1}} = F_A$. For every λ -presentable A we also have:

$$\begin{array}{ccccc}
 |\mathcal{L}_2|(A, B) & \xrightarrow{|\mathcal{L}_2|(\underline{\text{id}}, \mathcal{L}_2[\cong])} & |\mathcal{L}_2|(A, B \otimes \mathbb{1}) & \xrightarrow{\langle - \rangle_2^{-1}} & B \multimap |\mathcal{L}_2|(A, \mathbb{1}) \\
 \hat{\mathfrak{F}} \downarrow & \text{Diagram 6.9e} & \hat{\mathfrak{F}} \downarrow & \text{Lemma 6.7} & B \multimap \hat{\mathfrak{F}} \left(= \right) B \multimap F \\
 |\mathcal{L}_3|(A, B) & \xleftarrow{|\mathcal{L}_2|(\underline{\text{id}}, \mathcal{L}_3[\cong])} & |\mathcal{L}_3|(A, B \otimes \mathbb{1}) & \xleftarrow{\langle - \rangle_3} & B \multimap |\mathcal{L}_3|(A, \mathbb{1})
 \end{array}$$

hence $\hat{\mathfrak{F}} = \mathfrak{F}$, and the diagonal fill-in morphism is unique.

Thus, \mathfrak{E} is left-orthogonal to \mathcal{M}^{law} . ■

Next, we deal with factorisations of ranked monads.

Definition 7.7. Let \mathcal{F} be a class of morphisms in a locally λ -presentable category. We say that \mathcal{F} is closed under λ -directed colimits if for every λ -directed diagram $D : I \rightarrow \mathcal{C}^{\mathbb{S}}$ in the arrow category $\mathcal{C}^{\mathbb{S}} := \mathcal{C} \downarrow \mathcal{C}$ for which $Di \in \mathcal{F}$, for all $i \in I$, the colimit arrow is also in \mathcal{F} .

We say that a factorisation system $\langle \mathcal{E}, \mathcal{M} \rangle$ is closed under λ -directed colimits if \mathcal{M} is.

Note that, in light of the dual to Bousfield's factorisation theorem, in every factorisation system $\langle \mathcal{E}, \mathcal{M} \rangle$, the class \mathcal{E} is closed under all colimits, and in particular the λ -directed ones.

Example 7-5. The surjection-injection factorisation system in **Set** is closed under finitely directed colimits. Indeed, take any finitely-directed diagram $D : I \rightarrow \mathbf{Set}^{\mathbb{S}}$.

For every $i \in I$, denote $Di : D^1 i \xrightarrow{m_i} D^2 i$, and for every $i \leq j$ in I denote by $D(i \leq j)$ the pair $\langle D^1(i \leq j), D^2(i \leq j) \rangle$. Because colimits in functor categories are taken componentwise, we have two colimiting cocones $\langle C^1, c^1 \rangle$ and $\langle C^2, c^2 \rangle$ for D^1 and D^2 , respectively, and the colimit of D is given by a function $C^2 \xrightarrow{f} C^1$ such that $\langle c^1, c^2 \rangle$ is the colimiting cocone for D .

We need to show that f is injective. Take any x, y in C^1 such that $f(x) = f(y)$. By Lemma 5.3(1), there exist some $i_x, i_y, d_x \in D^1 i_x$, and $d_y \in D^1 i_y$ such that $c_{i_x}^1(d_x) = x$ and $c_{i_y}^1(d_y) = y$. We thus have $m_{i_x}(d_x) \in D^2 i_x$ and $m_{i_y}(d_y) \in D^2 i_y$ satisfying

$$c_{i_x}^2(m_{i_x}(d_x)) = f(c_{i_x}^1(d_x)) = f(x) = f(y) = c_{i_y}^2(m_{i_y}(d_y)) \quad (*)$$

From Lemma 5.3(2), there exists some $j \geq i_x, i_y$ such that:

$$D^2(j \geq i_x)(m_{i_x}(d_x)) = D^2(j \geq i_y)(m_{i_y}(d_y)) \quad (*)$$

We thus have:

$$\begin{aligned} m_j(D^1(j \geq i_x)(d_x)) &= D^2(j \geq i_x)(m_{i_x}(d_x)) \\ &= D^2(j \geq i_y)(m_{i_y}(d_y)) = m_j(D^1(j \geq i_y)(d_y)) \end{aligned} \quad (*)$$

Because f_j is injective, we deduce that

$$D^1(j \geq i_x)(d_x) = D^1(j \geq i_y)(d_y) \quad (*)$$

By applying c_j^1 , we deduce that:

$$x = c_{i_x}^1(d_x) = c_j^1(D^1(j \geq i_x)(d_x)) = c_j^1(D^1(j \geq i_y)(d_y)) = y \quad (*)$$

Thus f is injective. Therefore, the class of injective functions is closed under finitely directed colimits. \square

Example 7-6. The dense-full mono factorisation of $\omega\mathbf{CPO}$ is closed under countably directed colimits. The proof is identical to the previous example, replacing the assumption $f(x) = f(y)$ by $f(x) \leq f(y)$, the use of Lemma 5.3 by that of Lemma 5.4, and modifying the equalities marked with $(*)$ with appropriate inequalities. \square

Proposition 7.8. *Let \mathcal{V} be a λ -Power category, \mathcal{F} be a class of morphisms in \mathcal{V} closed under λ -directed colimits, and $\mathfrak{T} : \mathcal{L} \rightarrow \mathcal{L}'$ a morphism of Lawvere theories. If, for every λ -presentable object A , the component $\mathfrak{T}_{A, \mathbb{1}} : |\mathcal{L}|(A, \mathbb{1}) \rightarrow |\mathcal{L}'|(A, \mathbb{1})$ is an \mathcal{F} -morphism, then all components of the corresponding monad morphism $T_{\mathfrak{T}} : T_{\mathcal{L}} \rightarrow T_{\mathcal{L}'}$ are in \mathcal{F} .*

Proof

In Section 6.2 (see Equation (6.10)), we showed that, for any A , the A component of $T_{\mathfrak{T}}$ is the colimit of a λ -directed diagram $D_{\mathfrak{T}} : I \rightarrow \mathcal{V}^{\mathbb{S}}$ whose object map is given by:

$$D_{\mathfrak{T}}i : |\mathcal{L}|(Di, \mathbb{1}) \xrightarrow{\mathfrak{T}} |\mathcal{L}'|(Di, \mathbb{1}) \quad (6.10)$$

Thus, $D_{\mathfrak{T}}$ is a λ -directed diagram whose objects are \mathcal{F} -morphisms. As \mathcal{F} is closed under λ -directed colimits, we deduce that the A component of $T_{\mathfrak{T}}$ is in \mathcal{F} . \blacksquare

We transport our factorisation to ranked monads:

Theorem 7.9. *Let \mathcal{V} be a λ -Power category, and $\langle \mathcal{E}, \mathcal{M} \rangle$ a factorisation system in \mathcal{V} . If \mathcal{M} is a subclass of the monomorphisms closed under λ -directed colimits, then the category of λ -ranked monads $\mathcal{V}\text{-Monads}_{\lambda}$ has a factorisation system $\langle \mathcal{E}^{\text{mon}}, \mathcal{M}^{\text{mon}} \rangle$ where:*

- \mathcal{M}^{mon} consists of all monad morphisms between λ -ranked monads whose components are \mathcal{M} -morphisms; and
- \mathcal{E}^{mon} includes all monad morphisms between λ -ranked monads whose components are \mathcal{E} -morphisms, but may a-priori include other morphisms.

Proof

Transport the factorisation system $\langle \mathcal{E}^{\text{law}}, \mathcal{M}^{\text{law}} \rangle$ in $\mathbf{Law}_{\lambda}\mathcal{V}$ from Theorem 7.5 to a factorisation system $\langle \mathcal{E}^{\text{mon}}, \mathcal{M}^{\text{mon}} \rangle$ in $\mathcal{V}\text{-Monads}_{\lambda}$ using the equivalence from Theorem 6.15. Thus, \mathcal{E}^{mon} consists of all monad morphisms e such that $\mathcal{L}_e \in \mathcal{E}^{\text{law}}$, and \mathcal{M}^{mon} consists of all monad morphisms m such that $\mathcal{L}_m \in \mathcal{M}^{\text{law}}$.

As \mathcal{M} is closed under λ -directed colimits, we deduce by Proposition 7.8 that \mathfrak{M} is in \mathcal{M}^{law} if and only if all components of $T_{\mathfrak{M}}$ are in \mathcal{M} . As \mathcal{M} is closed under

composition with isomorphisms, we deduce that $m \in \mathcal{M}^{\text{mon}}$ if and only if $\mathcal{L}_m \in \mathcal{M}^{\text{law}}$ if and only if all components of $T_{\mathcal{L}_m} \cong m$ are in \mathcal{M} .

Similarly, assume e is a monad morphism whose components are all in \mathcal{E} . Then all the $A, \mathbb{1}$ components of \mathcal{L}_e are in \mathcal{E} , hence, by Proposition 7.6, \mathcal{L}_e is in \mathcal{E}^{law} , and e is in \mathcal{E}^{mon} . \blacksquare

Example 7-7. The category of finitary monads over **Set** has a factorisation system whose \mathcal{M} -class consists of the componentwise injective monad morphisms, and whose \mathcal{E} -class includes the componentwise surjective monad morphisms. Similarly, the category of countably ranked locally continuous monads over $\omega\mathbf{CPO}$ has a factorisation system whose \mathcal{M} -class consists of the componentwise full mono monad morphisms, and whose \mathcal{E} -class includes the componentwise dense monad morphisms. \square

Example 7-8. Let \mathbb{V} be any finite set with at least two elements denoting storable values. Take $\mathbb{C}\text{hor}$ to be \mathbb{V} . Consider the global state, environment, and overwrite monads for a single memory \mathbb{V} -cell i.e., $T_{\text{GS}(\mathbb{V})}$, $T_{\text{Env}(\mathbb{V})}$, and $T_{\text{OW}(\mathbb{V})}$, respectively (see Examples 2-1 and 2-3). Let $T_{\text{I}(\mathbb{V})}$, $T_{\text{O}(\mathbb{V})}$ be the monads for modelling input and output interactions (see Example 3-5). As we mentioned in Example 5-8, all these monads have finite rank.

Straightforward calculation shows that the components of the monad morphisms $m_{\{\text{lookup}\}} : T_{\text{Env}(\mathbb{V})} \rightarrow T_{\text{GS}(\mathbb{V})}$ and $m_{\{\text{update}\}} : T_{\text{OW}(\mathbb{V})} \rightarrow T_{\text{GS}(\mathbb{V})}$ from Example 2-3 are injective. Thus, $m_{\{\text{lookup}\}}, m_{\{\text{update}\}}$ are injective^{mon}-morphisms.

We define inductively the following four monad morphisms:

$$\begin{array}{ll}
e_{\{\text{lookup}\}} : T_{\text{I}(\mathbb{V})} & \rightarrow T_{\text{Env}(\mathbb{V})} & e_{\{\text{update}\}} : T_{\text{O}(\mathbb{V})} & \rightarrow T_{\text{OW}(\mathbb{V})} \\
e_{\{\text{lookup}\}} : x & \mapsto \lambda v. x & e_{\{\text{update}\}} : x & \mapsto \langle \mathbf{1}_1 \star, x \rangle \\
e_{\{\text{lookup}\}} : \langle I, \langle t_v \rangle \rangle & \mapsto \lambda v. e_{\{\text{lookup}\}}(t_v)(v) & e_{\{\text{update}\}} : \langle O, v_0, t \rangle & \mapsto \langle \mathbf{1}_2 v, x \rangle \\
& & \text{where } e_{\{\text{update}\}}(t) & = \langle \delta, x \rangle \\
& & \text{and } v & = \begin{cases} v & \delta = \mathbf{1}_1 \star \\ v' & \delta = \mathbf{1}_2 v' \end{cases}
\end{array}$$

$$\begin{array}{ll}
f_{\{\text{lookup}\}} : T_{\text{I}(\mathbb{V})} & \rightarrow T_{\text{GS}(\mathbb{V})} & f_{\{\text{update}\}} : T_{\text{O}(\mathbb{V})} & \rightarrow T_{\text{GS}(\mathbb{V})} \\
f_{\{\text{lookup}\}} : x & \mapsto \lambda v. \langle v, x \rangle & f_{\{\text{update}\}} : x & \mapsto \lambda v. \langle v, x \rangle \\
f_{\{\text{lookup}\}} : \langle I, \langle t_v \rangle \rangle & \mapsto \lambda v. f_{\{\text{lookup}\}}(t_v)(v) & f_{\{\text{update}\}} : \langle O, v_0, t \rangle & \mapsto \lambda v. f_{\{\text{lookup}\}}(t)(v_0)
\end{array}$$

Straightforward calculations show that these indeed are monad morphisms. To see

that the components of $e_{\{\text{lookup}\}}$ and $e_{\{\text{update}\}}$ are surjective functions, note that:

$$\begin{aligned} \lambda v.x_v &= e_{\{\text{lookup}\}}(\langle I, \langle x_v \rangle_{v \in \mathbb{V}} \rangle) \\ \langle \mathbf{1}_1 \star, x \rangle &= e_{\{\text{update}\}}(x) \quad \langle \mathbf{1}_2 v, x \rangle = e_{\{\text{update}\}}(\langle O, v, x \rangle) \end{aligned}$$

Finally, direct calculation shows that:

$$\begin{aligned} f_{\{\text{lookup}\}} : T_{\mathbf{I}(\mathbb{V})} &\xrightarrow{e_{\{\text{lookup}\}}} T_{\text{Env}(\mathbb{V})} \xrightarrow{m_{\{\text{lookup}\}}} T_{\text{GS}(\mathbb{V})} \\ f_{\{\text{update}\}} : T_{\mathbf{O}(\mathbb{V})} &\xrightarrow{e_{\{\text{update}\}}} T_{\text{OW}(\mathbb{V})} \xrightarrow{m_{\{\text{update}\}}} T_{\text{GS}(\mathbb{V})} \end{aligned}$$

Thus, these are the surjective^{mon}-injective^{mon} factorisations of the monad morphisms $f_{\{\text{lookup}\}}$ and $f_{\{\text{update}\}}$. Consequently, we have the surjective^{law}-injective^{law} factorisations:

$$\begin{aligned} \mathfrak{F}_{\{\text{lookup}\}} : \mathcal{L}_{\mathbf{I}(\mathbb{V})} &\xrightarrow{\mathfrak{E}_{\{\text{lookup}\}}} \mathcal{L}_{\text{Env}(\mathbb{V})} \xrightarrow{\mathfrak{M}_{\{\text{lookup}\}}} \mathcal{L}_{\text{GS}(\mathbb{V})} \\ \mathfrak{F}_{\{\text{update}\}} : \mathcal{L}_{\mathbf{O}(\mathbb{V})} &\xrightarrow{\mathfrak{E}_{\{\text{update}\}}} \mathcal{L}_{\text{OW}(\mathbb{V})} \xrightarrow{\mathfrak{M}_{\{\text{update}\}}} \mathcal{L}_{\text{GS}(\mathbb{V})} \end{aligned}$$

Note that if $|\mathbb{V}| \leq 1$, then $m_{\{\text{update}\}}$ is not componentwise injective. In this case, these are not surjective^{law}-injective^{law} factorisations. \square

7.3 Conservative restriction

For any effect hierarchy Σ , let $\langle \mathcal{V}, T \rangle$ be a CBPV model, let type be a type assignment for $|\Sigma|$ in \mathcal{V} , and, for all $\text{op} \in |\Sigma|$, an assignment of an algebraic operation of type $A \langle P \rangle$ for T . These data constitute a semantic model that ignores the effect hierarchy, and specifies meaning to all the effects together. Our goal is to construct a Σ -model that takes the effect hierarchy into account, when the CBPV model is given *algebraically*, as in Definition 2.13⁺. Recall that in this case, a λ -Power category replaces \mathcal{V} , the type assignment is λ -presentable, a λ -Lawvere \mathcal{V} -theory \mathcal{L} replaces T , and algebraic operations in this Lawvere theory replace the effect operations.

Our construction proceeds in two steps. First, we note that for every $\varepsilon \in \mathcal{E}$ there is an *initial* Lawvere theory $\mathcal{L}_{\langle \varepsilon, \text{type} \rangle}$ amongst all Lawvere theories that have an operation of type $P \langle A \rangle$ for every $\text{op} : P \langle A \rangle$ in ε . Because \mathcal{L} has operations of type $P \langle A \rangle$ for every $\text{op} : P \langle A \rangle$, initiality implies the existence of a unique Lawvere theory morphism $\mathfrak{F}_{\varepsilon} : \mathcal{L}_{\langle \varepsilon, \text{type} \rangle} \rightarrow \mathcal{L}$. The second step is to factorise this morphism using a factorisation system of $\mathbf{Law}_{\lambda} \mathcal{V}$:

$$\mathfrak{F}_{\varepsilon} : \mathcal{L}_{\langle \varepsilon, \text{type} \rangle} \xrightarrow{\mathfrak{E}_{\varepsilon}} \mathcal{L}_{\varepsilon} \xrightarrow{\mathfrak{M}_{\varepsilon}} \mathcal{L}$$

This factorisation yields a Σ -model, which we call the *conservative restriction model*.

We need a few auxiliary definitions.

Definition 7.10. Let \mathcal{V} be a λ -Power category with respect to the cartesian closed structure. A λ -presentable signature σ is a pair $\langle |\sigma|, \text{type} \rangle$ consisting of a set $|\sigma|$ and a λ -presentable type assignment type for $|\sigma|$ in \mathcal{V} .

Let σ be a λ -presentable signature. A σ -theory is a pair $\langle \mathcal{L}, \mathcal{L}[-] \rangle$, where:

- \mathcal{L} is a λ -Lawvere \mathcal{V} -theory, and
- $\mathcal{L}[-]$ assigns to each $\text{op} : A \langle P \rangle$ in $|\sigma|$ an operation $\mathcal{L}[\text{op}] : A \langle P \rangle$ in \mathcal{L} .

Let $\mathcal{L}, \widehat{\mathcal{L}}$ be two σ -theories. A morphism \mathfrak{T} of σ -theories from \mathcal{L} to $\widehat{\mathcal{L}}$ is a morphism of Lawvere theories $\mathfrak{T} : \mathcal{L} \rightarrow \widehat{\mathcal{L}}$ such that, for every $\text{op} \in |\sigma|$, \mathfrak{T} maps $\mathcal{L}[\text{op}]$ to $\widehat{\mathcal{L}}[\text{op}]$.

Example 7-9. Take \mathcal{V} to be **Set**, and let \mathcal{V} be any finite set denoting storable values. Take σ to be the signature with $|\sigma|$ a two element set $\{\text{lookup}, \text{update}\}$ and the finitely-presentable type assignment to be $\text{lookup} : \mathbb{V}$, $\text{update} : \mathbb{1} \langle \mathbb{V} \rangle$. Thus σ is a finitely-presentable type assignment.

The (finitary) Lawvere theory $\mathcal{L}_{\text{GS}(\mathbb{V})}$ is the Lawvere theory corresponding to the finitary global state monad $T_{\text{GS}(\mathbb{V})}$. As we saw in Example 6-12, this is a σ -theory. Moreover, the theory $\mathcal{L}_{\text{I/O}(\mathbb{V})}$ is also a σ -theory, where lookup is interpreted as the input operation and update as the output operation.

Finally, in Example 6-12 we saw a Lawvere theory morphism $\mathfrak{T} : \mathcal{L}_{\text{I/O}(\mathbb{V})} \rightarrow \mathcal{L}_{\text{GS}(\mathbb{V})}$ mapping input and output to look-up and update, respectively. Thus, \mathfrak{T} is a morphism of σ -theories. □

The first ingredient in our construction is the following theorem:

Theorem 7.11. For every λ -presentable signature σ in \mathcal{V} there exists an initial σ -theory \mathcal{L}_σ .

Proof

Our proof consists of two parts. First, we establish the existence of the initial σ -theory when $|\sigma|$ is a singleton $\{\text{op}\}$ using standard techniques for *free monads*. Once the existence of the $\mathcal{L}_{\{\text{op}:A \langle P \rangle\}}$ is established, we use the cocompleteness of $\mathbf{Law}_\lambda \mathcal{V}$, and show that the required σ -theory is the coproduct of these Lawvere theories, namely

$$\mathcal{L}_\sigma = \sum_{\substack{\text{op} \in |\sigma| \\ \text{op}:A \langle P \rangle}} \mathcal{L}_{\{\text{op}:A \langle P \rangle\}}$$

Our use of free monads is not essential, but merely a convenience, for we can thus refer to published work on free monads. Other techniques, e.g., enriched sketches [Kel82a] can be used too, but we will not elaborate on them further. In order to not tie the proof to free monads, we isolated their use to a small part of the proof, and the rest of the proof does not depend on the use of free monads.

Let Σ be the evident \mathcal{V} -endofunctor over \mathcal{V} given by

$$\Sigma := \prod_A \Sigma_P(-) = (P \times (-))^A$$

Note that the underlying ordinary functor $|\Sigma|_0$ preserves λ -directed colimits, as it is the composition of two adjoint functors between locally λ -presentable categories (see Theorem 5.17). Hyland et al. [HPP06, Section 2, prior to Example 6], employing techniques described by Kelly [Kel80], describe several sufficient conditions for the existence of the free monad for Σ . For our purposes, it suffices that $|\Sigma|_0$ is λ -ranked, \mathcal{V} -enriched, and \mathcal{V} is locally λ -presentable, and then the *free \mathcal{V} -monad* T_Σ for Σ exists. Its crucial three properties are:

- T_Σ is a λ -ranked \mathcal{V} -monad;
- T_Σ has a generic effect $\text{gen} : A \langle P \rangle$; and
- if T' is any other \mathcal{V} -monad and $\text{gen}' : A \langle P \rangle$ is a generic effect for T' , then there exists a unique \mathcal{V} -monad morphism from T_Σ to T' mapping gen to gen' .

By invoking Theorem 6.19, deduce that $\mathcal{L}_{\{\text{op}:A \langle P \rangle\}} := \mathcal{L}_{T_\Sigma}$ is the required Lawvere theory.

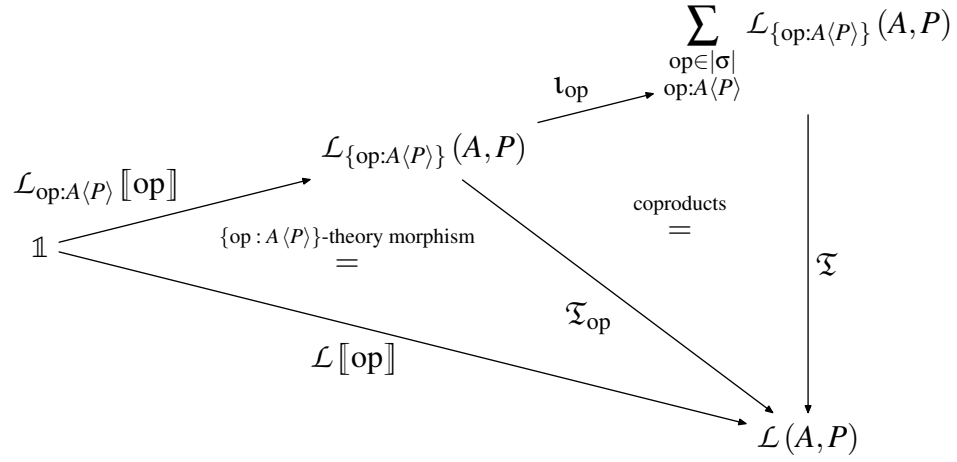
Next, for an arbitrary λ -presentable signature σ , choose:

$$\mathcal{L}_\sigma := \sum_{\substack{\text{op} \in |\sigma| \\ \text{op}:A \langle P \rangle}} \mathcal{L}_{\{\text{op}:A \langle P \rangle\}}$$

And interpret each $\text{op} \in |\sigma|$ as $\mathcal{L}_\sigma[\text{op}] := \mathfrak{t}_{\text{op}}(\mathcal{L}_{\{\text{op}:A \langle P \rangle\}}[\text{op}])$, i.e.:

$$\mathbb{1} \xrightarrow{\mathcal{L}_{\{\text{op}:A \langle P \rangle\}}[\text{op}]} |\mathcal{L}_{\{\text{op}:A \langle P \rangle\}}|(A, P) \xrightarrow{\mathfrak{t}_{\text{op}}} \sum_{\substack{\text{op} \in |\sigma| \\ \text{op}:A \langle P \rangle}} \mathcal{L}_{\{\text{op}:A \langle P \rangle\}}$$

If \mathcal{L} is any other σ -theory, then, for each $\text{op} \in |\sigma|$, \mathcal{L} is a $\{\text{op} : A \langle P \rangle\}$ -theory, and there is a unique morphism $\mathfrak{T}_{\text{op}} : \mathcal{L}_{\{\text{op}:A \langle P \rangle\}} \rightarrow \mathcal{L}$ preserving the interpretation of op . Choose as $\mathfrak{T} : \mathcal{L}_\sigma \rightarrow \mathcal{L}$ the coproduct morphism $\mathfrak{T} := [\mathfrak{T}_{\text{op}}]_{\text{op} \in |\sigma|}$. Then we have:



Conversely, if $\hat{\mathfrak{T}} : \mathcal{L}_\sigma \rightarrow \mathcal{L}$ is any other σ -theory morphism, then $\hat{\mathfrak{T}}_{\text{op}} := \hat{\mathfrak{T}} \circ \iota_{\text{op}}$ is a $\{\text{op} : A \langle P \rangle\}$ -theory morphism from $\mathcal{L}_{\{\text{op}:A \langle P \rangle\}}$ to \mathcal{L} , hence $\hat{\mathfrak{T}}_{\text{op}} = \mathfrak{T}_{\text{op}}$. For all $\text{op} \in |\sigma|$, we have $\hat{\mathfrak{T}} \circ \iota_{\text{op}} = \mathfrak{T} \circ \iota_{\text{op}}$, hence $\hat{\mathfrak{T}} = \mathfrak{T}$, and we have uniqueness. ■

Example 7-10. Let \mathbb{V} be a finite set denoting storable values. Consider the signature σ from Example 7-9, $\{\text{lookup} : \mathbb{V}, \text{update} : \mathbb{1} \langle \mathbb{V} \rangle\}$. Consider the input/output theory $\mathcal{L}_{\text{I/O}(\mathbb{V})}$ from Example 7-9. We will show $\mathcal{L}_{\text{I/O}(\mathbb{V})}$ is the initial σ -theory.

We already saw that $\mathcal{L}_{\text{I/O}(\mathbb{V})}$ is a σ -theory. Let $\langle \mathcal{L}, \mathcal{L}[-] \rangle$ be any other σ -theory. We therefore have a corresponding monad T over **Set** with algebraic operations $\text{lookup} : \mathbb{V}$ and $\text{update} : \mathbb{1} \langle \mathbb{V} \rangle$.

For every set X , define inductively:

$$\begin{aligned} f_X : T_{\text{I/O}(\mathbb{V})}X &\rightarrow TX \\ f_X : x &\mapsto \eta(x) \\ \langle I, \langle t_v \rangle_{v \in \mathbb{V}} \rangle &\mapsto \text{lookup}(\lambda v. f_X(t_v))(\star) \\ \langle O, v, t \rangle &\mapsto \text{update}(\lambda \star. f_X(t))(v) \end{aligned}$$

To establish that f is a monad morphism from $T_{\text{I/O}(\mathbb{V})}$ to T , we use the following special case of a result by Plotkin and Power [PP03, Proposition 1], adapted to include parameter types:

Let T be a monad over **Set**, and $\text{op} : A \langle P \rangle$ an algebraic operation for T . Then:

1. the transformation $\text{op} : (T-)^A \rightarrow (T-)^P$ is natural; and
2. this transformation respects the monadic multiplication μ in the sense that, for all $\lambda a. \hat{t}_a$ in $(T^2X)^A$ and $p \in P$:

$$\text{op}(\lambda a. \mu(\hat{t}_a))(p) = \lambda p. \mu(\text{op}(\lambda a. \hat{t}_a)(p))$$

Straightforward calculations show that the naturality of η , lookup, and update implies that of f , and that, by definition, f preserves the monadic unit. A straightforward inductive argument using the monad laws and property 2 above shows that f preserves the monadic multiplication.

Straightforward calculation shows that f preserves the operations. Indeed, for every $\lambda v.t_v$ in $(T_{I/O(\mathbb{V})}X)^{\mathbb{V}}$, we have:

$$\begin{array}{ccc}
 \lambda v.t_v & \xrightarrow{f^{\mathbb{V}}} & \lambda v.f(t_v) \\
 \downarrow \text{input} & & \downarrow \text{lookup} \\
 \langle I, \langle t_v \rangle \rangle & \xrightarrow{f^{\mathbb{1}}} & \lambda \star.(\text{lookup}(\lambda v.f(t_v))(\star)) \\
 & & \parallel \\
 & & \text{lookup}(\lambda v.f(t_v))
 \end{array}$$

Thus, f maps input to lookup. A similar calculation shows that f maps output to update. Therefore, using the equivalence between Lawvere theories and ranked monads, we deduce there exists a σ -theory morphism $\mathfrak{T} : \mathcal{L}_{I/O(\mathbb{V})} \rightarrow \mathcal{L}$, corresponding to f .

To conclude, assume $\hat{\mathfrak{T}} : \mathcal{L}_{I/O(\mathbb{V})} \rightarrow \mathcal{L}$ is any σ -theory morphism. We therefore have a monad morphism $\hat{f} : T_{I/O(\mathbb{V})} \rightarrow T$, mapping input and output to lookup and update, respectively. A straightforward inductive argument using the preservation of the unit and the mapping of the operations shows that \hat{f} and f coincide. We will only demonstrate how f and \hat{f} agree on $\langle I, \langle t_v \rangle \rangle$, if they agree over all components t_v . As \hat{f} maps input to lookup, we have:

$$\begin{array}{ccc}
 \lambda v.t_v & \xrightarrow{\hat{f}^{\mathbb{V}}} & \lambda v.\hat{f}(t_v) \\
 \downarrow \text{input} & & \downarrow \text{lookup} \\
 \langle I, \langle t_v \rangle \rangle & \xrightarrow{\hat{f}^{\mathbb{1}}} & \lambda \star.\hat{f}(\langle I, \langle t_v \rangle \rangle) \\
 & & \parallel \\
 & & \text{lookup}(\lambda v.\hat{f}(t_v))
 \end{array}$$

Thus,

$$\hat{f}(\langle I, \langle t_v \rangle \rangle) = \text{lookup}(\lambda v.\hat{f}(t_v))(\star) \stackrel{\text{induction hypothesis}}{=} \text{lookup}(\lambda v.f(t_v))(\star) \stackrel{f \text{ def.}}{=} f(\langle I, \langle t_v \rangle \rangle)$$

and \hat{f} and f agree on $\langle I, \langle t_v \rangle \rangle$.

As f, \hat{f} coincide, \mathfrak{T} and $\hat{\mathfrak{T}}$ coincide. Thus, $\mathcal{L}_{I/O(\mathbb{V})}$ is the initial σ -theory \mathcal{L}_σ . Similar arguments show that the theories $\mathcal{L}_{I(\mathbb{V})}$ and $\mathcal{L}_{O(\mathbb{V})}$ corresponding to the monads $T_{I(\mathbb{V})}$ and $T_{O(\mathbb{V})}$, respectively, are the initial $\{\text{lookup} : \mathbb{V}\}$ -, and $\{\text{update} : \mathbb{1} \langle \mathbb{V} \rangle\}$ -theories, respectively. \square

The following theorem is our central construction:

Theorem 7.12 (conservative restriction model). *Let Π be a set, and*

$$\mathcal{M} = \langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}, \mathcal{L}[-] \rangle$$

an algebraic CBPV Π -model. For every effect hierarchy Σ whose set of operation is Π , and for every factorisation system $\langle \mathcal{E}^{\text{law}}, \mathcal{M}^{\text{law}} \rangle$ in $\mathbf{Law}_\lambda \mathcal{V}$, the following data define an algebraic Σ -model

$$\mathcal{M}_\# := \langle \lambda, \mathcal{V}, \text{type}, \mathcal{L}_-, \mathcal{L}_-[-] \rangle$$

together with the auxiliary data $\langle \sigma_-, \mathfrak{T}_-, \mathfrak{E}_-, \mathfrak{M}_-, \mathfrak{S}_- \rangle$, where:

- σ_- assigns to every $\varepsilon \in \mathcal{E}$ the λ -presentable signature given by restricting type to the subset $\varepsilon \subseteq \Pi$, $\sigma_\varepsilon := \langle \varepsilon, \text{type}|_\varepsilon \rangle$;
- \mathfrak{T}_- assigns to every $\varepsilon \in \mathcal{E}$ the unique morphism $\mathfrak{T}_\varepsilon : \mathcal{L}_{\sigma_\varepsilon} \rightarrow \mathcal{L}$ mapping, for every $\text{op} \in \varepsilon$, the operation $\mathcal{L}_{\sigma_\varepsilon}[\text{op}]$ to $\mathcal{L}[\text{op}]$, where $\mathcal{L}_{\sigma_\varepsilon}$ is the initial σ_ε theory from Theorem 7.11;
- for every $\varepsilon \in \mathcal{E}$, $\langle \mathcal{L}_\varepsilon, \mathfrak{E}_\varepsilon, \mathfrak{M}_\varepsilon \rangle$ is a specified $\langle \mathcal{E}^{\text{law}}, \mathcal{M}^{\text{law}} \rangle$ factorisation:

$$\mathfrak{T}_\varepsilon : \mathcal{L}_{\sigma_\varepsilon} \xrightarrow{\mathfrak{E}_\varepsilon} \mathcal{L}_\varepsilon \xrightarrow{\mathfrak{M}_\varepsilon} \mathcal{L}$$

- $\mathcal{L}_-[-]$ assigns to each $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$ the unique operation $\mathcal{L}_\varepsilon[\text{op}]$ in \mathcal{L}_ε such that \mathfrak{E}_ε maps $\mathcal{L}_{\sigma_\varepsilon}[\text{op}]$ to $\mathcal{L}_\varepsilon[\text{op}]$, and then, \mathfrak{M}_ε maps $\mathcal{L}_\varepsilon[\text{op}]$ to $\mathcal{L}[\text{op}]$.
- \mathfrak{S}_- assigns to each $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} the unique morphism $\mathfrak{S}_{\varepsilon \subseteq \varepsilon'} : \mathcal{L}_{\sigma_\varepsilon} \rightarrow \mathcal{L}_{\sigma_{\varepsilon'}}$ that, for every $\text{op} \in \varepsilon$, maps $\mathcal{L}_{\sigma_\varepsilon}[\text{op}]$ to $\mathcal{L}_{\sigma_{\varepsilon'}}[\text{op}]$;
- \mathcal{L}_- assigns to every $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} the unique fill-in morphism:

$$\begin{array}{ccc} \mathcal{L}_{\sigma_\varepsilon} & \xrightarrow{\mathfrak{E}_\varepsilon} & \mathcal{L}_\varepsilon \\ \downarrow \mathfrak{S}_{\varepsilon \subseteq \varepsilon'} & \searrow \mathcal{L}_{\varepsilon \subseteq \varepsilon'} & \downarrow \mathfrak{M}_\varepsilon \\ \mathcal{L}_{\sigma_{\varepsilon'}} & \xrightarrow{\mathfrak{M}_{\varepsilon'}} & \mathcal{L} \end{array}$$

(with $\mathcal{L}_{\sigma_\varepsilon} \xrightarrow{=} \mathcal{L}_{\sigma_{\varepsilon'}} \xrightarrow{=} \mathcal{L}$ and $\mathcal{L}_{\sigma_{\varepsilon'}} \xrightarrow{=} \mathcal{L}$ in the diagram)

and, moreover, this morphism is in \mathcal{M}^{law} .

We call \mathcal{M}_{\sharp} the conservative restriction Σ -model for the given Π -model, relative to the given factorisation system.

Proof

First, we show these data are well-defined. Consider any $\varepsilon \in \mathcal{E}$. As type is a λ -presentable type assignment, σ_{ε} is a well-defined λ -presentable type assignment in \mathcal{V} . If we restrict $\mathcal{L}[-]$ to ε , we obtain a σ_{ε} -theory $\langle \mathcal{L}, \mathcal{L}[-]|_{\varepsilon} \rangle$. By initiality, there exists a unique morphism $\mathfrak{T}_{\varepsilon} : \mathcal{L}_{\sigma_{\varepsilon}} \rightarrow \mathcal{L}$ preserving the interpretations of all the operations in ε . Thus $\mathfrak{T}_{\varepsilon}$ is well-defined, and consequently, so are $\mathcal{L}_{\varepsilon}$, $\mathfrak{E}_{\varepsilon}$, and $\mathfrak{M}_{\varepsilon}$.

Consider any $\text{op} \in \varepsilon$. By Corollary 2.9 and Theorem 6.19, there exists a unique algebraic operation $\mathcal{L}_{\varepsilon}[\text{op}]$ for $\mathcal{L}_{\varepsilon}$ such that $\mathfrak{E}_{\varepsilon}$ maps $\mathcal{L}_{\sigma_{\varepsilon}}[\text{op}]$ to $\mathcal{L}_{\varepsilon}[\text{op}]$, and $\mathcal{L}_{\varepsilon}[\text{op}]$ is well-defined. Since both \mathfrak{T} and \mathfrak{E} preserve the interpretation of op , $\mathfrak{T}_{\varepsilon} = \mathfrak{M}_{\varepsilon} \circ \mathfrak{E}_{\varepsilon}$, we deduce by Corollary 2.9 and Theorem 6.19 that $\mathcal{M}_{\varepsilon}$ also preserves the interpretation of op .

Consider any $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} . By restricting $\mathcal{L}_{\varepsilon'}[-]$ to ε , we obtain a σ_{ε} -theory $\langle \mathcal{L}_{\varepsilon'}, \mathcal{L}_{\varepsilon'}[-]|_{\varepsilon} \rangle$. Therefore, there exists a unique morphism $\mathfrak{S}_{\varepsilon \subseteq \varepsilon'} : \mathcal{L}_{\sigma_{\varepsilon}} \rightarrow \mathcal{L}_{\varepsilon'}$ preserving the interpretations of the effects in ε , and \mathfrak{S}_{-} is well-defined.

Thus, both $\mathfrak{M}_{\varepsilon} \circ \mathfrak{E}_{\varepsilon}$ and $\mathfrak{M}_{\varepsilon'} \circ \mathfrak{S}_{\varepsilon \subseteq \varepsilon'}$ are σ_{ε} -theory homomorphisms. Initiality of $\mathcal{L}_{\sigma_{\varepsilon}}$ means that

$$\begin{array}{ccc}
 \mathcal{L}_{\sigma_{\varepsilon}} & \xrightarrow{\mathfrak{E}_{\varepsilon}} & \mathcal{L}_{\varepsilon} \\
 \mathfrak{S}_{\varepsilon \subseteq \varepsilon'} \downarrow & = & \downarrow \mathfrak{M}_{\varepsilon} \\
 \mathcal{L}_{\varepsilon'} & \xrightarrow{\mathfrak{M}_{\varepsilon'}} & \mathcal{L}
 \end{array}$$

By the orthogonality property of factorisation systems, we deduce the existence of a unique fill-in morphism:

$$\begin{array}{ccc}
 \mathcal{L}_{\sigma_{\varepsilon}} & \xrightarrow{\mathfrak{E}_{\varepsilon}} & \mathcal{L}_{\varepsilon} \\
 \mathfrak{S}_{\varepsilon \subseteq \varepsilon'} \downarrow & \begin{array}{c} = \\ \mathcal{L}_{\varepsilon \subseteq \varepsilon'} \\ = \end{array} & \downarrow \mathfrak{M}_{\varepsilon} \\
 \mathcal{L}_{\varepsilon'} & \xrightarrow{\mathfrak{M}_{\varepsilon'}} & \mathcal{L}
 \end{array} \tag{7.1}$$

The commutativity of these two triangles implies, again by Corollary 2.9 and Theorem 6.19, that $\mathcal{L}_{\varepsilon \subseteq \varepsilon'}$ preserves the operations in ε . Also, note that Bousfield's factorisation theorem (Theorem 7.4(3)) implies that $\mathcal{L}_{\varepsilon \subseteq \varepsilon'}$ is an \mathcal{M}^{law} -morphism.

These data do yield an algebraic Σ -model. Indeed, by their choice, λ is a regular cardinal, and \mathcal{V} a λ -Power category. Our choice of \mathcal{L}_- indeed yields a functor from \mathcal{E} to $\mathbf{Law}_\lambda \mathcal{V}$. First, note that \mathfrak{E}_ε is a σ_ε -morphism from $\mathcal{L}_{\sigma_\varepsilon}$ to \mathcal{L}_ε , hence initiality implies $\mathfrak{E}_\varepsilon = \mathfrak{S}_{\varepsilon \subseteq \varepsilon}$. Consequently,

$$\begin{array}{ccc}
 \mathcal{L}_{\sigma_\varepsilon} & \xrightarrow{\mathfrak{E}_\varepsilon} & \mathcal{L}_\varepsilon \\
 \mathfrak{S}_{\varepsilon \subseteq \varepsilon} \downarrow & \begin{array}{c} \parallel \\ \parallel \end{array} & \downarrow \mathfrak{M}_\varepsilon \\
 \mathcal{L}_\varepsilon & \xrightarrow{\mathfrak{M}_\varepsilon} & \mathcal{L}
 \end{array}$$

By orthogonality we deduce that $\mathcal{L}_{\varepsilon \subseteq \varepsilon} = \text{id}$. Consider any $\varepsilon \subseteq \varepsilon' \subseteq \varepsilon''$ in \mathcal{E} , then $\mathcal{L}_{\varepsilon' \subseteq \varepsilon''} \circ \mathfrak{S}_{\varepsilon \subseteq \varepsilon'}$ is a σ_ε -theory morphism from $\mathcal{L}_{\sigma_\varepsilon}$ to $\mathcal{L}_{\varepsilon''}$. By initiality,

$$\begin{array}{ccc}
 \mathcal{L}_{\sigma_\varepsilon} & \xrightarrow{\mathfrak{S}_{\varepsilon \subseteq \varepsilon''}} & \mathcal{L}_{\varepsilon''} \\
 \mathfrak{S}_{\varepsilon \subseteq \varepsilon'} \searrow & \begin{array}{c} \parallel \\ \parallel \end{array} & \nearrow \mathcal{L}_{\varepsilon' \subseteq \varepsilon''} \\
 & \mathcal{L}_{\varepsilon'} &
 \end{array} \tag{7.2}$$

Thus, on the one hand, we have:

$$\begin{array}{ccccc}
 \mathcal{L}_{\sigma_\varepsilon} & \xrightarrow{\mathfrak{E}_\varepsilon} & \mathcal{L}_\varepsilon & & \\
 \downarrow \mathfrak{S}_{\varepsilon \subseteq \varepsilon''} & \searrow \mathfrak{S}_{\varepsilon \subseteq \varepsilon'} & \mathcal{L}_{\varepsilon'} & \swarrow \mathcal{L}_{\varepsilon' \subseteq \varepsilon''} & \downarrow \mathfrak{M}_\varepsilon \\
 \mathcal{L}_{\varepsilon''} & \xrightarrow{\mathfrak{M}_{\varepsilon''}} & \mathcal{L} & & \\
 \text{Diagram 7.1} & \text{Diagram 7.1} & \text{Diagram 7.1} & & \\
 \text{=} & \text{=} & \text{=} & & \\
 \text{Diagram 7.2} & \text{=} & \text{Diagram 7.1} & & \\
 \text{=} & \text{=} & \text{=} & & \\
 \mathcal{L}_{\varepsilon' \subseteq \varepsilon''} & \text{=} & \mathfrak{M}_{\varepsilon'} & &
 \end{array}$$

Thus, by definition, $\mathcal{L}_{\varepsilon \subseteq \varepsilon''} = \mathcal{L}_{\varepsilon' \subseteq \varepsilon''} \circ \mathcal{L}_{\varepsilon \subseteq \varepsilon'}$. Thus, we have a functor $\mathcal{L}: \mathcal{E} \rightarrow \mathbf{Law}_\lambda \mathcal{V}$. By fiat, $\mathcal{L}_\varepsilon[-]$ assigns operations as required, and we have seen that \mathcal{L}_- preserves them. ■

Example 7-11. Let Σ be the effect hierarchy for global state, and \mathbb{V} a finite set such that $|\mathbb{V}| \geq 2$ denoting storable values. Consider the algebraic CBPV Π -model for global state from Example 6-14, and the surjective^{law}-injective^{law} factorisation system arising from the surjective-injective factorisation system of **Set** by Theorem 7.5. We work out the explicit description of the conservative restriction model.

Comparing the construction of the initial morphism $f : T_{I/O(\mathbb{V})} \rightarrow T_{GS(\mathbb{V})}$ in Example 7-10 to the definition of the morphisms $\mathfrak{T} : \mathcal{L}_{I/O(\mathbb{V})} \rightarrow \mathcal{L}_{GS(\mathbb{V})}$ in Example 6-12, and $\mathfrak{T}_{\{\text{lookup}\}} : \mathcal{L}_{I(\mathbb{V})} \rightarrow \mathcal{L}_{\text{Env}(\mathbb{V})}$ and $\mathfrak{T}_{\{\text{update}\}} : \mathcal{L}_{O(\mathbb{V})} \rightarrow \mathcal{L}_{\text{OW}(\mathbb{V})}$ from Example 7-8, shows these are indeed the initial Π -, $\{\text{lookup}\}$ -, and $\{\text{update}\}$ -theory morphisms, respectively. The initial \emptyset -theory morphism $\mathfrak{T}_\emptyset : \mathbf{Pres}_{\mathbb{X}_0}^{\text{op}}(\mathbf{Set}) \rightarrow \mathcal{L}_{GS(\mathbb{V})}$ is the functor $\mathcal{L}_{GS(\mathbb{V})} \llbracket - \rrbracket$.

In Example 6-12 we noted that the monad morphism corresponding to \mathfrak{T}_Π is surjective, From Proposition 7.6 we conclude that \mathfrak{T}_Π is in the class surjective^{law}. Therefore, we have the factorisation:

$$\mathfrak{T}_\Pi : \mathcal{L}_{I/O(\mathbb{V})} \xrightarrow{\mathfrak{T}_\Pi} \mathcal{L}_{GS(\mathbb{V})} \xrightarrow{\text{id}} \mathcal{L}_{GS(\mathbb{V})}$$

In Example 7-8 we presented a factorisation of $\mathfrak{T}_{\{\text{lookup}\}}$ and $\mathfrak{T}_{\{\text{update}\}}$:

$$\begin{array}{ccc} \mathfrak{T}_{\{\text{lookup}\}} : \mathcal{L}_{I(\mathbb{V})} & \xrightarrow{\mathcal{E}_{\{\text{lookup}\}}} & \mathcal{L}_{\text{Env}(\mathbb{V})} \xrightarrow{\mathfrak{M}_{\{\text{lookup}\}}} \mathcal{L}_{GS(\mathbb{V})} \\ \mathfrak{T}_{\{\text{update}\}} : \mathcal{L}_{O(\mathbb{V})} & \xrightarrow{\mathcal{E}_{\{\text{update}\}}} & \mathcal{L}_{\text{OW}(\mathbb{V})} \xrightarrow{\mathfrak{M}_{\{\text{update}\}}} \mathcal{L}_{GS(\mathbb{V})} \end{array}$$

The $X, \mathbb{1}$ components of this functor act on morphisms by post-composing with the monadic unit $\eta_{GS(\mathbb{V})}$ (see Example 6-10). As \mathbb{V} is non-empty, this monadic unit is injective, hence post-composing with it yields a monomorphism between the homsets. Thus, \mathfrak{T}_\emptyset is a injective^{law}-morphism, and we have the factorisation:

$$\mathfrak{T}_\emptyset : \mathbf{Pres}_{\mathbb{X}_0}^{\text{op}}(\mathbf{Set}) \xrightarrow{\text{id}} \mathbf{Pres}_{\mathbb{X}_0}^{\text{op}}(\mathbf{Set}) \xrightarrow{\mathfrak{T}_\emptyset} \mathcal{L}_{GS(\mathbb{V})}$$

Therefore, the object part of the functor \mathcal{L}_- for the conservative restriction model agrees with the model in Example 6-14. We know that the injective^{law}-part of the factorisation preserves the operations, hence uniquely determines them. As these arrows are precisely the morphisms $\mathcal{L}_{\varepsilon \subseteq \Pi}$ in the model from Example 6-14, we know they preserve the operations. Therefore, the operations in the conservative restriction model coincide with the operations in the model from Example 6-14. The fact that the injective^{law}-part of the factorisation coincides with $\mathcal{L}_{\varepsilon \subseteq \Pi}$ in the model from Example 6-14 also means the lower-right triangle in the orthogonality square defining $\mathcal{L}_{\varepsilon \subseteq \varepsilon'}$

commutes. Because they preserve the operations, initiality of \mathcal{L}_{σ_e} means the upper-left triangle also commutes. Therefore, the morphism parts of the functors \mathcal{L}_- of the two models coincide.

In conclusion, the model in Example 6-14 is the conservative restriction model. \square

This example illustrates that the conservative restriction model construction uniformly gives rise to a natural, intuitive Σ -model that seemed previously non-uniform. It also demonstrates that the conservative restriction models allow us to avoid explicitly *specifying* an exponentially large structure. Instead, we only need to define the desired algebraic CBPV model, and then *derive* the data we require as properties of this structure. As we will see, the conservative restriction model is tightly related to the underlying CBPV model that gives rise to it. However, specifying a Lawvere theory is still an elaborate process: even when the theory is given as a monad, we need to enrich it, and verify it has a rank.

In summary, we defined the hierarchical algebraic models, presented the categorical conservative restriction construction, and showed that when we apply it to global state we obtain the global state hierarchical model.

Chapter 8

Presentation models

We don't use dollars to represent,

We just use our inner sense and talent

—Black Eyed Peas



In this chapter we present a concrete and syntactic account of our *conservative restriction* construction by using standard concepts from *universal algebra* and *equational logic*. This concrete description, which uses *presentations* consisting of terms and equations, allows us to state concretely and strengthen the results of the previous chapter.

First, in Section 8.1, we recount the relevant results and terminology from universal algebra and equational logic. Next, in Section 8.2, we describe their connection to strong monads and Lawvere theories. Then, in Section 8.3, we complement the account of the previous chapter by studying translations of presentations. Finally, in Section 8.4, we describe the conservative restriction construction.

8.1 Universal algebra and equational logic

This section briefly recounts standard notions and results from universal algebra. For a thorough introduction, see, e.g., Burris and Sankappanavar [BS81, Chapter II, Sections 1 and 8].

A *signature* σ is a pair $\langle \pi, \text{ar} \rangle$ where

- π is a set whose elements f, g, h we call *operation symbols*; and
- ar assigns to each f in π a natural number $\text{ar}(f)$ called its *arity*.

When $\text{ar}(f) = n$, we say that $f : n$ in σ . Operation symbols with arity n are called n -ary operations, and nullary operations, $f : 0$, are called *constants*.

Example 8-1. The signature for semilattices consists of a single binary operation $\vee : 2$ called *join*. □

Example 8-2. The signature for monoids consists of a binary operation $\cdot : 2$ called *multiplication*, and a constant $1 : 0$ called *unit*. □

Example 8-3. Let $\mathbb{V} = \{v_1, \dots, v_n\}$ be a finite set, $n \geq 2$ denoting storable values. The signature for *mnemoids*¹ consists of an n -ary operation symbol $\text{lookup} : n$ and n different unary operation symbols $\text{update}_{v_i} : 1$, for all $i = 1, \dots, n$. Note that this signature depends on the enumeration order of \mathbb{V} . □

Let σ be a signature, and X be any set. We consider the elements of X as *variables*, and define the set of σ -terms over X , $\text{Terms}_\sigma(X)$ by induction:

- For every variable $x \in X$, we have $x \in \text{Terms}_\sigma(X)$.
- For every $f : n$, if $t_1, \dots, t_n \in \text{Terms}_\sigma(X)$, then $f(t_1, \dots, t_n) \in \text{Terms}_\sigma(X)$.

Given a σ -term t over X , we define the set $\text{var}(t) \subseteq X$ of variables appearing in t by induction:

- $\text{var}(x) := \{x\}$; and
- $\text{var}(f(t_1, \dots, t_n)) := \bigcup_{i=1}^n \text{var}(t_i)$.

A σ -equation e over X is a pair $\langle t, s \rangle$ of two σ -terms over X , written as $t = s$. This notation may cause confusion between an equation such as $1 \cdot \mathbf{x} = \mathbf{x}$ and syntactic equality which differentiates $1 \cdot \mathbf{x}$ from \mathbf{x} . In the following, we explicitly note when we refer to syntactic equality between terms. We extend the variable-set function to equations by setting $\text{var}(t = s) := \text{var}(t) \cup \text{var}(s)$.

We will henceforth fix a countably infinite set of variables \mathbb{V}_{or} , whose elements are denoted by $\mathbf{x}, \mathbf{y}, \mathbf{z}$ and their subscripts and superscripts $\mathbf{x}_1, \mathbf{y}^2, \mathbf{z}_2^1$, etc. Thus, Terms_σ refers to $\text{Terms}_\sigma(\mathbb{V}_{\text{or}})$.

Definition 8.1. A presentation Ax is a pair $\langle \sigma, E \rangle$ consisting of a signature σ and a set E of σ -equations (over the distinguished set of variables \mathbb{V}_{or}).

¹Paul-André Melliès coined the term ‘mnemoid’ in unpublished work from 2010.

We will usually define presentations by specifying their equations only, when their signature can be inferred.

Example 8-4. The presentation of semilattices consists of the associativity, commutativity and idempotency equations:

$$\mathbf{x} \vee (\mathbf{y} \vee \mathbf{z}) = (\mathbf{x} \vee \mathbf{y}) \vee \mathbf{z} \qquad \mathbf{x} \vee \mathbf{y} = \mathbf{y} \vee \mathbf{x} \qquad \mathbf{x} \vee \mathbf{x} = \mathbf{x} \qquad \square$$

Example 8-5. The presentation of monoids consists of the associativity equation and neutrality of the unit:

$$\mathbf{x} \cdot (\mathbf{y} \cdot \mathbf{z}) = (\mathbf{x} \cdot \mathbf{y}) \cdot \mathbf{z} \qquad \mathbf{1} \cdot \mathbf{x} = \mathbf{x} \qquad \mathbf{x} \cdot \mathbf{1} = \mathbf{x} \qquad \square$$

The following presentation of mnemoids is due to Melliés [Mel10]:

Example 8-6. Let $\mathbb{V} = \{v_1, \dots, v_n\}$ be a finite non-empty set denoting storable values and $\sigma_{\text{GS}(\mathbb{V})}$ be the signature for \mathbb{V} -mnemoids given in Example 8-3. The presentation for mnemoids consists of the following three equation schemas:

$$\begin{aligned} \text{lookup}(\text{update}_{v_1}(\mathbf{x}), \dots, \text{update}_{v_n}(\mathbf{x})) &= \mathbf{x} \\ \text{update}_{v_i}(\text{lookup}(\mathbf{x}_1, \dots, \mathbf{x}_n)) &= \text{update}_{v_i}(\mathbf{x}_i) \\ \text{update}_v(\text{update}_u(\mathbf{x})) &= \text{update}_u(\mathbf{x}) \end{aligned}$$

These three equations have an operational reading in terms of interactions with a single global memory \mathbb{V} -cell. The first equation states that a computation that first reads the state, and, depending on the stored value v , updates the cell to contain v , and then carries on executing \mathbf{x} is identical to the computation that immediately executes \mathbf{x} . The second equation states that a look-up proceeding an update yields the updated value. The last equation states that more recent updates erase previous updates. \square

Let σ be a signature. We call the presentation $\langle \sigma, \emptyset \rangle$ the *free presentation over σ* .

Example 8-7. Let $\text{Char} = \{c_1, \dots, c_n\}$ be a finite set denoting I/O terminal characters. The presentation of terminal I/O is the free presentation over the signature consisting of the an n -ary operation $\text{input} : n$ and n unary operations $\text{output}_{c_1}, \dots, \text{output}_{c_n}$. This presentation has the same signature as the mnemoid presentation, but no equations. \square

Example 8-8. Let \mathbb{E} be any non-empty set, possibly infinite, denoting possible exceptions. The presentation of \mathbb{E} -exceptions is the free presentation over the signature consisting of a constant raise_e for every $e \in \mathbb{E}$. \square

Let σ be a signature. A σ -algebra \underline{B} is a pair $\langle |\underline{B}|, \underline{B}[-] \rangle$ consisting of a set $|\underline{B}|$, and an assignment mapping each $f : n$ in σ to a function $\underline{B}[f] : |\underline{B}|^n \rightarrow |\underline{B}|$. The set $|\sigma|$ is called the *carrier* of the algebra. The set $\text{Terms}_\sigma(X)$ has an evident σ -algebra structure, which we call the *term algebra*.

Let \underline{B} be a σ -algebra and X a set. A *valuation over X in \underline{B}* is a function $\rho : X \rightarrow |\underline{B}|$. Given a term t , we say that a valuation over X *suffices for t* provided $\text{var}(t) \subseteq X$. We define valuations sufficing for equations similarly. We can extend any valuation ρ over X in \underline{B} to a function $\underline{B}[-] \rho : \text{Terms}_\sigma(X) \rightarrow |\underline{B}|$ inductively as follows:

- $\underline{B}[\mathbf{x}] \rho := \rho(\mathbf{x})$
- $\underline{B}[f(t_1, \dots, t_n)] \rho := \underline{B}[f](\underline{B}[t_1] \rho, \dots, \underline{B}[t_n] \rho)$

Let $t = s$ be a σ -equation, \underline{B} a σ -algebra, and ρ a valuation in \underline{B} sufficient for e . We say that $t = s$ is *true under the valuation ρ* if $\llbracket t \rrbracket \rho = \llbracket s \rrbracket \rho$. We say that a σ -equation is *true in a σ -algebra* if it is true under all valuations sufficient for it.

Definition 8.2. Let $Ax = \langle \sigma, E \rangle$ be a presentation. An *Ax -model* is a σ -algebra in which all the equations in E are true.

The models for the semilattice presentation are precisely the semilattices, and similarly for monoids. We call models of the mnemoid presentation mnemoids.

Let Ax be a presentation. We say that Ax *entails* an equation e , and write $Ax \models e$, if the equation e is valid in all Ax -models.

The following observation is due to Melliés [Mel10]:

Example 8-9. The presentation for mnemoids entails the following equation:

$$\text{lookup}(\text{lookup}(\mathbf{x}_1^1, \dots, \mathbf{x}_n^1), \dots, \text{lookup}(\text{lookup}(\mathbf{x}_1^n, \dots, \mathbf{x}_n^n))) = \text{lookup}(\mathbf{x}_1^1, \dots, \mathbf{x}_n^n)$$

The operational reading of this equation is that the memory cell does not change its contents between consecutive look-ups. \square

Let $\underline{B}, \underline{C}$ be two σ -algebras. A *homomorphism $h : \underline{B} \rightarrow \underline{C}$* is a function $h : |\underline{B}| \rightarrow |\underline{C}|$ such that, for every $f : n$ in \underline{B} and $b_1, \dots, b_n \in |\underline{B}|$:

$$h(\underline{B}[f](b_1, \dots, b_n)) = \underline{C}[f](h(b_1), \dots, h(b_n))$$

Let $Ax = \langle \sigma, E \rangle$ be a presentation. A *homomorphism between two Ax -models* is a homomorphism between them as σ -algebras.

Let Ax be a presentation and X a set. A *free Ax -model over X* is a pair $\langle \underline{B}, \eta \rangle$ consisting of an Ax -model \underline{B} and a function $\eta : X \rightarrow |\underline{B}|$ such that for every other such pair $\langle \underline{C}, f \rangle$ there exists a unique homomorphism $h : \underline{B} \rightarrow \underline{C}$ satisfying $h \circ \eta = f$. The free Ax -model over any set always exists, and it is unique up to a unique isomorphism preserving η .

Example 8-10. The carrier of the free semilattice over a set X is given by the non-empty finite powerset $\mathcal{P}_+^{X_0}(X)$. The join operation is given by union. \square

Example 8-11. The carrier of the free monoid over a set X is given by the set X^* of finite sequences of X -elements. The monoid unit is the empty sequence, $\langle \rangle$. The monoid multiplication is concatenation, $++$. \square

The following three examples are due to Plotkin and Power [PP02]:

Example 8-12. Let $\mathbb{V} = \{v_1, \dots, v_n\}$ be a finite set with at least two elements denoting storable values. The carrier of the free monoid TX over a set X is given by $(\mathbb{V} \times X)^\mathbb{V}$. The lookup operation is given by:

$$\begin{aligned} \text{lookup} : (TX)^n &\rightarrow TX \\ \text{lookup} : \lambda i. (\lambda v. \langle u_{i,v}, x_{i,v} \rangle) &\mapsto \lambda v_i. \langle u_{i,v_i}, x_{i,v_i} \rangle \end{aligned}$$

For every $v_0 \in \mathbb{V}$, the update_{v_0} operation is given by:

$$\begin{aligned} \text{update}_{v_0} : (TX) &\rightarrow TX \\ \text{update}_{v_0} : k &\mapsto \lambda v. k(v_0) \end{aligned}$$

Compare these operations with the algebraic operations for the global state monad given in Example 2-2. \square



Let Ax be a presentation. Denote by $\mathbf{Mod}(Ax, \mathbf{Set})$ the category consisting of Ax -models as objects and homomorphisms between them. Consider the forgetful functor $U_{Ax} : \mathbf{Mod}(Ax, \mathbf{Set}) \rightarrow \mathbf{Set}$ forgetting the algebra structure. The free model over X is a universal arrow from X to U . The existence of the free Ax -model implies that U_{Ax} always has a left adjoint $F_{Ax} \dashv U_{Ax}$ given on objects by the free algebra.



The entailment relation forms a semantic notion of validity. We now discuss syntactic validity via *provability*. Given a signature σ , a *substitution* θ from a set X to a set Y assigns to each element in X a σ -term over Y . Thus, substitutions from X to Y are valuations over X in the term algebra $\text{Terms}_\sigma(Y)$. Therefore, we may say

AXIOM: $e \in E \implies Ax \vdash e$		
REFL: $Ax \vdash t = t$	SYMM: $\frac{Ax \vdash t = s}{Ax \vdash s = t}$	TRANS: $\frac{Ax \vdash t = s \quad Ax \vdash s = u}{Ax \vdash t = u}$
SUBST: $\frac{Ax \vdash e}{Ax \vdash e\theta} \quad (\theta \text{ suffices for } e)$		
CONG: $\frac{\text{for all } \mathbf{x} \in \text{var}(t): Ax \vdash \theta_1(\mathbf{x}) = \theta_2(\mathbf{x})}{Ax \vdash t\theta_1 = t\theta_2} \quad (\theta_1, \theta_2 \text{ suffice for } t)$		

Figure 8.1: equational logic

that a substitution suffices for a term or an equation. When θ suffices for t , we write $t\theta$ for the term resulting from performing the substitution, and similarly for an equation e . A *renaming* is a substitution to Y assigning only variables in Y .

Given a presentation $Ax = \langle \sigma, E \rangle$, Figure 8.1 inductively defines the provability relation $Ax \vdash t = s$ over σ -equations. The provability relation is thus the least congruence relation, with respect to the operations in σ , that contains E , and that is closed under substitution. When the presentation is *free*, i.e., $E = \emptyset$, the provability relation coincides with syntactic equality.

We extend the provability relation from Terms_σ to terms over any set X by defining $Ax \vdash t = s$ if and only if there exist provably equal σ -terms t', s' (over Var) and a renaming θ to X sufficient for $t' = s'$ such that $t'\theta$ and $s'\theta$ are syntactically equivalent to t and s , respectively. The extended provability relation is then the least congruence over the term algebra $\text{Terms}_\sigma(X)$, with respect to its operations, contains all instances of equations in E , and closed under substitution. We can therefore quotient each term algebra $\text{Terms}_\sigma(X)$ by the provability relation, with the operations factoring through the congruence relation. The resulting algebra FX is then an Ax -model called the *term model over X* .

Theorem 8.3. *Let $Ax = \langle \sigma, E \rangle$ be a presentation, and X is a set. The term model over X , with the function mapping each variable in X to its equivalence class, is the free*

Ax -model over X . Moreover, every σ -equation over X is provable if and only if it is valid in the term model FX .

In particular, entailment and provability coincide:

Theorem 8.4 (soundness and completeness). *For every presentation Ax :*

$$Ax \models e \quad \iff \quad Ax \vdash e$$

Example 8-13. The free semilattice $\mathcal{P}_+^{\mathbf{X}^0}(X)$ can be viewed as the term model where each set $\{x_1, \dots, x_n\}$ corresponds to the equivalence class of $x_1 \vee \dots \vee x_n$. \square

Example 8-14. The free monoid X^* can be viewed as the term model where each finite sequence $\langle x_1, \dots, x_n \rangle$, $n \geq 0$, corresponds to the equivalence class of $x_1 \cdots x_n$. \square

Example 8-15. Let $\mathbb{V} = \{v_1, \dots, v_n\}$, $n \geq 2$ be a finite set denoting storable values. The free mnemoid $(\mathbb{V} \times X)^{\mathbb{V}}$ can be viewed as the term model where each function $\lambda v. \langle u_v, x_v \rangle$ corresponds to the equivalence class of the following term:

$$\text{lookup}(\text{update}_{v_1}(x_{v_1}), \dots, \text{update}_{v_n}(x_{v_n})) \quad \square$$

We say that a presentation Ax is *equationally inconsistent* if $Ax \vdash \mathbf{x} = \mathbf{y}$. A presentation that is not inconsistent is consistent. Theorem 8.4 implies that if Ax is inconsistent, every Ax -model has at most one element. In this case, if Ax has no constant symbols, the free model over the empty set is the empty set. If Ax does contain at least one constant symbol, the free model over the empty set is also a singleton.

Finally, we discuss morphisms between presentations. The following notions are *not* standard. Let σ, σ' be signatures. A *translation of signatures* $\mathfrak{T} : \sigma \rightarrow \sigma'$ is an assignment assigning to every $f : n$ in σ a σ' -term $\mathfrak{T}(f)$ over $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. Note that every translation \mathfrak{T} extends uniquely to a homomorphism $\mathfrak{T} : \text{Terms}_\sigma \rightarrow \text{Terms}_{\sigma'}$ between the term algebras of the corresponding signatures (without any equations).

Let $Ax = \langle \sigma, E \rangle$, $Ax' = \langle \sigma', E' \rangle$ be presentations. A *translation of presentations* $\mathfrak{T} : Ax \rightarrow Ax'$ is a translation $\mathfrak{T} : \sigma \rightarrow \sigma'$ such that, for every $t = s$ in E ,

$$Ax' \vdash \mathfrak{T}(t) = \mathfrak{T}(s)$$

The category of presentations **Presentation** is given by presentations and translations between them. The identities are the translations mapping each $f : n$ to

$$\mathfrak{T}(f) := f(\mathbf{x}_1, \dots, \mathbf{x}_n)$$

Composition $\mathfrak{T}_2 \circ \mathfrak{T}_1$ is given by composing the homomorphic extension of \mathfrak{T}_2 with \mathfrak{T}_1 .

Example 8-16. Let Ax_{Ab} be the presentation of Abelian groups whose signature σ_{Ab} consists of a binary operation $+$: 2, a constant 0 : 0, a unary operation $-$: 1, and whose equations are

$$\begin{array}{ll} \mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x} & \mathbf{x} + (-\mathbf{x}) = 0 \\ (\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z}) & \mathbf{x} + 0 = \mathbf{x} \end{array}$$

Let Ax_{Neg} be the presentation whose signature σ_{Neg} consists of a binary operation $-$: 2, a constant 0 : 0, and whose equations are

$$\mathbf{x} - (\mathbf{y} - (\mathbf{z} - (\mathbf{x} - \mathbf{y}))) = \mathbf{z} \qquad \mathbf{x} - \mathbf{x} = 0$$

Define the following translation $\mathfrak{T} : \sigma_{Neg} \rightarrow \sigma_{Ab}$:

$$\mathfrak{T}(0) := 0 \qquad \mathfrak{T}(-) := \mathbf{x}_1 + (-\mathbf{x}_2)$$

Then \mathfrak{T} is also a translation $\mathfrak{T} : Ax_{Neg} \rightarrow Ax_{Ab}$.

Define the following translation $\mathfrak{T}^{-1} : \sigma_{Ab} \rightarrow \sigma_{Neg}$:

$$\mathfrak{T}^{-1}(0) := 0 \qquad \mathfrak{T}^{-1}(+) := \mathbf{x}_1 - (0 - \mathbf{x}_2) \qquad \mathfrak{T}^{-1}(-) := 0 - \mathbf{x}_1$$

Tarski [Tar38] showed that the Abelian group axioms follow from the two axioms of σ_{Neg} , and therefore $\mathfrak{T}^{-1} : Ax_{Ab} \rightarrow Ax_{Neg}$ is a translation.

The two compositions are given by:

$$\begin{aligned} \mathfrak{T}^{-1} \circ \mathfrak{T}(0) &= 0 & \mathfrak{T}^{-1} \circ \mathfrak{T}(-) &= \mathfrak{T}^{-1}(\mathbf{x}_1 + (-\mathbf{x}_2)) \\ & & &= \mathbf{x}_1 - (0 - (0 - \mathbf{x}_2)) \end{aligned}$$

$$\begin{aligned} \mathfrak{T} \circ \mathfrak{T}^{-1}(0) &= 0 & \mathfrak{T} \circ \mathfrak{T}^{-1}(+) &= \mathfrak{T}(\mathbf{x}_1 - (0 - \mathbf{x}_2)) \\ & & &= \mathbf{x}_1 + (-(0 + (0 + (-\mathbf{x}_2)))) \end{aligned}$$

$$\mathfrak{T} \circ \mathfrak{T}^{-1}(-) = \mathfrak{T}(0 - \mathbf{x}_1) = 0 + (0 - \mathbf{x}_1)$$

Note that $\mathfrak{T}^{-1} \circ \mathfrak{T}(0)$ and $\text{id}(0)$ are syntactically equivalent, but $\mathfrak{T}^{-1} \circ \mathfrak{T}(-)$ and $\text{id}(-)$ are not, and therefore $\mathfrak{T}^{-1} \circ \mathfrak{T}$ is *not* the identity translation. However, the following does hold:

$$Ax_{Neg} \vdash \mathfrak{T}^{-1} \circ \mathfrak{T}(-) = \text{id}(-) \qquad \square$$

The previous example suggests an alternative for translations. Given two presentations Ax, Ax' , we define the relation \sim over translations $\mathfrak{T}_1, \mathfrak{T}_2 : Ax \rightarrow Ax'$ by $\mathfrak{T}_1 \sim \mathfrak{T}_2$ if and only if for every $f : n$ in Ax :

$$Ax' \vdash \mathfrak{T}_1(f) = \mathfrak{T}_2(f)$$

For example, the translations from Example 8-16 satisfy $\mathfrak{T}^{-1} \circ \mathfrak{T} \sim \text{id}$ and $\mathfrak{T} \circ \mathfrak{T}^{-1} \sim \text{id}$. Therefore, it is natural to consider the category **Theory** consisting of presentations together with equivalence classes of translations as morphisms between them. We will do so in the next section using Lawvere theories and monads. The advantage of our chosen notion of translations is that their equality can be decided syntactically.

We will be mainly concerned with a particular kind of translation. An *extension* is a translation $\mathfrak{T} : Ax \rightarrow Ax'$ such that for every $f : n$ in Ax there is a unique $f' : n$ in Ax' such that $\mathfrak{T}(f) = f'(\mathbf{x}_1, \dots, \mathbf{x}_n)$ syntactically, and, moreover, if $f'_1 = f'_2$ then $f_1 = f_2$.

Example 8-17. Let $\mathbb{V} = \{v_1, \dots, v_n\}$ be a finite set with at least two elements denoting storable values. The *environment presentation* is given by the two equations:

$$\begin{aligned} \text{lookup}(\mathbf{x}, \dots, \mathbf{x}) &= \mathbf{x} \\ \text{lookup}(\text{lookup}(\mathbf{x}_1^1, \dots, \mathbf{x}_n^1), \dots, \text{lookup}(\mathbf{x}_1^n, \dots, \mathbf{x}_n^n)) &= \text{lookup}(\mathbf{x}_1^1, \dots, \mathbf{x}_n^n) \end{aligned}$$

Then, by Example 8-15, we have an extension from the environment presentation to the monoid presentation mapping `lookup` to `lookup`. \square

As in the previous example, the extension will usually be obvious. In this case we simply say that Ax' is an extension of Ax .

8.2 Universal algebra and monads

We review the well-known connections between presentations, monads and Lawvere theories. Let Ax be a presentation. Let $F : \mathbf{Set} \rightarrow \mathbf{Mod}(Ax, \mathbf{Set})$ be the free Ax -model functor, and $U : \mathbf{Mod}(Ax, \mathbf{Set}) \rightarrow \mathbf{Set}$ be the forgetful functor assigning to each model its carrier set. Then $T := UF$ is a monad. Its unit $\eta : X \rightarrow TX$ is the function assigning to each variable $x \in X$ its equivalence class $[x] \in TX$. The monadic multiplication $\mu : T^2X \rightarrow TX$ is defined inductively as follows:

- For every $[t] \in FX$, $\mu[t] := [t]$
- For every $\text{op} : n$, $\tau_1, \dots, \tau_n \in FTX$ such that $\mu(\tau_1) = [t_1], \dots, \mu(\tau_n) = [t_n]$, define: $\mu[\text{op}(\tau_1, \dots, \tau_n)] := [\text{op}(t_1, \dots, t_n)]$.

Each $\text{op} : n$ in Ax yields an *algebraic operation* $\text{op} : n$ (cf. Definition 2.1) given by

$$\begin{aligned} \text{op} : (TX)^n &\rightarrow (TX)^1 \\ \lambda \mathbf{i}^*. [t_i] &\mapsto [\text{op}(t_1, \dots, t_n)] \end{aligned}$$

The corresponding generic effect $\text{gen} : \mathbb{1}$ is given by:

$$\begin{aligned} \text{gen} : \mathbb{1} &\rightarrow T\mathbb{1} \\ \star &\mapsto [\text{op}(\mathbf{1}_1\star, \dots, \mathbf{1}_n\star)] \end{aligned}$$

More generally, each P -indexed family of terms $\langle s_p \rangle_{p \in P}$ over n variables yields an algebraic operation of type $\mathbb{n} \langle P \rangle$:

$$\begin{aligned} \text{op} : (TX)^A &\rightarrow (TX)^P \\ \lambda \mathbf{1}_i\star.[t_i] &\mapsto \lambda p.[s_p(t_1, \dots, t_n)] \end{aligned}$$

The corresponding generic effect $\text{gen} : \mathbb{n} \langle P \rangle$ is given by:

$$p \mapsto [s_p(\mathbf{1}_1\star, \dots, \mathbf{1}_n\star)]$$

Example 8-18. The monads corresponding to the two inconsistent theories are the constantly $\mathbb{1}$ monad, when the theory contains a constant, and the monad mapping the empty set to itself and every non-empty set to $\mathbb{1}$, when the theory contains no constants. \square

Example 8-19. Let $\mathbb{V} = \{v_1, \dots, v_n\}$ be a finite set with at least two elements denoting storable values. The global state monad $T_{\text{GS}(\mathbb{V})}$ is the free mnemoid monad, the look-up operation for this monad is the operation corresponding to the look-up mnemoid operation. For the update operation, note that for each $v \in \mathbb{V}$, applying the generic effect for update at v , i.e. $\text{set}!(v)$ yields the same result as applying the generic effect for update_v to \star .

Similarly, the environment monad $T_{\text{Env}(\mathbb{V})}$ is the free model monad for the environment presentation from Example 8-17. Every function $f \in X^{\mathbb{V}}$ corresponds to the equivalence class of the term

$$\text{lookup}(f(v_1), \dots, f(v_n))$$

The look-up operation of the environment monad is the operation corresponding to lookup.

Finally, the overwrite monad $T_{\text{OW}(\mathbb{V})}$ is the free model monad for the following *overwrite presentation*: the signature is given by $\{\text{update}_v : \mathbb{1} \mid v \in \mathbb{V}\}$, and the equations are given by

$$\text{update}_v(\text{update}_{v'}(\mathbf{x})) = \text{update}_{v'}(\mathbf{x})$$

for every $v, v' \in \mathbb{V}$. In this case, for every $v \in \mathbb{V}$, applying the generic effect for update at v , i.e. $\text{set}!(v)$ for update_v to \star . \square

Example 8-20. Recall the model for exceptions and terminal I/O from Example 3-5. Let $\text{Char} = \{c_1, \dots, c_n\}$ be a set denoting terminal characters. The monad from Example 3-5 is the free model for the free presentation over the signature

$$\{\text{input} : n, \text{output}_c : 1, \text{raise}_s : 0 \mid c \in \text{Char}, s \in \text{Str}\}$$

As a free presentation, the term model coincides with the term algebra. Thus the different constructors used in Example 3-5 correspond to the abstract syntax tree constructors of the term model. \square

More generally, the free model monad for any free presentation coincides with the term algebra for that signature.

Let $\mathfrak{T} : \text{Ax} \rightarrow \text{Ax}'$ be a translation. Let T, T' be the monads corresponding to Ax, Ax' , respectively. We then have a family of functions:

$$\begin{aligned} m_X : TX &\rightarrow T'X \\ [t] &\mapsto [\mathfrak{T}(t)] \end{aligned}$$

Then m is a monad morphism from T to T' . Moreover, for every P -indexed family of n -ary terms $\langle t_p \rangle_{p \in P}$, this monad morphism m maps the generic effect corresponding to $\langle t_p \rangle$ to the generic effect corresponding to $\langle \mathfrak{T}(t_p) \rangle$.



The monad corresponding to a presentation is finitary. The assignments $\text{Ax} \mapsto T_{\text{Ax}}$ and $\mathfrak{T} \mapsto T_m$ form a functor $T_- : \mathbf{Presentation} \rightarrow \mathbf{Set-Monads}_{\aleph_0}$.

We recall the well-known connection between presentations and finitary monads in the following theorem:

Theorem 8.5. *The functor $T_- : \mathbf{Presentation} \rightarrow \mathbf{Set-Monads}_{\aleph_0}$ is essentially surjective and full. Moreover, T_- factorises as*

$$T_- : \mathbf{Presentation} \xrightarrow{Q} \mathbf{Theory} \simeq \mathbf{Set-Monads}_{\aleph_0}$$

where Q is the evident identity-on-objects full functor mapping each translation to its \sim -equivalence class. Therefore, for all $\mathfrak{T}, \mathfrak{T}' : \text{Ax} \rightarrow \text{Ax}'$, we have $T_{\mathfrak{T}} = T_{\mathfrak{T}'}$ if and only if $\mathfrak{T} \sim \mathfrak{T}'$.

Crucially, T_- is not faithful. For example, the two presentations of Abelian groups from Example 8-16 yield the free Abelian group monad, and all translations in that example are mapped to the identity monad morphism. However, the action of T_- on translations is injective on representatives of *distinct* equivalence classes of the

relation \sim over translations. This result captures the notion that finitary monads are presentation invariants of universal algebra.

We now turn to Lawvere theories. We can post-compose the functor T_- with the isomorphism $\mathbf{Set}\text{-Monads}_{\mathbb{N}_0} \cong \mathbf{Law}_{\mathbb{N}_0}\mathbf{Set}$ between finitary monads and Lawvere theories. Therefore, every Lawvere theory can be understood in terms of presentations. Lawvere's thesis [Law63] introduced Lawvere theories to give a presentation invariant account of universal algebra. The following constructions and observations are due to him.

Let Ax be a presentation, then the Lawvere theory \mathcal{L} corresponding to Ax is given as follows. Recall that our notion of a (finitary) Lawvere (set-enriched) theory includes *all* finite sets as objects, and not only the natural numbers. Given two finite sets X, Y , the hom-set $|\mathcal{L}|(X, Y)$ is the set of Y -indexed tuples of Ax -equivalence classes of terms over X . These can be viewed as substitutions whose domain is Y , but the terms are quotiented modulo Ax . The identity morphism in $|\mathcal{L}|(X, X)$ maps each $x \in X$ to the equivalence class of the variable $[x]$. Composition $g \circ f$ corresponds to substituting in g according to f . The category $\mathbf{Pres}_{\mathbb{N}_0}^{\text{op}}\mathbf{Set}$, i.e., the opposite category to finite sets, is then considered as renamings. A function $f : X \leftarrow Y$ assigns to each variable in Y its renamed variable. The functor $\mathcal{L}[-] : \mathbf{Pres}_{\mathbb{N}_0}^{\text{op}}\mathbf{Set} \rightarrow |\mathcal{L}|$ maps each renaming to its counterpart in $|\mathcal{L}|$. Given a translation $\mathfrak{T} : Ax \rightarrow Ax'$, the corresponding morphism of Lawvere theories $\mathfrak{T} : \mathcal{L} \rightarrow \mathcal{L}'$ applies the homomorphic extension of \mathfrak{T} to each equivalence class component of the substitution it acts on. Theorem 8.5 implies that all Lawvere theories and their morphisms arise from presentations and their translations.

The correspondence extends to models:

Theorem 8.6. *Let Ax be a presentation, \mathcal{L} the corresponding Lawvere theory, and T the corresponding finitary monad. Then $\mathbf{Mod}(Ax, \mathbf{Set}) \cong \mathbf{Mod}(\mathcal{L}, \mathbf{Set}) \cong \mathbf{Set}^T$.*

This fact explains our choice of notation for T -algebras: if \underline{B} is a T -algebra, then the elements of $T|\underline{B}|$ are represented by Ax -terms over $|\underline{B}|$. The algebra $\underline{B}[-]$ then homomorphically interprets this term in $|\underline{B}|$.

8.3 Conservative and surjective translations



We focus on the following important class of translations:

Definition 8.7. *A translation $\mathfrak{T} : Ax \rightarrow Ax'$ is conservative when, for every*

pair of Ax -terms, t, s , we have:

$$Ax' \vdash \mathfrak{T}(t) = \mathfrak{T}(s) \quad \Longrightarrow \quad Ax \vdash t = s$$

We denote conservative translations by $\mathfrak{T} : Ax \rightarrow Ax'$. In particular, a *conservative extension* is an extension of Ax to Ax' that satisfies, for all Ax -terms t, s :

$$Ax' \vdash t = s \quad \Longrightarrow \quad Ax \vdash t = s$$

Thus, a presentation is consistent if and only if the unique translation from the empty presentation is conservative.

Theorem 8.8. *A translation $\mathfrak{T} : Ax \rightarrow Ax'$ is conservative if and only if the corresponding monad morphism is component-wise injective.*

Proof



Let $m : T \rightarrow T'$ be the corresponding monad morphism. In light of Proposition 7.8, it suffices to show that \mathfrak{T} is conservative if and only if m_X is injective for all finite sets X . As every element in TX is of the form $[t]$ for some Ax -term t , and as $m([t]) = [\mathfrak{T}(t)]$, we have:

$$\begin{aligned} m([t]) = m([s]) & \iff Ax' \vdash \mathfrak{T}(t) = \mathfrak{T}(s) \\ [t] = [s] & \iff Ax \vdash t = s \end{aligned}$$

and we are done. ■

We turn to the dual notion:

Definition 8.9. *A translation $\mathfrak{T} : Ax \rightarrow Ax'$ is surjective when, for every Ax' -term s , there exists an Ax -term t such that $Ax' \vdash \mathfrak{T}(t) = s$.*

We denote surjective translations by $\mathfrak{T} : Ax \rightarrow Ax'$. Surjective translations relate to component-wise surjective monad morphisms as conservative translations relate to component-wise injective morphisms:

Theorem 8.10. *A translation $\mathfrak{T} : Ax \rightarrow Ax'$ is surjective if and only if the corresponding monad morphism is component-wise surjective.*

Proof

Because the class of surjections is closed under colimits, by Proposition 7.8, it suffices to show that \mathfrak{T} is surjective if and only if m_X is surjective for all finite sets X . The remainder of the proof follows as in Theorem 8.8. ■

The classes of surjective and conservative translations are fundamental building blocks in the category of presentations:

Theorem 8.11. *Every translation admits a surjective-conservative factorisation.*

Explicitly, let $\mathfrak{T} : \mathbf{Ax}_1 \rightarrow \mathbf{Ax}_2$ be any translation. Let \mathbf{Ax} be the presentation consisting of \mathbf{Ax}_1 's signature and all the equations $t = s$ where t, s are \mathbf{Ax}_1 -terms such that $\mathbf{Ax}_2 \vdash \mathfrak{T}(t) = \mathfrak{T}(s)$. Let $\mathfrak{E} : \mathbf{Ax}_1 \rightarrow \mathbf{Ax}$ be the evident extension, and $\mathfrak{M} : \mathbf{Ax} \rightarrow \mathbf{Ax}_2$ be given by $\mathfrak{M}(\text{op}) = \mathfrak{T}(\text{op})$. Then $\mathfrak{T} : \mathbf{Ax}_1 \xrightarrow{\mathfrak{E}} \mathbf{Ax} \xrightarrow{\mathfrak{M}} \mathbf{Ax}_2$ is a factorisation of \mathfrak{T} .

Proof

We need to show \mathfrak{E} and \mathfrak{M} are well-defined surjective and conservative translations, respectively. The fact that they factorise \mathfrak{T} is immediate.

Given any equation $\mathbf{Ax}_1 \vdash t = s$, then

$$\mathbf{Ax}_2 \vdash \mathfrak{T}(\mathfrak{E}(t)) = \mathfrak{T}(t) = \mathfrak{T}(s) = \mathfrak{T}(\mathfrak{E}(s))$$

Therefore, $\mathbf{Ax} \vdash \mathfrak{E}(t) = \mathfrak{E}(s)$. Thus, $\mathfrak{E} : \mathbf{Ax}_1 \rightarrow \mathbf{Ax}$ is a well-defined translation. Because the term algebras for \mathbf{Ax}_1 and \mathbf{Ax}' coincide, this translation is surjective.

Given any equation $\mathbf{Ax} \vdash t = s$, then, by definition:

$$\mathbf{Ax}_2 \vdash \mathfrak{M}(t) = \mathfrak{T}(t) = \mathfrak{T}(s) = \mathfrak{M}(s)$$

Therefore $\mathfrak{M} : \mathbf{Ax} \rightarrow \mathbf{Ax}_2$ is a well-defined translation. If $\mathbf{Ax}_2 \vdash \mathfrak{M}(t) = \mathfrak{M}(s)$, then, by definition, $\mathbf{Ax} \vdash t = s$, hence \mathfrak{M} is conservative. ■



Note that this factorisation may not be unique, as different translations may be equivalent under the \sim relation. However, when we move to Lawvere theories and finitary monads, this factorisation gives rise to a factorisation system.

Corollary 8.12. *The \mathcal{E} -class of the factorisation system of finitary monads over **Set** from Theorem 7.9 arising from the surjective-injective factorisation of **Set** consists of all component-wise surjective monad morphisms.*

*Equivalently, the \mathcal{E} -class of the factorisation system of Lawvere theories from Theorem 7.5 arising from the surjective-injective factorisation of **Set** consists of all morphisms \mathfrak{E} whose components $\mathfrak{E}_{X, \mathbb{1}}$ are surjective.*

Proof

In light of Proposition 7.8, both parts of the theorem are equivalent. Let $e : T_1 \rightarrow T_2$ be any monad morphism in the \mathcal{E} -class for the factorisation system of finitary monads over **Set**.

By Theorem 8.5, there exists a translation of presentations $\mathfrak{E} : Ax_1 \rightarrow Ax_2$ such that the corresponding monad morphism is isomorphic to e . By Theorem 8.11, \mathfrak{E} has a surjective-conservative factorisation $\mathfrak{E} : Ax_1 \xrightarrow{\mathfrak{e}'} Ax' \xrightarrow{\mathfrak{m}'} Ax_2$. Therefore, by Theorems 8.8 and 8.10, e has a component-wise surjective-injective factorisation

$$e : T_1 \xrightarrow{e'} T' \xrightarrow{m'} T_2$$

But, by Theorem 7.9, this factorisation is a factorisation in the factorisation system from Theorem 7.9. As $e : T_1 \xrightarrow{e} T_2 \cong T_2$ is another such factorisation, we deduce that $e \cong e'$, and e is component-wise surjective. ■



A direct consequence of the previous corollary and Theorem 8.5 is that the surjective-conservative factorisation of Theorem 8.11 is essentially unique in the following sense:

Corollary 8.13. *If $\mathfrak{T} : Ax_1 \xrightarrow{\mathfrak{e}} Ax \xrightarrow{\mathfrak{m}} Ax_2$ and $\mathfrak{T}' : Ax_1 \xrightarrow{\mathfrak{e}'} Ax' \xrightarrow{\mathfrak{m}'} Ax_2$ are two surjective-conservative factorisations such that $\mathfrak{T} \sim \mathfrak{T}'$, then there exist two translations $Ax \xrightarrow{\mathfrak{J}} Ax' \xrightarrow{\mathfrak{J}^{-1}} Ax$ such that $\mathfrak{J} \circ \mathfrak{J}^{-1} \sim \text{id}$, $\mathfrak{J}^{-1} \circ \mathfrak{J} \sim \text{id}$, and*

$$\begin{array}{ccccc}
 & & Ax & & \\
 & \mathfrak{e} \nearrow & \downarrow \mathfrak{J} & \nwarrow \mathfrak{m} & \\
 Ax_1 & & & & Ax_2 \\
 & \mathfrak{e}' \searrow & \downarrow \mathfrak{J}' & \swarrow \mathfrak{m}' & \\
 & & Ax' & &
 \end{array}$$

Moreover, if $\mathfrak{J}' : Ax \rightarrow Ax'$ is any other such translation, then $\mathfrak{J}' \sim \mathfrak{J}$.

8.4 Presentation models

We now utilise the algebraic descriptions to account for our hierarchical models.

Definition 7.1 (revisited). *Let Π be a set. A presentation type assignment for Π is a set-theoretic type assignment (see Definition 2.10 (revisited)) such that, for all $op : A \langle P \rangle$, the arity type A is a finite set.*

For example, if \mathbb{V} is a finite set denoting storable values, then the global state type assignment from Example 2-9 (revisited) is a presentation type assignment.

Definition 3.2* (revisited)⁺. Let Σ be a hierarchy. A **presentation Σ -model** is a quadruple

$$\langle \text{type}, \text{Ax}_-, \mathfrak{L}_-, \mathcal{T}_- \llbracket - \rrbracket \rangle$$

where:

- *type* is a **presentation type assignment** for Σ ;
- Ax_- assigns to each ε in \mathcal{E} a **presentation Ax_ε** ;
- \mathfrak{L}_- assigns to each $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} a **translation $\mathfrak{L}_{\varepsilon \subseteq \varepsilon'} : \text{Ax}_\varepsilon \rightarrow \text{Ax}_{\varepsilon'}$** ;
- $\mathcal{T}_- \llbracket - \rrbracket$ assigns to each $\varepsilon \in \mathcal{E}$, and $\text{op} : A \langle P \rangle$ in ε a **P -indexed family $\mathcal{T}_\varepsilon \llbracket \text{op} \rrbracket$ of terms over A** ;
- for all $\varepsilon \subseteq \varepsilon' \subseteq \varepsilon''$ in \mathcal{E} :

$$\mathfrak{L}_{\varepsilon \subseteq \varepsilon'} \sim \text{id}, \quad \mathfrak{L}_{\varepsilon' \subseteq \varepsilon''} \circ \mathfrak{L}_{\varepsilon \subseteq \varepsilon'} \sim \mathfrak{L}_{\varepsilon \subseteq \varepsilon''}$$

i.e., \mathfrak{L}_- is functorial;

and, for all $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} , $\text{op} : A \langle P \rangle$ in ε , and $p \in P$:

$$\text{Ax}_{\varepsilon'} \vdash \mathfrak{L}_{\varepsilon \subseteq \varepsilon'}(\mathcal{T}_\varepsilon \llbracket \text{op} \rrbracket(p)) = \mathcal{T}_{\varepsilon'} \llbracket \text{op} \rrbracket(p)$$

Thus, \mathfrak{L}_- preserves the **effect terms**.

Let $\langle \text{type}, \text{Ax}_-, \mathfrak{L}_-, \mathcal{T}_- \llbracket - \rrbracket \rangle$ be a presentation Σ -model. We construct a set-theoretic Σ -model $\langle \text{type}, T_-, m_-, \mathcal{G}_- \llbracket - \rrbracket \rangle$ as follows. For every ε in \mathcal{E} , T_ε is the free Ax_ε -model monad. For every $\varepsilon \subseteq \varepsilon'$, $m_{\varepsilon \subseteq \varepsilon'}$ is the monad morphism corresponding to the translation $\mathfrak{L}_{\varepsilon \subseteq \varepsilon'}$. Finally, for every ε in \mathcal{E} and $\text{op} : A \langle P \rangle$ in ε , we set

$$\begin{aligned} \mathcal{G}_\varepsilon \llbracket \text{op} \rrbracket : P &\rightarrow T_\varepsilon A \\ p &\mapsto [\mathcal{T}_\varepsilon \llbracket \text{op} \rrbracket(p)] \end{aligned}$$



Note that each of the monads T_ε in this construction is finitary. The converse statement also holds, in light of the essential surjectivity of the functor $T_- : \mathbf{Presentation} \rightarrow \mathbf{Set-Monads}_{\aleph_0}$ (Theorem 8.5). Thus, every set-theoretic Σ -model with a presentation type assignment arises from a presentation Σ -model in this way.



Our goal is to generate hierarchical models from given algebraic descriptions of the non-hierarchical models. We therefore first need a notion of such models.

Definition 2.13⁺ (revisited). Let Π be a set. A presentation Π -model is a triple

$$\langle \text{type}, \text{Ax}, \mathcal{T}[-] \rangle$$

where:

- *type* is a presentation type assignment for Π ;
- *Ax* is a presentation; and
- $\mathcal{T}[-]$ assigns to each $\text{op} : A \langle P \rangle$ in Π a P -indexed family $\mathcal{T}[\text{op}]$ of terms over A .

Example 8-21. Take $\Pi := \{\text{lookup}, \text{update}\}$. Let $\mathbb{V} = \{v_1, \dots, v_n\}$, $n \geq 2$ be any finite set denoting storable values. Let *type* be the presentation type assignment from Example 6-13. Let *Ax* be the presentation of \mathbb{V} -mnemoids from Example 8-6. Define:

$$\mathcal{T}[\text{lookup}] : \star \mapsto \text{lookup}(v_1, \dots, v_n) \quad \mathcal{T}[\text{update}] : v_0 \mapsto \text{update}_{v_0}(\star)$$

Then $\langle \text{type}, \text{Ax}, \mathcal{T}[-] \rangle$ is a presentation Π -model for global \mathbb{V} -state. \square

Example 8-22. Take $\Pi := \{\text{choose}\}$. Let *type* be the presentation type assignment given by $\text{choose} : 2$, *Ax* be the semilattice presentation from Example 8-4, and set

$$\mathcal{T}[\text{choose}] : \star \mapsto (\mathbf{t}_1 \star) \vee (\mathbf{t}_2 \star)$$

Then $\langle \text{type}, \text{Ax}, \mathcal{T}[-] \rangle$ is a presentation Π -model for non-deterministic choice. \square

Example 8-23. Take $\Pi := \{\text{input}, \text{output}, \text{raise}\}$. Let $\text{Char} = \{c_1, \dots, c_n\}$ be a finite set denoting terminal characters. Let *type* be the presentation type assignment given by

$$\text{input} : \text{Char} \quad \text{output} : \mathbb{1} \langle \text{Char} \rangle \quad \text{raise} : \mathbb{0} \langle \text{Str} \rangle$$

Let *Ax* be the free presentation over the signature:

$$\{\text{input} : n, \text{output}_c : 1, \text{raise}_s : 0 \mid c \in \text{Char}, s \in \text{Str}\}$$

Set

$$\begin{aligned} \mathcal{T}[\text{input}] & : \star \mapsto \text{input}(c_1, \dots, c_n) \\ \mathcal{T}[\text{output}] & : c \mapsto \text{output}_c(\star) \\ \mathcal{T}[\text{raise}] & : s \mapsto \text{raise}_s \end{aligned}$$

Then $\langle \text{type}, \text{Ax}, \mathcal{T}[-] \rangle$ is a presentation Π -model for terminal I/O and exceptions. \square

Note that these examples illustrate that our notion of presentation Π -models is lightweight: the only proof obligations it incurs are the finiteness of the arity types, and the well-formedness of the equations and effect terms.



The Π -model in Example 8-23 does *not* arise from an algebraic Π -model with a finitary Lawvere theory, as the type assignment is not finitely presentable. We can capture this model by switching to countable Lawvere theories, but these capture monads that cannot be presented using our finitary notion of presentations. Therefore, there is a mismatch between our notion of a presentation Π -model and our notion of a finitary algebraic Π -model. As we do not wish to diverge from the standard notions of universal algebra and of enriched Lawvere theories, we do not resolve this mismatch in this thesis.



We now present the main construction of this chapter, the conservative restriction model:

Theorem 7.12 (revisited). *Let Π be a set, and let $\mathcal{M} = \langle \text{type}, \text{Ax}, \mathcal{T}[-] \rangle$ be a presentation Π -model. For every effect hierarchy Σ whose set of operations is Π , we have a Σ -model defined by*

$$\mathcal{M}_{\sharp} := \langle \text{type}, \text{Ax}_{-}, \mathcal{L}_{-}, \mathcal{T}_{-}[-] \rangle$$

together with the auxiliary data $\langle \text{enum}_{-}, \sigma_{-}, \mathcal{T}_{-}, \mathcal{E}_{-}, \mathfrak{M}_{-}, \mathcal{G}_{-} \rangle$, where:

- enum_{-} assigns to each $\text{op} : A \langle P \rangle$, $|A| = n$, in Π a bijection $\text{enum}_{\text{op}} : A \cong \{1, \dots, n\}$;
- σ_{-} assigns to every $\varepsilon \in \mathcal{E}$ a signature σ_{ε} given by

$$\sigma_{\varepsilon} := \{ \text{op}_p : n \mid \text{op} : A \langle P \rangle, |A| = n, p \in P \}$$

- \mathcal{T}_{-} assigns to every $\varepsilon \in \mathcal{E}$ the translation $\mathcal{T}_{\varepsilon} : \langle \sigma_{\varepsilon}, \theta \rangle \rightarrow \text{Ax}$ given by the substitution

$$\mathcal{T}_{\varepsilon}(\text{op}) := \mathcal{T}[\text{op}] \theta$$

where θ is the substitution given by $\theta(a) := \mathbf{x}_{\text{enum}_{\text{op}} a}$ for every $a \in A$;

- for every $\varepsilon \in \mathcal{E}$, $\langle \text{Ax}_{\varepsilon}, \mathcal{E}_{\varepsilon}, \mathfrak{M}_{\varepsilon} \rangle$ is the factorisation from Theorem 8.11,

$$\mathcal{T}_{\varepsilon} : \langle \sigma_{\varepsilon}, \theta \rangle \xrightarrow{\mathcal{E}_{\varepsilon}} \text{Ax}_{\varepsilon} \xrightarrow{\mathfrak{M}_{\varepsilon}} \text{Ax}$$

i.e., Ax_{ε} has σ_{ε} as signature, $\mathcal{E}_{\varepsilon}$ is the evident surjective extension, and $\mathfrak{M}_{\varepsilon}$ is the evident conservative translation;

- $\mathcal{T}_- \llbracket - \rrbracket$ assigns to every $\varepsilon \in \mathcal{E}$ and $\text{op} : A \langle P \rangle$, $|A| = n$ in ε the family

$$\mathcal{T}_\varepsilon \llbracket \text{op} \rrbracket := \langle \text{op}_p(\text{enum}_{\text{op}}^{-1} 1, \dots, \text{enum}_{\text{op}}^{-1} n) \rangle_{p \in P}$$

- \mathcal{S}_- assigns to every $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} the evident extension $\mathcal{S}_{\varepsilon \subseteq \varepsilon'} : \langle \sigma_\varepsilon, \emptyset \rangle \rightarrow \text{Ax}_{\varepsilon'}$; and
- \mathcal{L}_- assigns to every $\varepsilon \subseteq \varepsilon'$ the evident extension $\mathcal{L}_{\varepsilon \subseteq \varepsilon'} : \text{Ax}_\varepsilon \rightarrow \text{Ax}_{\varepsilon'}$, and, moreover, this extension is conservative.

We call $\mathcal{M}_\#$ the set-theoretic conservative restriction Σ -model for the given Π -model.

Note that this theorem gives us an explicit and uniform definition of $\mathcal{M}_\#$. However, working out the exact syntactic description of each presentation Ax_ε is impractical, as these presentations always have infinitely many axioms. In every concrete case, we will find alternative and simpler ways to present the monad corresponding to Ax_ε . Our technique is to rely on Corollary 8.13: we will factorise the translation \mathfrak{T}_ε to a surjective-conservative factorisation. The factorising presentation is then, up to the equivalence \sim , canonically isomorphic to Ax_ε .

Example 7-11 (revisited). Let $\Pi := \{\text{lookup}, \text{update}\}$. Let $\mathbb{V} = \{v_1, \dots, v_n\}$, $n \geq 2$, be a finite set denoting storable values. Let $\mathcal{M} := \langle \text{type}, \text{Ax}, \mathcal{T} \llbracket - \rrbracket \rangle$ be the global state Π -model from Example 8-21. Let Σ be the full powerset hierarchy. We analyse the conservative restriction model $\mathcal{M}_\#$.

The presentation Ax_\emptyset can be alternatively described by the empty presentation. Indeed, because the theory of mnemoids is consistent, we have a factorisation consisting of extensions:

$$\langle \sigma_\emptyset, \emptyset \rangle = \langle \emptyset, \emptyset \rangle \twoheadrightarrow \langle \emptyset, \emptyset \rangle \twoheadrightarrow \text{Ax}$$

Therefore, by Corollary 8.13, Ax_\emptyset and the empty presentation both present the same monad.

The presentation $\text{Ax}_{\{\text{lookup}\}}$ can be alternatively described by the environment presentation $\text{Ax}_{\text{Env}(\mathbb{V})}$ from Example 8-17, as the two evident extensions form a factorisation

$$\mathfrak{T}_{\{\text{lookup}\}} : \langle \{\text{lookup}_* : n\}, \emptyset \rangle \twoheadrightarrow \text{Ax}_{\text{Env}(\mathbb{V})} \twoheadrightarrow \text{Ax}$$

Indeed, as $\text{Ax}_{\text{Env}(\mathbb{V})}$'s signature is $\{\text{lookup} : n\}$, the first translation is surjective. The monad morphism corresponding to the second translation is the monad morphism $m_{\{\text{lookup}\} \subseteq \{\text{lookup}, \text{update}\}}$ from Example 3-2* (revisited). Direct calculation shows this

monad morphism is componentwise injective, hence the second translation is conservative.

Similarly, the presentation $Ax_{\{\text{update}\}}$ can be alternatively described by the over-write presentation $Ax_{\text{OW}(\mathbb{V})}$ from Example 8-19.

Finally, the presentation $Ax_{\{\text{lookup}, \text{update}\}}$ can be alternatively described by the presentation Ax itself.

In summary, the set-theoretic Σ -model corresponding to \mathcal{M}_{\sharp} is the Σ -model for global state from Example 3-2* (revisited). \square

Example 8-24. Let $\Pi := \{\text{input}, \text{output}, \text{raise}\}$ and $\mathcal{M} := \langle \text{type}, Ax, \mathcal{T}[\] \rangle$ be the terminal I/O and exception presentation Π -model from Example 8-23. Let Σ be the full powerset hierarchy. We analyse conservative restriction model \mathcal{M}_{\sharp} .

For every $\varepsilon \subseteq \Pi$, we have the following factorisation of extensions:

$$\mathfrak{T}_{\varepsilon} : \langle \sigma_{\varepsilon}, \emptyset \rangle \twoheadrightarrow \langle \sigma_{\varepsilon}, \emptyset \rangle \twoheadrightarrow Ax$$

Indeed, the first extension is evidently surjective. As Ax is free, the second extension is conservative. Therefore, Ax_{ε} may be alternatively presented by $\langle \sigma_{\varepsilon}, \emptyset \rangle$. Therefore, the set-theoretic model corresponding to \mathcal{M}_{\sharp} is the Σ -model for terminal I/O and exceptions given in Example 3-5. \square

The last two examples show that the conservative restriction model yields the natural Σ -models from Section 3.3. Note how we transitioned from specifying *structure* to specifying *properties*: there is very little overhead in proof obligations while describing the structure of the starting Π -model. A more elaborate analysis of the *properties* of the resulting conservative restriction models recovers the rest of the elaborate model structure. Thus, we avoid the need to specify the exponentially large structure required by set-theoretic Σ -models.

In summary, we reviewed the relevant background in universal algebra, equational logic, and its relationship to finitary monads and Lawvere theories, and reformulated our conservative restriction construction in terms of presentations.

Chapter 9

Relational models

We search our lives forever for perfection in relations

—Simply Red



In this chapter we study a useful subclass of our categorical models in which the objects are suitable *relations*, subsets in the set-theoretic case and ω -chain-closed subsets in the domain-theoretic case. Relational models organise the data required to present *logical relations* proofs [Fil07, Mit90, Rey74] for relating different semantic models.

We leave a general account of relational models to future work. Here, we only consider predicates over sets. However, we structure our account in a form we believe holds in greater generality.

First, in Section 9.1, we introduce set-theoretic logical relations as a category. Next, in Section 9.2, we present our *monadic lifting* construction, which we use in Corollary 9.12 to relate the conservative restriction model of Theorem 7.12 and the benchmark model of Example 7-2.

9.1 Set-theoretic logical relations

We define predicates over sets:

Definition 9.1. *The category $\mathbf{Pred}_{\text{Set}}$ of set-theoretic predicates is given as follows:*

- the **objects** consist of pairs $\langle X, P \rangle$ where X is a set and P is a subset of X ; and
- the **morphisms** from $\langle X, P \rangle$ to $\langle Y, Q \rangle$ consist of pairs $\langle f, \hat{f} \rangle$ where $f : X \rightarrow Y$ is a function and $\hat{f} : P \rightarrow Q$ such that

$$\begin{array}{ccc} P & \xrightarrow{\dot{f}} & Q \\ \text{|\cap} & = & \text{|\cap} \\ X & \xrightarrow{f} & Y \end{array}$$

We will denote objects of **Pred** as $P \subseteq X$, and say that P is a *predicate over X* . Note that if \dot{f} exists, then it is uniquely determined by f , and we say that f *lifts* to a morphism from $P \subseteq X$ to $Q \subseteq Y$. Also note that **Pred** is categorically equivalent to the full subcategory of the arrow category $\mathbf{Set}^{\mathcal{S}}$ induced by the morphisms in the \mathcal{M} -class of the surjection-injection factorisation system of **Set**.

As is well-known:

Proposition 9.2. *The category **Pred** is cartesian closed and has all products and coproducts. As a consequence, it is distributive. This structure is given as follows:*

- **products:** We have, $\prod_{i \in I} (P_i \subseteq X_i) = (\prod_{i \in I} P_i) \subseteq (\prod_{i \in I} X_i)$, and the set-theoretic projections lift to the product of predicates. Given, for all $i \in I$, a morphism $\langle f_i, \dot{f}_i \rangle : Q \subseteq Y \rightarrow P_i \subseteq X_i$, then $\langle \langle f_i, \dot{f}_i \rangle \rangle = \langle \langle f_i \rangle, \langle \dot{f}_i \rangle \rangle$.
- **coproducts:** Similarly, $\sum_{i \in I} (P_i \subseteq X_i) = (\sum_{i \in I} P_i) \subseteq (\sum_{i \in I} X_i)$, and the set-theoretic injections lift to the coproduct of predicates. Given, for all $i \in I$, a morphism $\langle f, \dot{f} \rangle : P_i \subseteq X_i \rightarrow Q \subseteq Y$, then $[\langle f_i, \dot{f}_i \rangle] = \langle [f_i], [\dot{f}_i] \rangle$.
- **exponentials:** $(Q \subseteq Y)^{P \subseteq X} = R \subseteq Y^X$, where R is the subset of all functions from X to Y that lift. Explicitly,

$$R = \{f : X \rightarrow Y \mid \text{for all } p \text{ in } P, f(p) \in Q\}$$

The evaluation map lifts to the evaluation map in **Pred**, and for every predicate morphism

$$\langle f, \dot{f} \rangle : (S \subseteq Z) \times (P \subseteq X) \rightarrow Q \subseteq Y$$

the curried function $\lambda X. f$ lifts to $\lambda P \subseteq X. \langle f, \dot{f} \rangle$.

Proof

Jacobs [Jac01, Section 9.2 and Exercise 9.2.1] includes this structure as an exercise.

Katsumata [Kat13] spells it out in detail. ■

The following category is the central concept of this chapter:

Definition 9.3. *The category **LogRel** of set-theoretic logical relations is given by:*

- the **objects** X consist of triples $\langle X_1, X_2, \dot{X} \rangle$, where X_1 and X_2 are sets and \dot{X} is a predicate over $X_1 \times X_2$; and
- the **morphisms** $f : X \rightarrow Y$ consist of triples $\langle f_1, f_2, \dot{f} \rangle$, where $\langle f_1 \times f_2, \dot{f} \rangle$ is a predicate morphism from $\dot{X} \subseteq X_1 \times X_2$ to $\dot{Y} \subseteq Y_1 \times Y_2$.

We have the following forgetful functors:

$$\begin{array}{l}
 U_{\mathbf{Set} \times \mathbf{Set}} : \mathbf{LogRel} \quad \rightarrow \quad \mathbf{Set} \times \mathbf{Set} \quad \text{Cod} : \mathbf{Pred} \rightarrow \mathbf{Set} \\
 \langle X_1, X_2, \dot{X} \rangle \mapsto \langle X_1, X_2 \rangle \quad P \subseteq X \rightarrow X \\
 \langle f_1, f_2, \dot{f} \rangle \mapsto \langle f_1, f_2 \rangle \quad \langle f, \dot{f} \rangle \mapsto f \\
 U_{\mathbf{Pred}} : \mathbf{LogRel} \quad \rightarrow \quad \mathbf{Pred} \\
 \langle X_1, X_2, \dot{X} \rangle \mapsto \dot{X} \subseteq X_1 \times X_2 \\
 \langle f_1, f_2, \dot{f} \rangle \mapsto \langle f_1 \times f_2, \dot{f} \rangle
 \end{array}$$

In fact, **LogRel** arises as the pullback square in the (large) category of categories and functors:

$$\begin{array}{ccc}
 \mathbf{LogRel} & \xrightarrow{U_{\mathbf{Pred}}} & \mathbf{Pred} \\
 U_{\mathbf{Set} \times \mathbf{Set}} \downarrow & \lrcorner & \downarrow \text{Cod} \\
 \mathbf{Set} \times \mathbf{Set} & \xrightarrow{\times} & \mathbf{Set}
 \end{array}$$

This known characterisation is part of the logical relations folklore, see, for example, Katsumata [Kat13]. We expect this characterisation to be useful when generalising from sets and subsets to objects in a category and their predicates. As in **Pred**, note that the morphism \dot{f} , if it exists, is uniquely determined by the components f_1, f_2 . When \dot{f} exists, we say that $\langle f_1, f_2 \rangle$ *lifts* to **LogRel**.

The following result is well-known:

Proposition 9.4. *The category **LogRel** is cartesian closed and has all products and coproducts. As a consequence, it is distributive. The functor $U_{\mathbf{Set} \times \mathbf{Set}}$ preserves the products, coproducts and this closed structure. Explicitly, these structures are given as follows:*

- **products:** $\prod_{i \in I} X^i = \langle \prod_{i \in I} X_1^i, \prod_{i \in I} X_2^i, \dot{\prod}_{i \in I} X^i \rangle$ where

$$\dot{\prod}_{i \in I} X^i = \{ \langle \langle x_1^i \rangle, \langle x_2^i \rangle \rangle \mid \forall i \in I. \langle x_1^i, x_2^i \rangle \in \dot{X}^i \}$$

The pair of set-theoretic projections lifts to the product of predicates. Given, for all $i \in I$, a morphism $f^i : Y \rightarrow X^i$, then the pair $\langle \langle f_1^i \rangle_{i \in I}, \langle f_2^i \rangle_{i \in I} \rangle$ lifts to $\langle f^i \rangle_{i \in I}$.

- **coproducts:** $\sum_{i \in I} X^i = \langle \sum_{i \in I} X_1^i, \sum_{i \in I} X_2^i, \sum_{i \in I} \dot{X}^i \rangle$. Note that indeed we have

$$\sum \dot{X}^i \subseteq \sum (X_1^i \times X_2^i) \subseteq (\sum X_1^i) \times (\sum X_2^i)$$

The pair of set-theoretic injections lifts to the coproduct of predicates. Given, for all $i \in I$, a logical relations morphism $f^i : X^i \rightarrow Y$, then the pair $\langle [f_1^i]_{i \in I}, [f_2^i]_{i \in I} \rangle$ lifts to $[f^i]_{i \in I}$.

- **exponentials:** $Y^X = \langle Y_1^{X_1}, Y_2^{X_2}, R \rangle$, where

$$R = \left\{ \langle f_1, f_2 \rangle \in Y_1^{X_1} \times Y_2^{X_2} \mid \text{for all } \langle x_1, x_2 \rangle \text{ in } \dot{X}, \langle f_1(x_1), f_2(x_2) \rangle \in \dot{Y} \right\}$$

The pair $\langle \text{eval}, \text{eval} \rangle$ lifts to the evaluation map in **Pred**, and for every logical relation morphism $f : Z \times X \rightarrow Y$ the pair of functions $\langle \lambda X_1. f_1, \lambda X_2. f_2 \rangle$ lifts to $\lambda X. f$.

Proof

A vast generalisation of this result using bifibrations is discussed by Jacobs [Jac01, Section 9.2]. Katsumata [Kat13] spells out this structure for a more specialised situation, using faithful bifibrations, that includes our notion of logical relations. ■

9.2 Monadic lifting

Our goal is to construct *relational* CBPV models, i.e., models in **LogRel**, relating two models in **Set**. Such models are given by a pair of monads. Straightforward calculation shows that if T_1, T_2 are two (strong) monads over (cartesian) categories $\mathcal{V}_1, \mathcal{V}_2$, then setting $T \langle x, y \rangle$ to $\langle T_1 x, T_2 y \rangle$ yields a monad over $\mathcal{V}_1 \times \mathcal{V}_2$, with the monadic structure given componentwise, i.e. the unit by $\langle \eta_1, \eta_2 \rangle$ and the multiplication by $\langle \mu_1, \mu_2 \rangle$. The following definition captures the data we will need in our logical relations models.

Definition 9.5. Let $\langle T_1, T_2 \rangle$ be a pair of (strong) monads over **Set**. A lifting of T_1, T_2 to **LogRel** is a strong monad T , such that:

- for every logical relation X , TX is a lifting of T_1X_1 and T_2X_2 , and we write $TX = \langle T_1X_1, T_2X_2, \dot{T}X \rangle$;
- for every logical relation morphism f , $\langle T_1f_1, T_2f_2 \rangle$ lifts to Tf ;
- the monadic unit $\langle \eta_1, \eta_2 \rangle$ lifts to the unit of T ;
- the monadic multiplication $\langle \mu_1, \mu_2 \rangle$ lifts to the multiplication of T ; and
- the monadic strength $\langle \text{str}_1, \text{str}_2 \rangle$ lifts to the strength of T .

Note that, in order to lift a given pair of monads T_1, T_2 , the only additional structure required is the relation part $\dot{T} \langle X_1, X_2, \dot{X} \rangle$ of the monad T . The remainder of the monadic structure of the lifting is uniquely determined by the monadic structure of T_1 and T_2 , i.e., we only need to validate that the existing structure satisfies these properties.

Example 9-1. Every pair of monads T_1, T_2 admits a lifting via the total relation, i.e., by setting $\dot{T} \langle X_1, X_2, \dot{X} \rangle := T_1X_1 \times T_2X_2$. \square

The models, i.e. monads, Benton et al. [BK99, BKHB06, BB07, BKBH07, BKBH09] use to validate effect-dependent transformations are lifted models. Due to time and space constraints, we do not describe these models here explicitly.

Lemma 9.6. Let T be a lifting of T_1, T_2 . Let $\underline{B}_1, \underline{B}_2$ be algebras for T_1, T_2 , respectively, and \underline{B} a lifting of the algebra structure, i.e., a predicate $\dot{B} \subseteq |\underline{B}_1| \times |\underline{B}_2|$ such that the pair of algebra maps $\langle \underline{B}_1[-], \underline{B}_2[-] \rangle$ lifts. Thus, such a \underline{B} is an algebra for T . Further, for all logical relations morphisms $f : X \times Y \rightarrow |\underline{B}|$, the pair of monadic liftings $\langle f_1^\dagger, f_2^\dagger \rangle$ lifts to the monadic lifting $f^\dagger : X \times TY \rightarrow |\underline{B}|$.

Proof

As f^\dagger is expressed using $\lambda-.-, T$, and the tupling morphism, the monadic lifting also lifts to **LogRel**. \blacksquare

We define the lifting of other concepts, such as generic effects, monad morphisms, and algebraic operations similarly. Let A, P be logical relations. Let $\text{gen}_1 : A_1 \langle P_1 \rangle$, $\text{gen}_2 : A_2 \langle P_2 \rangle$ be generic effects for the monads T_1, T_2 , respectively. We say the pair $\langle \text{gen}_1, \text{gen}_2 \rangle$ lifts to a generic operation of type $A \langle P \rangle$ for T when the pair lifts as a function $P \rightarrow TA$.

Similarly, let op_i be two algebraic operations of types $A_i \langle P_i \rangle$ for $T_i, i = 1, 2$. The pair $\langle \text{op}_1, \text{op}_2 \rangle$ lifts when, for every T -algebra \underline{B} , the components $\langle \text{op}_1^{\underline{B}_1}, \text{op}_2^{\underline{B}_2} \rangle$ lift.

Let T, T' be liftings of T_1, T_2 , and T'_1, T'_2 , respectively. A pair of monad morphism $m_i : T_i \rightarrow T'_i, i = 1, 2$ lifts when each component $\langle m_1, m_2 \rangle$ lifts to a morphism of logical relations $m_X : TX \rightarrow T'X$.

Let Π be a set of operations. Let $\text{type}_1, \text{type}_2$ be set-theoretic type assignments for Π . A lifting of $\text{type}_1, \text{type}_2$ is a type assignment type for Π in **LogRel**, such that, for every $\text{op} \in \Pi$, if type assigns $\text{op} : A \langle P \rangle$, then type_i assigns $\text{op} : A_i \langle P_i \rangle$, for $i = 1, 2$. In the following, we are only concerned with such type assignment liftings which assign the pair of diagonal relations for every operation. Given two set-theoretic CBPV Π -models

$$\langle \mathbf{Set}, T_1, \text{type}_1, O_1 \llbracket - \rrbracket \rangle, \quad \langle \mathbf{Set}, T_2, \text{type}_2, O_2 \llbracket - \rrbracket \rangle$$

a lifting of these models is a CBPV Π -model

$$\langle \mathbf{LogRel}, T, \text{type}, O \llbracket - \rrbracket \rangle$$

such that T is a lifting of the given monads, type is a lifting of the type assignments, and, for every op in Π , $O \llbracket \text{op} \rrbracket$ is a lifting of $O_1 \llbracket \text{op} \rrbracket, O_2 \llbracket \text{op} \rrbracket$. Our goal is to lift a given pair of set-theoretic CBPV Π -models to such a relational model. In order to present our construction we require the following auxiliary notion.

Definition 9.7. Let T_1, T_2 be monads over **Set**, let X be a logical relation, and let $R \subseteq T_1X_1 \times T_2X_2$ be a relation.

- We say that the monadic unit respects R at X if $\eta_1 \times \eta_2[\dot{X}] \subseteq R$.
- Let A, P be logical relations and $\text{op}_1 : A_1 \langle P_1 \rangle, \text{op}_2 : A_2 \langle P_2 \rangle$ be algebraic operations for T_1, T_2 . We say that the operations $\langle \text{op}_1, \text{op}_2 \rangle$ respect R at X under $\langle A, P \rangle$ when $\langle \text{op}_1^{FX_1}, \text{op}_2^{FX_2} \rangle$ lifts to a logical relation morphism:

$$\text{op}^{FX} : \langle T_1X_1, T_2X_2, R \rangle^A \rightarrow \langle T_1X_1, T_2X_2, R \rangle^P$$

Explicitly, this condition holds when, for every $\langle \kappa_1, \kappa_2 \rangle \in (T_1X_1)^{A_1} \times (T_2X_2)^{A_2}$, for which $\langle a_1, a_2 \rangle \in \dot{A}$ implies $\langle \kappa_1(a_1), \kappa_2(a_2) \rangle \in R$, we have, for every pair $\langle p_1, p_2 \rangle \in \dot{P}$

$$\langle \text{op}_1(\kappa)(p_1), \text{op}_2(\kappa)(p_2) \rangle \in R$$

We present the central construction of this chapter:

Theorem 9.8. Let Π be a set of operations. Consider any two set-theoretic CBPV Π -models

$$\langle \mathbf{Set}, T_1, \text{type}_1, O_1 \llbracket - \rrbracket \rangle, \quad \langle \mathbf{Set}, T_2, \text{type}_2, O_2 \llbracket - \rrbracket \rangle$$

Let type be a lifting of $\text{type}_1, \text{type}_2$.

For every logical relation X , set $TX := \langle T_1X_1, T_2X_2, \bigcap \mathcal{R}X \rangle$, where $\mathcal{R}X$ is the set of all relations $R \subseteq T_1X_1 \times T_2X_2$ such that:

- the monadic unit respects R ; and
- for every $\text{op} \in \Pi$, $\text{op} : A \langle P \rangle$ via type , the operations $\langle \text{op}_1, \text{op}_2 \rangle$ respect R at X under $\langle A, P \rangle$.

Then:

- T is a lifting of T_1, T_2 ;
- for every $\text{op} \in \Pi$, $\langle O_1 \llbracket \text{op}_1 \rrbracket, O_2 \llbracket \text{op}_2 \rrbracket \rangle$ lift to an operation $O \llbracket \text{op} \rrbracket : A \langle P \rangle$ for T ;

Thus, $\langle \mathbf{LogRel}, T, \text{type}, O \llbracket - \rrbracket \rangle$ is a lifting of the two given models to type . In addition,

- for every other lifting

$$\langle \mathbf{LogRel}, T', \text{type}, O' \llbracket - \rrbracket \rangle$$

of the two given models with the same type assignment lifting, there is a (necessarily unique) monad morphism between the corresponding free liftings:

$$\langle \text{id}, \text{id}, m \rangle : T \rightarrow T'$$

preserving the operations in Π .

We call T the free lifting of T_1 and T_2 via type , and the resulting CBPV Π -model the free lifting model.

Proof

We proceed as follows. First, we show that for every logical relation X , $\bigcap \mathcal{R}X \in \mathcal{R}X$. Next, we show that the action of T on morphisms lifts. Thus T is indeed an endofunctor over \mathbf{LogRel} . We then show the monadic unit, multiplication and strength lift to T . Thus, T is a strong monad. Next, we show the operations lift, and obtain a lifting of the CBPV models. Finally, we show that this model is free.

Consider any logical relation X , and set $R := \bigcap \mathcal{R}X$.

Consider any $R' \in \mathcal{R}X$. As the unit respects R' , $\eta_1 \times \eta_2[\dot{X}] \subseteq R'$. Therefore

$$\eta_1 \times \eta_2[\dot{X}] \subseteq \bigcap \mathcal{R}X = R$$

i.e., the unit respects R .

Consider any $\text{op} \in \Pi$, $\text{op} : A \langle P \rangle$. Consider any $\langle \kappa_1, \kappa_2 \rangle \in (T_1 X_1)^{A_1} \times (T_2 X_2)^{A_2}$ satisfying, for every $\langle a_1, a_2 \rangle \in \dot{A}$, $\langle \kappa(a_1), \kappa(a_2) \rangle \in R$. Consider any $\langle p_1, p_2 \rangle \in \dot{P}$.

Consider any $R' \in \mathcal{R}X$. For every $\langle a_1, a_2 \rangle \in \dot{A}$, we have:

$$\langle \kappa_1(a_1), \kappa_2(a_2) \rangle \in R \subseteq R'$$

As op respects R' , we deduce that

$$\langle O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1), O_2 \llbracket \text{op} \rrbracket (\kappa_2)(p_2) \rangle \in R'$$

As we considered an arbitrary $R' \in \mathcal{R}X$, we deduce that:

$$\langle O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1), O_2 \llbracket \text{op} \rrbracket (\kappa_2)(p_2) \rangle \in \bigcap \mathcal{R}X = R$$

and op respects R . We showed thus that both the unit and operations respect R , i.e., R is in $\mathcal{R}X$.

Next, consider any predicate morphism $f = \langle f_1, f_2, \dot{f} \rangle : X \rightarrow Y$. Denote $R := \bigcap \mathcal{R}X$, $S := \bigcap \mathcal{R}Y$. Set $R' := (T_1 f_1 \times T_2 f_2)^{-1}[S]$. Then $R' \in \mathcal{R}X$.

The the unit respects relation R' . Indeed,

$$\begin{aligned} T_1 f_1 \times T_2 f_2 [\eta_1 \times \eta_2 [\dot{X}]] &= (T_1 f_1 \circ \eta_1) \times (T_2 f_2 \circ \eta_2) [\dot{X}] \\ &\begin{array}{ccc} \eta_i \text{ naturality} & f : X \rightarrow Y & S \in \bigcap \mathcal{R}Y \\ \downarrow & \downarrow & \downarrow \\ & = (\eta_1 \circ f_1) \times (\eta_2 \circ f_2) [\dot{X}] \subseteq \eta_1 \times \eta_2 [\dot{Y}] \subseteq S \end{array} \end{aligned}$$

Therefore

$$\eta_1 \times \eta_2 [\dot{X}] \subseteq (T_1 f_1 \times T_2 f_2)^{-1}[S] = R'$$

and the unit respects R' .

Consider any $\text{op} \in \Pi$, $\text{op} : A \langle P \rangle$. Then op respects the relation R' . Indeed, consider any $\langle \kappa_1, \kappa_2 \rangle \in (T_1 X_1)^{A_1} \times (T_2 X_2)^{A_2}$ satisfying, for every $\langle a_1, a_2 \rangle \in \dot{A}$, $\langle \kappa(a_1), \kappa(a_2) \rangle \in R'$, i.e.,

$$\langle T_1 f_1(\kappa_1(a_1)), T_2 f_2(\kappa_2(a_2)) \rangle \in S$$

Consider any $\langle p_1, p_2 \rangle \in \dot{P}$. As S respects op , we have

$$\langle O_1 \llbracket \text{op} \rrbracket (\lambda a_1. T_1 f_1(\kappa_1(a_1)))(p_1), O_2 \llbracket \text{op} \rrbracket (\lambda a_2. T_2 f_2(\kappa_2(a_2)))(p_2) \rangle \in S$$

Therefore

$$\begin{aligned} &T_1 f_1 \times T_2 f_2 (\langle O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1), O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1) \rangle) \\ &= \langle T_1 f_1(O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1)), T_2 f_2(O_2 \llbracket \text{op} \rrbracket (\kappa_2)(p_2)) \rangle \\ &\begin{array}{l} O_i \llbracket \text{op} \rrbracket \text{ naturality (see Example 7-10)} \\ \downarrow \\ = \langle O_1 \llbracket \text{op} \rrbracket (\lambda a_1. T_1 f_1(\kappa_1(a_1)))(p_1), O_2 \llbracket \text{op} \rrbracket (\lambda a_2. T_2 f_2(\kappa_2(a_2)))(p_2) \rangle \in S \end{array} \end{aligned}$$

Hence,

$$\langle O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1), O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1) \rangle \in (T_1 f_1 \times T_2 f_2)^{-1}[S] = R'$$

and thus op respects R' .

We showed that the unit and the operations respect R' , hence $R' \in \mathcal{R}X$, and thus $R \subseteq R$. Therefore

$$T_1 f_1 \times T_2 f_2[R] \subseteq T_1 f_1 \times T_2 f_2[R'] \subseteq S$$

Thus, $\langle T_1 f_1, T_2 f_2 \rangle$ lifts. We therefore have an endofunctor T over **LogRel**.

The monadic unit lifts, as the unit respects $\cap \mathcal{R}X$. The monadic multiplication also lifts. Indeed, consider any logical relation X . Denote $R := \cap \mathcal{R}X$, $S := \cap \mathcal{R}TX$. Set $S' := (\mu_1 \times \mu_2)^{-1}[R]$. Then $S' \in \mathcal{R}TX$.

Indeed,

$$\begin{array}{c} \text{monad law} \\ \downarrow \\ \mu_1 \times \mu_2[\eta_1 \times \eta_2[R]] = \text{id} \times \text{id}[R] = R \end{array}$$

Therefore, $\eta_1 \times \eta_2[R] \subseteq S'$, and the unit respects S' . Consider any $\text{op} \in \Pi$, $\text{op} : A \langle P \rangle$. Then op respects the relation S' . Indeed, consider any $\langle \kappa_1, \kappa_2 \rangle \in (T_1 X_1)^{A_1} \times (T_2 X_2)^{A_2}$ satisfying, for every $\langle a_1, a_2 \rangle \in \dot{A}$, $\langle \kappa(a_1), \kappa(a_2) \rangle \in S'$, i.e., $\langle \mu_1(\kappa_1(a_1)), \mu_2(\kappa_2(a_2)) \rangle$ is in R . Consider any $\langle p_1, p_2 \rangle \in \dot{P}$. As op respects R , we have

$$\langle O_1 \llbracket \text{op} \rrbracket (\mu_1 \circ \kappa_1)(p_1), O_2 \llbracket \text{op} \rrbracket (\mu_2 \circ \kappa_2)(p_2) \rangle \in R$$

Therefore,

$$\begin{aligned} \mu_1 \times \mu_2(\langle O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1), O_2 \llbracket \text{op} \rrbracket (\kappa_2)(p_2) \rangle) \\ = \langle \mu_1(O_1 \llbracket \text{op} \rrbracket (\kappa_1)(p_1)), \mu_2(O_2 \llbracket \text{op} \rrbracket (\kappa_2)(p_2)) \rangle \\ \text{see Example 7-10} \\ \downarrow \\ = \langle O_1 \llbracket \text{op} \rrbracket (\mu_1 \circ \kappa_1)(p_1), O_2 \llbracket \text{op} \rrbracket (\mu_2 \circ \kappa_2)(p_2) \rangle \in R \end{aligned}$$

and thus op respects S' . We showed the unit and the operations respect S' , hence $S' \in \mathcal{R}TX$. Consequently, μ lifts.

Consider any logical relations X, Y . From Kock [Koc72] follows that, for all $\langle x_i, k_i \rangle \in X_i \times T_i Y_i$, $\text{str}_i(x_i, k_i) = T_i(\lambda y_i. \langle x_i, y_i \rangle)(k_i)$. As the strengths are expressed using the cartesian closed structure and the action of T_i on morphisms, the strengths also lift.

Explicitly, consider any $\langle \langle x_1, k_1 \rangle, \langle x_2, k_2 \rangle \rangle \in X \dot{\times} TY$. Consider any $\langle y_1, y_2 \rangle \in \dot{Y}$. Then, $\langle \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \rangle \in X \dot{\times} Y$. Therefore, $\langle \lambda y_1. \langle x_1, y_1 \rangle, \lambda y_2. \langle x_2, y_2 \rangle \rangle$ lifts, hence by applying the lifted monad T we deduce that

$$\langle T_1(\lambda y_1. \langle x_1, y_1 \rangle), T_2(\lambda y_2. \langle x_2, y_2 \rangle) \rangle$$

lifts. As $\langle k_1, k_2 \rangle \in \cap \mathcal{R}Y$, we deduce that $\langle \text{str}_1(x_1, k_1), \text{str}_2(x_2, k_2) \rangle \in \cap \mathcal{R}(X \times Y)$, and the monadic strengths lift.

Every operation in Π lifts. Indeed, consider any $\text{op} \in \Pi$, $\text{op} : A \langle P \rangle$. For every logical relation X , as $O \llbracket \text{op} \rrbracket$ respects $\cap \mathcal{R}X$, we deduce that $O \llbracket \text{op} \rrbracket$ lifts at the component FX . Recall that all components of an algebraic operation $\text{op} : A_i \langle P_i \rangle$ can be expressed in terms of the component op_{FA_i} , the cartesian closed structure, and the strong monadic structure (see the proof of Corollary 2.5 and Theorem 2.4). We have just shown all these structures lift to logical relations, therefore all components of the interpretation of op lift. Thus, all operations in Π lift.

Finally, let $\langle \mathbf{LogRel}, T', \text{type}, O' \llbracket - \rrbracket \rangle$ be any other lifting. Then $\langle \text{id}, \text{id} \rangle$ lifts to a monad morphism from T to T' . Indeed, consider any logical relation X , and denote $T'X = \langle T_1X_1, T_2X_2, R \rangle$. Because T' is a lifting, the monadic unit lifts, hence the unit respects R . Similarly, because $O' \llbracket - \rrbracket$ is a lifting, all the operations in Π respect R . Therefore, $R \in \mathcal{R}X$, hence $\cap \mathcal{R}X \subseteq R$. Therefore the pair of identities lift to a morphism $TX \rightarrow T'X$. Thus $\langle \mathbf{LogRel}, T, \text{type}, O \llbracket - \rrbracket \rangle$ is the free lifting. ■

Our construction also extends to monad morphisms:

Theorem 9.9. *Let Π be a set of operations. Consider any four set-theoretic CBPV Π -models:*

$$\begin{aligned} & \langle \mathbf{Set}, T_1, \text{type}_1, O_1 \llbracket - \rrbracket \rangle, \langle \mathbf{Set}, T_2, \text{type}_2, O_2 \llbracket - \rrbracket \rangle, \\ & \langle \mathbf{Set}, T'_1, \text{type}_1, O'_1 \llbracket - \rrbracket \rangle, \langle \mathbf{Set}, T'_2, \text{type}_2, O'_2 \llbracket - \rrbracket \rangle \end{aligned}$$

Let type be a lifting of $\text{type}_1, \text{type}_2$. Every pair of monad morphisms $m_1 : T_1 \rightarrow T'_1$, $m_2 : T_2 \rightarrow T'_2$ that preserve the operations in Π lifts to a monad morphism

$$\langle m_1, m_2, m \rangle : T \rightarrow T'$$

As a consequence, this monad morphism preserves the lifted operations.

Proof

The proof amounts to showing that each component of the pair of monad morphisms lifts. We proceed along the same lines of the previous proof. Consider any logical relation X , and denote $R := (m_1 \times m_2)^{-1}[\cap \mathcal{R}'X]$, where $\mathcal{R}'X$ is the relation component of $T'X$. The preservation of units under monad morphisms implies the monadic unit respects R . The assumption that the monad morphisms preserves the operations implies the operations in Π respect R . Thus, $R \in \mathcal{R}X$, and $\langle m_1, m_2 \rangle$ lifts. ■

Before we conclude our discussion of set-theoretic lifting, we investigate its action on diagonal relations, i.e., logical relations of the form $\langle X, X, = \rangle$.

Lemma 9.10. *Let Π be a set of operations. Consider any two CBPV Π -models with the same type assignment:*

$$\langle \mathbf{Set}, T_1, \text{type}, O_1 \llbracket - \rrbracket \rangle, \langle \mathbf{Set}, T_2, \text{type}, O_2 \llbracket - \rrbracket \rangle$$

Let $m : T_1 \rightarrow T_2$ be a monad morphism preserving the operations in Π .

Denote by type the lifting of type , assigning to each operation the pair of diagonal relations of its type assignment via type , i.e., if $\text{type}(\text{op}) = \langle A, P \rangle$, then $\text{type}(\text{op}) = \langle \langle A, A, = \rangle, \langle P, P, = \rangle \rangle$. Denote by T_Π the free monad for the signature $\langle \Pi, \text{type} \rangle$. Denote by $e : T_\Pi \rightarrow T_1$ the (unique) monad morphism preserving Π .

For every set X , there exists a (unique) function $f : T_\Pi X \rightarrow \bigcap \mathcal{R} \langle X, X, = \rangle$ satisfying

$$\begin{array}{ccc} T_\Pi X & \xrightarrow{g} & T_1 X \\ f \downarrow & = & \downarrow \langle \text{id}, m \rangle \\ \dot{T} X & \subseteq & T_1 X \times T_2 X \end{array}$$

Proof

We use the inductive description of T_Π as the term algebra for the signature induced by $\langle \Pi, \text{type} \rangle$. We construct f by induction, essentially as $\langle \text{id}, m \rangle \circ g$. Because the monadic unit respects $\dot{T} X$ that preserves the operations in Π , f is well-defined. We then prove by induction on $T_\Pi X$ that the square commutes for all $k \in T_\Pi X$. Uniqueness of f follows as inclusions are monic. ■

We can now calculate the explicit description of the action of a class of free lifting on diagonal relations:

Theorem 9.11. *Let Π be a set of operations, T_2 be a finitary monad, and*

$$\langle \mathbf{Set}, T_2, \text{type}, O_2 \llbracket - \rrbracket \rangle$$

be a CBPV Π -model. Let T_Π be the free monad for the signature $\langle \Pi, \text{type} \rangle$, and denote by $T_\Pi \xrightarrow{e} T_1 \xrightarrow{m} T_2$ the conservative restriction factorisation corresponding to the surjection-injection factorisation of \mathbf{Set} , giving rise to the CBPV Π -model

$$\langle \mathbf{Set}, T_1, \text{type}, O_1 \llbracket - \rrbracket \rangle$$

Let type be the diagonal lifting of type , and T be the free lifting arising from type .

For every set X ,

$$\dot{T} \langle X, X, = \rangle = \langle \text{id}, m \rangle [T_1 X] = \{ \langle k, m(k) \rangle \mid k \in T_1 X \}$$

Proof

Consider any set X . Denote $R := \dot{T} \langle X, X, = \rangle$, $S := \langle \text{id}, m \rangle [T_1 X]$. We show that $R = S$. First, note that the unit respects $S \subseteq T_1 X \times T_2 X$ by the definition of monad morphisms, and the operations respect S by the definition of preservation of operations by monad morphisms. Therefore, by definition, $R \subseteq S$.

For the converse inclusion, recall that the \mathcal{E} -morphisms in the factorisation system of finitary monads arising from the injection-surjection factorisation of **Set** are componentwise surjective, Corollary 8.12. Therefore, by the previous lemma, we have a function $f : T_{\Pi} X \rightarrow R$, satisfying

$$\begin{array}{ccc} T_{\Pi} X & \xrightarrow{e} & T_1 X \\ f \downarrow & = & \downarrow \langle \text{id}, m \rangle \\ R & \xrightarrow{\subseteq} & T_1 X \times T_2 X \end{array}$$

Therefore, there exists a unique fill-in diagonal

$$\begin{array}{ccc} T_{\Pi} X & \xrightarrow{e} & T_1 X \\ f \downarrow & \begin{array}{c} = \\ h \end{array} & \downarrow \langle \text{id}, m \rangle \\ R & \xrightarrow{\subseteq} & T_1 X \times T_2 X \end{array}$$

Chasing the lower-right triangle shows that, for all $k \in T_1 X$, $\langle k, m(k) \rangle = h(k) \in R$. Thus, $S \subseteq R$. ■

We employ Theorems 9.8, 9.9, and 9.11 to relate the conservative restriction models and the benchmark models. Consider any two Σ -models

$$\mathcal{M}_1 = \langle \mathbf{Set}, \text{type}_1, \mathbb{P}_1, O_-^1[-] \rangle, \quad \mathcal{M}_2 = \langle \mathbf{Set}, \text{type}_2, \mathbb{P}_2, O_-^2[-] \rangle,$$

for some effect hierarchy Σ . A lifting of $\mathcal{M}_1, \mathcal{M}_2$ is a Σ -model

$$\mathcal{M} = \langle \mathbf{LogRel}, \text{type}, \mathbb{P}, O_-[-] \rangle$$

where:

- type is a lifting of $\text{type}_1, \text{type}_2$;
- for every $\varepsilon \in \mathcal{E}$, $\mathbb{P}\varepsilon$ is a lifting of $\mathbb{P}_1\varepsilon, \mathbb{P}_2\varepsilon$;
- for every $\varepsilon \subseteq \varepsilon'$ in \mathcal{E} , $\mathbb{P}(\varepsilon \subseteq \varepsilon')$ is a lifting of $\mathbb{P}_1(\varepsilon \subseteq \varepsilon'), \mathbb{P}_2(\varepsilon \subseteq \varepsilon')$; and
- for every $\varepsilon \in \mathcal{E}$ and $\text{op} \in \varepsilon$, $O_\varepsilon \llbracket \text{op} \rrbracket$ is a lifting of $O_\varepsilon^1 \llbracket - \rrbracket, O_\varepsilon^2 \llbracket - \rrbracket$.

Note that in order to lift two Σ -models, the only additional structure we require is the lifting of the type assignments and the lifting of each monad in the hierarchy. The other conditions are properties that need to be verified.

Using this terminology, we obtain the following corollary:

Corollary 9.12. *Let Σ be an effect hierarchy, and \mathcal{M} an algebraic CBPV Π -model. Denote by type the diagonal lifting of the type assignment of \mathcal{M} . Consider the benchmark model (see Example 7-2)*

$$\mathcal{M}_b = \langle \mathbf{Set}, \text{type}, \mathbb{P}_b, O_-^b \llbracket - \rrbracket \rangle$$

and the conservative restriction model (see Theorem 7.12)

$$\mathcal{M}_\sharp = \langle \mathbf{Set}, \text{type}, \mathbb{P}_\sharp, O_-^\sharp \llbracket - \rrbracket \rangle$$

The conservative comparison relational model \mathcal{M} is given by

$$\mathcal{M} = \langle \mathbf{LogRel}, \text{type}, \mathbb{P}, O_- \llbracket - \rrbracket \rangle$$

where for every $\varepsilon \in \mathcal{E}$, $\langle \mathbf{LogRel}, \mathbb{P}\varepsilon, \text{type}, O_\varepsilon \llbracket - \rrbracket \rangle$ is the free lifting of the two CBPV Π -models $\langle \mathbf{Set}, \mathbb{P}_\sharp\varepsilon, \text{type}, O_\varepsilon^\sharp \llbracket - \rrbracket \rangle$ and $\langle \mathbf{Set}, \mathbb{P}_b\varepsilon, \text{type}, O_\varepsilon^b \llbracket - \rrbracket \rangle$. Moreover, for every set X , we have

$$\mathbb{P}\varepsilon \langle X, X, = \rangle = \langle \mathbb{P}_\sharp\varepsilon X, \mathbb{P}_b\varepsilon X, \langle \text{id}, m_\varepsilon^X \rangle \llbracket \mathbb{P}_\sharp\varepsilon X \rrbracket \rangle$$

where $m_\varepsilon^X : \mathbb{P}_\sharp\varepsilon X \rightarrow TX = \mathbb{P}_b\varepsilon X$ is the injection given in the definition of the conservative restriction model.

We defined the category of set-theoretic logical relations models and presented the free lifting construction, culminating in our central result, Corollary 9.12.

Part II

Effect-dependent optimisation

Chapter 10

Generic intermediate language

Speak my language!

—The Cure



In this chapter we give a general definition of a Gifford-style type-and-effect system. We begin with a given source-level language with algebraic effects and its semantics. We then define the type-and-effect system for this language. We think of the annotated language as an intermediate language for code optimisation. Using the semantic constructs of Part I we define semantics to the intermediate language, and relate it to the source language semantics.

First, in Section 10.1, we define the source language, its type system, and its categorical and set-theoretic semantic structures. Next, in Section 10.2, we define the intermediate language corresponding to this source language, together with its type-and-effect system and categorical, relational, and set-theoretic semantic structures. Finally, in Section 10.3, we generate semantics for the intermediate language from a given algebraic semantics to the source language, and relate this generated semantics to the original, unannotated, semantics.

10.1 CBPV with algebraic operations

We need to develop our effect analysis over a concrete concise language. In order to remain general, we need a fundamental lambda calculus that is well-designed to deal with effects. Levy's Call-by-Push-Value [Lev04] fits this description exactly. The additional benefits of CBPV are: it subsumes both the call-by-value and call-by-name paradigms, opening application areas in both ML and HASKELL; Pretnar and Plotkin's

account of *effect handlers* [PP09a, Pre09] is formulated in terms of CBPV, hence our account requires less modifications to accommodate handlers; and, as we will see in Section 11.2, the CBPV paradigm decomposes complicated optimisations into orthogonal ones.

10.1.1 Syntax and type system

We present a family of CBPV languages parametrised by *signatures*, at both the syntax and the type system levels. We begin with the syntax level:

Definition 10.1. A CBPV syntax signature Π^{syntax} is a triple $\langle \mathbf{Bsc}, \Pi, S \rangle$, where:

- **Bsc** is a set of basic types, ranged over by Q ;
- Π is a set of effect operation symbols, ranged over by op ; and
- S is a set of built-in constants, ranged over by c .

For example, **Bsc** may include the types:

- **Word** for 64-bit words;
- **Str** for strings; and
- **Loc** for memory locations;
- **Exc** for exceptions.
- **Char** for characters;

The set Π may include the operation symbols:

- lookup, update for accessing state.
- input, output for reading input and printing output.
- raise for raising an exception.

The set S typically includes primitives to manipulate basic values, such as:

- number literals 0, 1, 2, 0xFEED;
- string manipulation primitives, e.g. string concatenation ++; and
- boolean operations =, >=, <;
- predefined exception constants such as ArithmeticOverflow and DivideByZero.
- string literals "cabab";

Given a syntax signature Π^{syntax} , the syntax of CBPV is given by Figure 10.1. It follows the CBPV dichotomy between values and computations. We make this distinction explicit using a kind system in preparation for the more sophisticated kind system we will need for the intermediate language in the next section. Using a kind system for both highlights the differences between and similarities of the two languages.

Our types are the standard CBPV types. Note that basic types are always value types. The unit, product, zero, and sum value types are standard. The type $\underline{U}B$ is the type of thunks, i.e., suspended computations, of type B . The *returner type* FA is the type of computations that return a value of type A . It plays a similar rôle to that of the monadic type TA (where T is a monad) in Haskell. We also have products of computations, and functions that are computations that depend on a value. The *ground value types* $G \in \mathbf{Gnd}$ are those value types which do not include thunks. We call types of the form FG *ground returner types*.

All of the built-in constants of CBPV are value terms. The variables, unit value, and pairing construct are standard. Injections are annotated with their sum type. Computations M are thunked into values **thunk** M .

In CBPV, we can turn any value into a computation by returning it. The construct $M \mathbf{to} x : A. N$ sequences computations; it is analogous to the HASKELL construct $x \leftarrow M; N$ or the ‘let’ construct in ML. Note the intrinsic typing (a.k.a. Church-style typing).

We briefly describe an operational intuition behind the CBPV syntax. The standard operational semantics of CBPV uses a stack machine, similar to other stack machines for call-by-value and call-by-name semantics of the lambda calculus, such as Felleisen and Friedman’s CK-machine [FF86] and the Krivine machine¹. The two λ terms pop from the stack while the application terms $V'M$ and $\mathbf{i}'M$ push the onto it. Thus, $\lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \}$ pops a *tag* off the stack and executes M_1 or M_2 accordingly. In turn, $\mathbf{1}'M$ pushes the tag $\mathbf{1}$ onto the stack and continues to execute M . Similarly, $\lambda x : A. M$ pops a value of type A and binds x to it, while $V'M$ pushes V onto the stack.

The pattern-matching terms eliminating products, zero and sum values are standard. Thunked computations are eliminated by forcing.

Importantly, computational effects are caused by the $\text{op}_A^B M$ terms. As an example, the following computation consists of a memory lookup operation, dereferencing

¹Jean-Louis Krivine, *Un interpréteur du λ -calcul*, unpublished, 1986.

Kinds: $K ::= \text{Val} \mid \text{Comp}$

Value

Types: $A, B, \dots ::= Q \mid \mathbf{1} \mid A_1 \times A_2 \mid \mathbf{0} \mid A_1 + A_2 \mid \underline{U}B$

Computation

Types: $\underline{A}, \underline{B}, \dots ::= \text{FA} \mid \underline{B}_1 \times \underline{B}_2 \mid A \rightarrow \underline{B}$

Ground Value

Types: $G ::= Q \mid \mathbf{1} \mid G_1 \times G_2 \mid \mathbf{0} \mid G_1 + G_2$

Value

Terms: $V ::= b \mid x \mid \star \mid (V_1, V_2) \mid \mathbf{inj}_1^{A_1+A_2} V$
 $\quad \quad \quad \mid \mathbf{inj}_2^{A_1+A_2} V \mid \mathbf{thunk} M$

Computation

Terms: $M, N, \dots ::= \mathbf{return} V \mid M \mathbf{to} x:A.N$
 $\quad \quad \quad \mid \lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \} \mid \mathbf{1}'M \mid \mathbf{2}'M \mid \lambda x:A.M$
 $\quad \quad \quad \mid V'M \mid \mathbf{match} V \mathbf{as} (x_1:A_1, x_2:A_2).M \mid \mathbf{match} V : \mathbf{0} \mathbf{as} \{ \}^B$
 $\quad \quad \quad \mid \mathbf{match} V \mathbf{as} \{ \mathbf{inj}_1 x_1:A_1.M_1, \mathbf{inj}_2 x_2:A_2.M_2 \} \mid \mathbf{force} V \mid \text{op}_V^B M$

Figure 10.1: CBPV syntax

$$\begin{array}{c}
\frac{\vdash_k Q : \mathbf{Val} \quad \vdash_k \mathbf{1} : \mathbf{Val} \quad \frac{\vdash_k A_1 : \mathbf{Val} \quad \vdash_k A_2 : \mathbf{Val}}{\vdash_k A_1 \times A_2 : \mathbf{Val}}}{\vdash_k \mathbf{0} : \mathbf{Val}} \\
\frac{\vdash_k A_1 : \mathbf{Val} \quad \vdash_k A_2 : \mathbf{Val} \quad \frac{\vdash_k \underline{B} : \mathbf{Comp}}{\vdash_k \underline{U}\underline{B} : \mathbf{Val}}}{\vdash_k A_1 + A_2 : \mathbf{Val}} \\
\frac{\frac{\vdash_k A : \mathbf{Val}}{\vdash_k \mathbf{F}A : \mathbf{Comp}} \quad \frac{\vdash_k \underline{B}_1 : \mathbf{Comp} \quad \vdash_k \underline{B}_2 : \mathbf{Comp}}{\vdash_k \underline{B}_1 \times \underline{B}_2 : \mathbf{Comp}}}{\vdash_k A \rightarrow \underline{B} : \mathbf{Comp}}
\end{array}$$

Figure 10.2: CBPV kind system

memory location ℓ , followed by returning the memory word w stored there:

$$\text{deref!}(\ell) := \text{lookup}_\ell^{\mathbf{FWord}}(\lambda w : \mathbf{Word}. \mathbf{return} w)$$

The *effect operation symbol* lookup takes as a *parameter* the location ℓ to be dereferenced. It then dereferences the memory word at location ℓ , binds the result to w , and proceeds to execute $\mathbf{return} w$.

As another example, consider a non-deterministic choice operator choose , and a computation for non-deterministic coin tossing:

$$\text{toss} := \text{choose}_\star^{\mathbf{F}(1+1)}(\lambda v : \mathbf{1} + \mathbf{1}. \mathbf{return} v)$$

In this case the parameter is the unit value \star .

In general, $\text{op}_A^B M$ is an effect operation term with parameter V and *argument* M . Terms of the form

$$\text{gen}_{\text{op}}(A) := \text{op}_V^{\mathbf{F}A}(\lambda x : A. \mathbf{return} x)$$

are called *generic effects*. Thus deref! and toss are the generic effects corresponding to lookup and choose respectively.

The kind system for CBPV, given a syntax signature Π^{syntax} , is displayed in Figure 10.2; it consists of a kind judgement relation \vdash_k between types and kinds. We denote by \mathbf{Val} the set of well-kinded value types $\{A \mid \vdash_k A : \mathbf{Val}\}$. Similarly, we write \mathbf{Comp} for the set of well-kinded computation types. Normally, CBPV is not presented

with a kind system, as the kind system can be enforced by the BNF grammar for types. Thus, **Val** contains all value types, and **Comp** includes all computation types. A *well-kinded context* $\Gamma : \text{Dom}(\Gamma) \rightarrow \mathbf{Val}$ is a function from a *finite* set of variables to **Val**. We write $\Gamma, x : A$ for the extension $\Gamma[x \mapsto A]$.

Given a term signature Π^{syntax} , an *arity assignment* $\text{type} : \Pi \rightarrow \mathbf{Gnd} \times \mathbf{Gnd}$ sends elements of Π to pairs of ground types. When $\text{type}(\text{op}) = \langle A, P \rangle$ we write instead $\text{op} : A \langle P \rangle$. The first component A is called the *arity type* and the second component P is called the *parameter type*. When the parameter type is $P = \mathbf{1}$ we simply write $\text{op} : A$. When $\text{op} : \mathbf{0}$ we call an *(effect) constant symbol*.

For example, $\text{lookup} : \mathbf{Word} \langle \mathbf{Loc} \rangle$ as memory look-up takes a location as parameter and its argument expects the corresponding memory word. Similarly, $\text{choose} : \mathbf{2}$ where, $\mathbf{2} = \mathbf{1} + \mathbf{1}$, as choose has a trivial parameter. We say that choose is a *binary operation symbol*.

A *constant type assignment* is any function $A_- : S \rightarrow \mathbf{Val}$. Example constant type assignments are:

$$\begin{aligned} \text{'a'} &: \mathbf{Char} \\ ++ &: \mathbf{U}((\mathbf{Str} \times \mathbf{Str}) \rightarrow \mathbf{F Str}) \\ + &: \mathbf{U}(\mathbf{Word} \times \mathbf{Word} \rightarrow \mathbf{F Word}) \\ \text{call/cc}_B^V &: \mathbf{U}(\mathbf{U}(V \rightarrow \underline{B}) \rightarrow \underline{B}) \rightarrow \underline{B} \end{aligned}$$

We define type level CBPV signatures, and complete the parametrisation of our CBPV language:

Definition 10.2. A CBPV signature is a triple $\Pi^{\text{CBPV}} = \langle \Pi^{\text{syntax}}, \text{type}, A_- \rangle$ where:

- Π^{syntax} is a CBPV term signature;
- type is an arity assignment for Π^{syntax} ; and
- $A_- : S \rightarrow \mathbf{Val}$ is a constant type assignment for Π^{syntax} .

The type system of CBPV, given a signature Π^{CBPV} , is displayed in Figure 10.3; it is given by type judgement relations $\Gamma \vdash_v A : A$ and $\Gamma \vdash_c M : \underline{B}$, where Γ, A, \underline{B} are well-kinded contexts, value types and computation kinds, respectively, and where A, M are value and computation terms, respectively. Signatures Π^{CBPV} determine the language of CBPV, meaning its syntax and kind and typing relations. We call closed ground returners $\vdash_c P : \mathbf{FG}$ programs.

$$\begin{array}{c}
\Gamma \vdash_v c : A_c \quad \frac{\Gamma(x) = A}{\Gamma \vdash_v x : A} \quad \Gamma \vdash_v \star : \mathbf{1} \quad \frac{\Gamma \vdash_v V_1 : A_1 \quad \Gamma \vdash_v V_2 : A_2}{\Gamma \vdash_v (V_1, V_2) : A_1 \times A_2} \\
\\
\frac{\Gamma \vdash_v V : A_i}{\Gamma \vdash_v \mathbf{inj}_i^{A_1 + A_2} V : A_1 + A_2} \quad (\mathbf{i} = \mathbf{1}, \mathbf{2}) \quad \frac{\Gamma \vdash_c M : \underline{B}}{\Gamma \vdash_v \mathbf{thunk} M : \underline{UB}} \\
\\
\frac{\Gamma \vdash_v V : A}{\Gamma \vdash_c \mathbf{return} V : \underline{FA}} \\
\\
\frac{\Gamma \vdash_c M : \underline{FA} \quad \Gamma, x : A \vdash_c N : \underline{B}}{\Gamma \vdash_c M \mathbf{to} x : A. N : \underline{B}} \\
\\
\frac{\Gamma \vdash_c M_1 : \underline{B}_1 \quad \Gamma \vdash_c M_2 : \underline{B}_2}{\Gamma \vdash_c \lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \} : \underline{B}_1 \times \underline{B}_2} \quad \frac{\Gamma \vdash_c M : \underline{B}_1 \times \underline{B}_2}{\Gamma \vdash_c \mathbf{i}' M : \underline{B}_i} \quad (\mathbf{i} = \mathbf{1}, \mathbf{2}) \\
\\
\frac{\Gamma, x : A \vdash_c M : \underline{B}}{\Gamma \vdash_c \lambda x : A. M : A \rightarrow \underline{B}} \quad \frac{\Gamma \vdash_v V : A \quad \Gamma \vdash_c M : A \rightarrow \underline{B}}{\Gamma \vdash_c V' M : \underline{B}} \\
\\
\frac{\Gamma \vdash_v V : A_1 \times A_2 \quad \Gamma, x_1 : A_1, y : A_2 \vdash_c M : \underline{B}}{\Gamma \vdash_c \mathbf{match} V \mathbf{as} (x_1 : A_1, x_2 : A_2). M : \underline{B}} \quad (x_1 \neq x_2) \\
\\
\frac{\Gamma \vdash_v V : \mathbf{0}}{\Gamma \vdash_c \mathbf{match} V : \mathbf{0} \mathbf{as} \{ \}^{\underline{B}} : \underline{B}} \\
\\
\frac{\Gamma \vdash_v V : A_1 + A_2 \quad \Gamma, x_1 : A_1 \vdash_c M_1 : \underline{B} \quad \Gamma, x_2 : A_2 \vdash_c M_2 : \underline{B}}{\Gamma \vdash_c \mathbf{match} V \mathbf{as} \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} : \underline{B}} \\
\\
\frac{\Gamma \vdash_v V : \underline{UB}}{\Gamma \vdash_c \mathbf{force} V : \underline{B}} \\
\\
\frac{\Gamma \vdash_c V : P \quad \Gamma \vdash_c M : A \rightarrow \underline{B}}{\Gamma \vdash_c \mathbf{op}_V^B M : \underline{B}} \quad (\mathbf{op} : A \langle P \rangle, \mathbf{op} \in \Pi)
\end{array}$$

Figure 10.3: CBPV type system

The type system is straightforward, apart from the rules for effect operation symbols. The effect operation rules formalise the informal explanation given earlier. Another way to view this rule is via continuation passing — we pass a continuation M that, depending on the effect's result, proceeds after the effect has been caused. For example, the rules for the I/O effects $\text{input} : \text{Char}$ and $\text{output} : \mathbf{1} \langle \text{Char} \rangle$ are:

$$\frac{\Gamma \vdash_c M : \text{Char} \rightarrow \underline{B}}{\Gamma \vdash_c \text{input}^B M : \underline{B}} \quad \frac{\Gamma \vdash_v A : \text{Char} \quad \Gamma \vdash_c M : \mathbf{1} \rightarrow \underline{B}}{\Gamma \vdash_c \text{output}_A^B M : \underline{B}}$$

The latter is analogous to Levy's `print` statement [Lev04]. For the corresponding generic effects we derive the familiar `get`, `put`:

$$\Gamma \vdash_c \text{get} : \mathbf{F} \text{Char} \quad \frac{\Gamma \vdash_v A : \text{Char}}{\Gamma \vdash_c \text{put} : \mathbf{F} \mathbf{1}}$$

Theorem 10.3. *Every well-kinded type has a unique kind, and every well-typed term has a unique type.*

Proof

Standard induction. ■

We define capture avoiding substitution in the usual manner. We will only substitute value terms for variables. The variable binders are the following constructs:

- $M \text{ to } x : A. N$ binds x in N ;
- $\lambda x : A. M$ binds x in M ;
- **match** V **as** $(x_1 : A_1, x_2 : A_2). M$ binds x_1 and x_2 in M ; and
- **match** V **as** $\{\text{inj}_1 x_1 : A_1. M_1, \text{inj}_2 x_2 : A_2. M_2\}$ binds x_1 in M_1 and x_2 in M_2 .

As usual, we identify terms up to α -equivalence.

Lemma 10.4 (substitution). *For every well-typed phrase (value term or computation term) $\Gamma \vdash P : X$, for all well-kinded contexts Δ and $\text{Dom}(\Gamma)$ -indexed family of values V_- , satisfying, for all $x \in \text{Dom}(\Gamma)$, $\Delta \vdash V_x : \Gamma(x)$, we have $\Delta \vdash P[V_x/x]_{x \in \text{Dom}(\Gamma)}$.*


Proof

Standard induction. ■

$$\begin{array}{ll}
\mathcal{V}[\mathbf{Q}] := \mathcal{B}[\mathbf{Q}] & \mathcal{CTX}[\Gamma] := \prod_{x \in \text{Dom}(\Gamma)} [\Gamma(x)] \\
\mathcal{V}[\mathbf{1}] := \mathbf{1} & \\
\mathcal{V}[A_1 \times A_2] := [A_1] \times [A_2] & \\
\mathcal{V}[\mathbf{0}] := \mathbf{0} & C[\mathbf{F}A] := F[A] \\
\mathcal{V}[A_1 + A_2] := [A_1] + [A_2] & C[\underline{B}_1 \times \underline{B}_2] := [\underline{B}_1] \times [\underline{B}_2] \\
\mathcal{V}[\mathbf{U}\underline{B}] := |[\underline{B}]| & C[A \rightarrow \underline{B}] := [\underline{B}]^{[A]}
\end{array}$$

Figure 10.4: CBPV type interpretation

10.1.2 Categorical semantics

 We now turn to the denotational semantics of CBPV. Let \mathcal{V} be a distributive category, and Π^{syntax} a CBPV syntax signature. A *base-type interpretation* is an assignment of a \mathcal{V} -object $\mathcal{B}[\mathbf{Q}]$ to every basic type $\mathbf{Q} \in \mathbf{Bsc}$. This assignment extends to a *ground-type interpretation* $\mathcal{G}[-]$, which assigns, to every $G \in \mathbf{Gnd}$ a \mathcal{V} -object:

$$\begin{array}{l}
\mathcal{G}[\mathbf{Q}] := \mathcal{B}[\mathbf{Q}] \\
\mathcal{G}[\mathbf{1}] := \mathbf{1} \quad \mathcal{G}[G_1 \times G_2] := [G_1] \times [G_2] \\
\mathcal{G}[\mathbf{0}] := \mathbf{0} \quad \mathcal{G}[G_1 + G_2] := [G_1] + [G_2]
\end{array}$$

We will henceforth omit the name of the semantic function if it can be inferred from the context. For example, we may choose $\mathcal{B}[\mathbf{Word}]$ and $\mathcal{B}[\mathbf{Loc}]$ to be 2^{64} in a 64-bit setting, and $\mathcal{B}[\mathbf{Char}]$ to be 2^7 for ASCII characters. As these interpretations are finite sets, every ground type involving them denotes a finite set.

Let $\langle \mathcal{V}, T \rangle$ be a CBPV model. Denote by $F \dashv |-| : \mathcal{V}^T \rightarrow \mathcal{V}$ the Eilenberg-Moore resolution for the monad T . Given a term signature Π^{syntax} and a base-type interpretation $\mathcal{G}[-]$, we define a *type interpretation* for kind judgements of types (see Figure 10.4):

- values are interpreted as \mathcal{V} -objects via $\mathcal{V}[-]$;
- computations are interpreted as \mathcal{V}^T -objects, i.e., algebras, via $C[-]$; and
- contexts as finite products in \mathcal{V} via $\mathcal{CTX}[-]$.

This interpretation is straightforward. Note that if $\underline{B}_1, \underline{B}_2$ are two algebras for T , then the product of their underlying objects, $|\underline{B}_1| \times |\underline{B}_2|$ carries an algebra structure, given

by:

$$\underline{B}_1 \times \underline{B}_2 \llbracket - \rrbracket : T(|\underline{B}_1| \times |\underline{B}_2|) \xrightarrow{\langle T\pi_1, T\pi_2 \rangle} T|\underline{B}_1| \times T|\underline{B}_2| \xrightarrow{\underline{B}_1 \llbracket - \rrbracket \times \underline{B}_2 \llbracket - \rrbracket} |\underline{B}_1| \times |\underline{B}_2|$$

The fact that this is indeed an algebra for T follows from the solution of an exercise in Mac Lane's book [ML98, Exercise VI.2.2]. Similarly, the algebra structure on $|\underline{B}|^A$ is well-defined (see Lemma 4.3(2)).

Let $\langle \mathcal{V}, T \rangle$ be a CBPV model, Π^{CBPV} a CBPV signature, and $\mathcal{B} \llbracket - \rrbracket$ a base-type interpretation. We define a type assignment $\mathcal{R} \llbracket \text{type} \rrbracket$ for Π in \mathcal{V} by assigning $\text{op} : \llbracket A \rrbracket \langle \llbracket P \rrbracket \rangle$ to every $\text{op} \in \Pi$, $\text{op} : A \langle P \rangle$.

With these notions in place, we are ready to define the semantic data required for modelling CBPV with effects:

Definition 10.5. Let Π^{CBPV} be a CBPV signature. A Π^{CBPV} -model \mathcal{M} is a quintuple

$$\langle \mathcal{V}, \mathcal{B} \llbracket - \rrbracket, T, \mathcal{O} \llbracket - \rrbracket, \mathcal{K} \llbracket - \rrbracket \rangle$$

where:

- \mathcal{V} is a distributive category;
- $\mathcal{B} \llbracket - \rrbracket$ is a basic-type assignment;
- $\langle \mathcal{V}, T, \llbracket \text{type} \rrbracket, \mathcal{O} \llbracket - \rrbracket \rangle$ is a CBPV Π -model (cf. Definition 2.13);
- $\mathcal{K} \llbracket - \rrbracket$ assigns to every constant $c \in S$ an arrow

$$\mathcal{K} \llbracket c \rrbracket : \mathbb{1} \rightarrow \llbracket A_c \rrbracket$$

Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{B} \llbracket - \rrbracket, T, \mathcal{O} \llbracket - \rrbracket, \mathcal{K} \llbracket - \rrbracket \rangle$ be a Π^{CBPV} -model. We interpret the CBPV terms as follows (see Figure 10.5):

- value terms $\Gamma \vdash_v V : A$ are interpreted as \mathcal{V} -morphisms

$$\mathcal{V}\mathcal{T} \llbracket V \rrbracket : \text{CTX} \llbracket \Gamma \rrbracket \rightarrow \mathcal{V} \llbracket A \rrbracket$$

- computation terms $\Gamma \vdash_c M : \underline{B}$ are interpreted as \mathcal{V} -morphisms

$$\text{CT} \llbracket M \rrbracket : \text{CTX} \llbracket \Gamma \rrbracket \rightarrow |\mathcal{C} \llbracket \underline{B} \rrbracket|$$

$$\begin{aligned}
\llbracket c \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\mathbb{1}} \mathbb{1} \xrightarrow{\mathcal{K}[\llbracket c \rrbracket]} \llbracket A_c \rrbracket & \llbracket x \rrbracket : \llbracket \Gamma \rrbracket &= \prod_{x \in \text{Dom}(\Gamma)} \llbracket \Gamma(x) \rrbracket \xrightarrow{\pi_x} \llbracket \Gamma(x) \rrbracket & \llbracket \star \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\mathbb{1}} \mathbb{1} \\
\llbracket (A_1, A_2) \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \llbracket A_1 \rrbracket, \llbracket A_2 \rrbracket \rangle} \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket \\
\llbracket \mathbf{inj}_i^{A_1 + A_2} V \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\llbracket V \rrbracket} \llbracket A_i \rrbracket \xrightarrow{i} \llbracket A_1 \rrbracket + \llbracket A_2 \rrbracket & \llbracket \mathbf{thunk } M \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\llbracket M \rrbracket} \llbracket \llbracket B \rrbracket \rrbracket \\
\llbracket \mathbf{return } V \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{V} \llbracket A \rrbracket \xrightarrow{\eta} |F \llbracket A \rrbracket| \\
\llbracket M \mathbf{ to } x : A. N \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \text{id}, \llbracket M \rrbracket \rangle} \llbracket \Gamma \rrbracket \times T \llbracket A \rrbracket \xrightarrow{\llbracket N \rrbracket^\dagger} \llbracket \llbracket B \rrbracket \rrbracket \\
\llbracket \lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \} \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \llbracket M_1 \rrbracket, \llbracket M_2 \rrbracket \rangle} \llbracket \llbracket B_1 \rrbracket \rrbracket \times \llbracket \llbracket B_2 \rrbracket \rrbracket \\
\llbracket \mathbf{i} \cdot M \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\llbracket M \rrbracket} \llbracket \llbracket B_1 \rrbracket \rrbracket \times \llbracket \llbracket B_2 \rrbracket \rrbracket \xrightarrow{\pi_i} \llbracket \llbracket B_i \rrbracket \rrbracket & \llbracket \lambda x : A. M \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\lambda[A]. \llbracket M \rrbracket} \llbracket \llbracket B \rrbracket \rrbracket^{[A]} \\
\llbracket V \cdot M \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \llbracket M \rrbracket, \llbracket V \rrbracket \rangle} \llbracket \llbracket B \rrbracket \rrbracket^{[A]} \times \llbracket A \rrbracket \xrightarrow{\text{eval}} \llbracket \llbracket B \rrbracket \rrbracket \\
\llbracket \mathbf{match } V \mathbf{ as } (x_1 : A_1, x_2 : A_2). M \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \text{id}, \llbracket V \rrbracket \rangle} \llbracket \Gamma \rrbracket \times \llbracket A_1 \rrbracket \times \llbracket A_2 \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket \llbracket B \rrbracket \rrbracket \\
\llbracket \mathbf{match } V : \mathbf{0} \mathbf{ as } \{ \}^B \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\llbracket V \rrbracket} 0 \xrightarrow{i} \llbracket \llbracket B \rrbracket \rrbracket \\
\llbracket \mathbf{match } V \mathbf{ as } \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \text{id}, \llbracket V \rrbracket \rangle} \llbracket \Gamma \rrbracket \times (\llbracket A_1 \rrbracket + \llbracket A_2 \rrbracket) \\
&\cong (\llbracket \Gamma \rrbracket \times \llbracket A_1 \rrbracket) + (\llbracket \Gamma \rrbracket \times \llbracket A_2 \rrbracket) \xrightarrow{\llbracket \llbracket M_1 \rrbracket, \llbracket M_2 \rrbracket \rrbracket} \llbracket \llbracket B \rrbracket \rrbracket \\
&\uparrow \\
&\text{distributivity} \\
\llbracket \mathbf{force } V \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\llbracket V \rrbracket} \llbracket \llbracket B \rrbracket \rrbracket \\
\llbracket \mathbf{op}_V^B M \rrbracket : \llbracket \Gamma \rrbracket &\xrightarrow{\langle \text{id}, \llbracket V \rrbracket \rangle} \llbracket \Gamma \rrbracket \times \llbracket P \rrbracket \xrightarrow{\text{id} \times \mathcal{G}[\text{op}]} \llbracket \Gamma \rrbracket \times T \llbracket A \rrbracket \xrightarrow{\llbracket M \rrbracket^\dagger} \llbracket \llbracket B \rrbracket \rrbracket
\end{aligned}$$

Figure 10.5: term interpretation

The type system and the definition of models ensure these denotations are well-defined. Using α -equivalence, we may assume that all differently bound variable names in terms are distinct. Thus, whenever we extend the context, we indeed have:

$$\mathit{CTX} \llbracket \Gamma, x : A \rrbracket \cong \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket$$

In addition, we use the generic model structure to simplify the denotations of effect operation symbols.

The definition of an *algebraic* CBPV Π^{CBPV} -model is straightforward:

Definition 10.5⁺. Let Π^{CBPV} be a CBPV signature. An algebraic Π^{CBPV} -model \mathcal{M} is a

$$\langle \lambda, \mathcal{V}, \mathcal{B} \llbracket - \rrbracket, \mathcal{L}, \mathcal{L} \llbracket - \rrbracket, \mathcal{K} \llbracket - \rrbracket \rangle$$

where:

- λ is a regular cardinal;
- \mathcal{V} is a λ -Power category;
- $\mathcal{B} \llbracket - \rrbracket$ is a basic-type assignment;
- $\langle \lambda, \mathcal{V}, \mathcal{L} \llbracket - \rrbracket, \llbracket \text{type} \rrbracket, \mathcal{L} \llbracket - \rrbracket \rangle$ is an algebraic CBPV Π -model;
- $\mathcal{K} \llbracket - \rrbracket$ assigns to every constant $c \in S$ an arrow

$$\mathcal{K} \llbracket c \rrbracket : \mathbb{1} \rightarrow \llbracket A_c \rrbracket$$

Note that part of this definition requires our choice of basic-type assignment to ensure all operation type assignments are λ -presentable objects. We can ensure this condition if all ground types involved in effect operation type assignments are interpreted as λ -presentable objects. Indeed, as the λ -presentable objects in a λ -Power category are closed under finite products and coproducts, a simple inductive argument shows this condition on the type assignment holds in this case.

10.1.3 Set-theoretic semantics



We now spell out the special case where the semantics is given in terms of sets and functions, i.e., $\mathcal{V} = \mathbf{Set}$. Let Π^{syntax} a CBPV term signature. A *set-theoretic base-type interpretation* is an assignment of a set $\mathcal{B} \llbracket Q \rrbracket$ to every basic type $Q \in \mathbf{Bsc}$.

$$\begin{array}{ll}
\mathcal{V}[\mathbf{Q}] := \mathcal{B}[\mathbf{Q}] & \mathcal{CTX}[\Gamma] := \prod_{x \in \text{Dom}(\Gamma)} [\Gamma(x)] \\
\mathcal{V}[\mathbf{1}] := \mathbb{1} & \\
\mathcal{V}[A_1 \times A_2] := [A_1] \times [A_2] & \\
\mathcal{V}[\mathbf{0}] := \mathbb{0} & \mathcal{C}[\mathbf{FA}] := T[A] \\
\mathcal{V}[A_1 + A_2] := [A_1] + [A_2] & \mathcal{C}[\underline{B}_1 \times \underline{B}_2] := [\underline{B}_1] \times [\underline{B}_2] \\
\mathcal{V}[\underline{UB}] := \mathcal{C}[\underline{B}] & \mathcal{C}[A \rightarrow \underline{B}] := [\underline{B}]^{[A]}
\end{array}$$

Figure 10.4 (revisited): CBPV type interpretation

This assignment extends to a *ground-type interpretation* $\mathcal{G}[-]$, which assigns, to every $G \in \mathbf{Gnd}$ a set:

$$\begin{array}{ll}
\mathcal{G}[\mathbf{Q}] := \mathcal{B}[\mathbf{Q}] & \\
\mathcal{G}[\mathbf{1}] := \mathbb{1} & \mathcal{G}[G_1 \times G_2] := [G_1] \times [G_2] \\
\mathcal{G}[\mathbf{0}] := \mathbb{0} & \mathcal{G}[G_1 + G_2] := [G_1] + [G_2]
\end{array}$$

We will henceforth omit the name of the semantic function if it can be inferred from the context. For example, we may choose $\mathcal{B}[\mathbf{Word}]$ and $\mathcal{B}[\mathbf{Loc}]$ to be 2^{64} in a 64-bit setting, and $\mathcal{B}[\mathbf{Char}]$ to be 2^7 for ASCII characters. As these interpretations are finite sets, every ground type involving them denotes a finite set.

Let T be a monad over \mathbf{Set} . Given a syntax signature Π^{syntax} and a base-type interpretation $\mathcal{G}[-]$, we define a *type interpretation* for kind judgements of types (see Figure 10.4), where each type is interpreted as a set. Note that each interpretation of a computation type $\mathcal{C}[\underline{B}]$ comes equipped with a bind function

$$\ggg : TX \times (\mathcal{C}[\underline{B}])^X \rightarrow \mathcal{C}[\underline{B}]$$

For the types $T[A]$ (i.e., returners), the function \ggg is the usual monadic bind function. For the computation products, given any a in TX and f in $[\underline{B}]^X$, we define:

$$a \ggg f := \langle a \ggg f_1, a \ggg f_2 \rangle$$

where $f_i : X \rightarrow [\underline{B}_i]$ are the two component functions of f , i.e., for all x in X , $f(x)$ consists of the pair $\langle f_1(x), f_2(x) \rangle$. For function types, given any a in TX and α in $([\underline{B}]^Y)^X$, we define:

$$a \ggg \alpha := \lambda y. (a \ggg \lambda x. \alpha(x)(y))$$

Let T be a strong monad over \mathbf{Set} , Π^{CBPV} a CBPV signature, and $\mathcal{B}[-]$ a base-type interpretation. We define a set-theoretic type assignment $\mathcal{R}[\text{type}]$ for Π by assigning $\text{op} : \llbracket A \rrbracket \langle \llbracket P \rrbracket \rangle$ to every $\text{op} \in \Pi$.

With these notions in place, we are ready to define the semantic data required for modelling CBPV with effects using sets and functions:

Definition 10.5 (revisited). Let Π^{CBPV} be a set-theoretic CBPV signature. A set-theoretic Π^{CBPV} -model \mathcal{M} is a quadruple

$$\langle T, \mathcal{B}[-], \mathcal{G}[-], \mathcal{K}[-] \rangle$$

where:

- $\mathcal{B}[-]$ is a basic-type assignment;
- T is a monad over \mathbf{Set} ;
- $\mathcal{G}[-]$ assigns to every $\text{op} : A \langle P \rangle$ in Π a generic effect $\mathcal{G}[\text{op}] : \llbracket P \rrbracket \rightarrow T \llbracket A \rrbracket$;
- $\mathcal{K}[-]$ assigns to every constant $c \in S$ an element $\mathcal{K}[c]$ in $\llbracket A_c \rrbracket$.

Let $\mathcal{M} = \langle T, \mathcal{B}[-], \mathcal{G}[-], \mathcal{K}[-] \rangle$ be a set-theoretic Π^{CBPV} -model. We interpret the CBPV terms as $\Gamma \vdash P : X$ as functions $\llbracket P \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket X \rrbracket$ (see Figure 10.5). The denotations have straightforward definitions. Note our use of the monadic unit, η , and bind functions. We denote the empty function $\emptyset \rightarrow X$ by i_X . Note how the type system and the definition of models ensure these denotations are well-defined.

By replacing the monad with a presentation and the generic effect with an indexed family of terms we obtain the notion of a *presentation Π^{CBPV} -model*. Each presentation model yields a set-theoretic model, and hence can be used to interpret CBPV.

10.2 Multi-adjunctive intermediate languages



We now introduce a type-and-effect refinement of our family of CBPV languages. In our semantics, for each effect set ε , we have a separate CBPV model, all sharing the same values, hence we dub our family of languages as *multi-adjunctive intermediate languages* (MAIL). We follow the structure of the previous section.

$$\begin{aligned}
\mathcal{V}\mathcal{T} \llbracket b \rrbracket (\gamma) &:= \llbracket b \rrbracket & \mathcal{V}\mathcal{T} \llbracket x \rrbracket (\gamma) &:= \pi_x(\gamma) & \mathcal{V}\mathcal{T} \llbracket \star \rrbracket (\gamma) &:= \star \\
\mathcal{V}\mathcal{T} \llbracket (A_1, A_2) \rrbracket (\gamma) &:= \langle \llbracket A_1 \rrbracket (\gamma), \llbracket A_2 \rrbracket (\gamma) \rangle \\
\mathcal{V}\mathcal{T} \llbracket \mathbf{inj}_i^{A_1+A_2} A \rrbracket (\gamma) &:= \iota_i(\llbracket A \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T} \llbracket \mathbf{return} A \rrbracket (\gamma) &:= \eta(\llbracket A \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T} \llbracket M \mathbf{to} x : A. N \rrbracket (\gamma) &:= \llbracket M \rrbracket (\gamma) \ggg (\lambda a. \llbracket N \rrbracket (\gamma[x \mapsto a])) \\
\mathcal{C}\mathcal{T} \llbracket \mathbf{i}' M \rrbracket (\gamma) &:= \pi_i(\llbracket M \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T} \llbracket A' M \rrbracket (\gamma) &:= (\llbracket M \rrbracket (\gamma)) (\llbracket A \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T} \llbracket \mathbf{match} A : \mathbf{0} \mathbf{as} \{ \}^B \rrbracket &:= i_{\llbracket B \rrbracket} \\
\mathcal{C}\mathcal{T} \llbracket \mathbf{match} A \mathbf{as} \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} \rrbracket (\gamma) \\
&:= \begin{cases} \llbracket M_1 \rrbracket (\gamma[x_1 \mapsto a_1]) & \llbracket A \rrbracket (\gamma) = \iota_1 a_1 \\ \llbracket M_2 \rrbracket (\gamma[x_2 \mapsto a_2]) & \llbracket A \rrbracket (\gamma) = \iota_2 a_2 \end{cases} \\
\mathcal{C}\mathcal{T} \llbracket \mathbf{op}_A^B M \rrbracket (\gamma) &:= \mathcal{G}[\mathbf{op}] (\llbracket A \rrbracket (\gamma)) \ggg (\llbracket M \rrbracket (\gamma))
\end{aligned}$$

Figure 10.5 (revisited): CBPV term interpretation

10.2.1 Syntax and type-and-effect system

We begin by refining the syntax parameters:

Definition 10.6. A MAIL syntax signature Σ^{syntax} is a triple $\langle \mathbf{Bsc}, \Sigma, S \rangle$, where:

- \mathbf{Bsc} is a set of basic types, ranged over by Q ;
- Σ is an effect hierarchy (see Definition 3.1); and
- S is a set of built-in constants, ranged over by c .

Note that every MAIL syntax signature Σ^{syntax} induces a CBPV syntax signature $\Sigma_{\mathfrak{h}}^{\text{syntax}}$, by erasing the hierarchy \mathcal{E} . Conversely, every CBPV syntax signature can be regarded as a MAIL syntax signature Π^{syntax} , by taking \mathcal{E} to be the singleton $\{\Pi\}$.

Given a syntax signature Σ^{syntax} , the syntax of MAIL is given by Figure 10.6. It refines the CBPV dichotomy between values and computations. Instead of one kind of

Kinds: $K ::= \text{Val} \mid \text{Comp}_{\mathcal{E}}$

Value

Types: $A, B, \dots ::= Q \mid \mathbf{1} \mid A_1 \times A_2 \mid \mathbf{0} \mid A_1 + A_2 \mid \text{U}_{\mathcal{E}} \underline{B}$

Computation

Types: $\underline{A}, \underline{B}, \dots ::= F_{\mathcal{E}} A \mid \underline{B}_1 \times \underline{B}_2 \mid A \rightarrow \underline{B}$

Ground Value

Types: $G ::= Q \mid \mathbf{1} \mid G_1 \times G_2 \mid \mathbf{0} \mid G_1 + G_2$

Value

Terms: $V ::= b \mid x \mid \star \mid (V_1, V_2) \mid \mathbf{inj}_1^{A_1+A_2} V$
 $\quad \quad \quad \mid \mathbf{inj}_2^{A_1+A_2} V \mid \mathbf{thunk} M$

Computation

Terms: $M, N, \dots ::= \mathbf{coerce}_{\mathcal{E}_1 \subseteq \mathcal{E}_2} M \mid \mathbf{return}_{\mathcal{E}} V \mid M \mathbf{to} x:A. N$
 $\quad \quad \quad \mid \lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \} \mid \mathbf{1}' M \mid \mathbf{2}' M \mid \lambda x : A. M$
 $\quad \quad \quad \mid V' M \mid \mathbf{match} V \mathbf{as} (x_1 : A_1, x_2 : A_2). M \mid \mathbf{match} V : \mathbf{0} \mathbf{as} \{ \}^B$
 $\quad \quad \quad \mid \mathbf{match} V \mathbf{as} \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} \mid \mathbf{force} V \mid \text{op}_V^B M$

Figure 10.6: MAIL syntax

$$\begin{array}{c}
\begin{array}{c}
\frac{\Gamma_k Q : \text{Val} \quad \Gamma_k \mathbf{1} : \text{Val} \quad \frac{\Gamma_k A_1 : \text{Val} \quad \Gamma_k A_2 : \text{Val}}{\Gamma_k A_1 \times A_2 : \text{Val}}}{\Gamma_k \mathbf{0} : \text{Val}} \\
\frac{\Gamma_k A_1 : \text{Val} \quad \Gamma_k A_2 : \text{Val}}{\Gamma_k A_1 + A_2 : \text{Val}} \quad \frac{\Gamma_k \underline{B} : \text{Comp}_\varepsilon}{\Gamma_k U_\varepsilon \underline{B} : \text{Val}} \\
\frac{\Gamma_k A : \text{Val}}{\Gamma_k F_\varepsilon A : \text{Comp}_\varepsilon} \quad \frac{\Gamma_k \underline{B}_1 : \text{Comp}_\varepsilon \quad \Gamma_k \underline{B}_2 : \text{Comp}_\varepsilon}{\Gamma_k \underline{B}_1 \times \underline{B}_2 : \text{Comp}_\varepsilon} \\
\frac{\Gamma_k A : \text{Val} \quad \Gamma_k \underline{B} : \text{Comp}_\varepsilon}{\Gamma_k A \rightarrow \underline{B} : \text{Comp}_\varepsilon}
\end{array}
\end{array}$$

Figure 10.7: MAIL kind system

computation we have ε -computations Comp_ε , for each effect set $\varepsilon \in \mathcal{E}$. These computation may only cause effects in ε . We view MAIL as multiple copies of CBPV, one for each ε , sharing the same values. One can translate between these different CBPVs by means of coercion (see below).

Our types are a slight variation on CBPV types. We modified the CBPV thunk and returner types to thunks $U_\varepsilon \underline{B}$ of ε -computations of type \underline{B} , and returner ε -computations $F_\varepsilon A$ returning a value of type A . Note that for every ground type G we have $G_{\text{h}} = G$.

Many type and effect systems contain a sub-effecting rule: if M is an ε_1 -computation and $\varepsilon_1 \subseteq \varepsilon_2$ then M is an ε_2 -computation. One implication of such a rule is that well-typed terms can have multiple types, and even multiple type derivations. As a consequence, the denotational semantics can no longer be defined directly on terms. Instead, it needs to be defined on the proofs that these terms are well-typed, and then *coherence* results are needed to show that the different semantics are compatible with each other.

This issue is familiar from languages that support *subtyping*. One standard way to circumvent it, followed here, is to use explicit coercion between a subtype and its supertype [TCGS91]. The terms $\mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M$ explicitly coerce ε_1 -computations to ε_2 -computations.

The other terms of MAIL are identical to their CBPV counterparts. In particular, note how erasing the effect annotation and coercions from any MAIL phrase P for some signature Σ^{syntax} yields the corresponding CBPV phrase P_{h} for the signature $\Sigma_{\text{h}}^{\text{syntax}}$.

The kind system for MAIL, given a signature Σ^{syntax} , is displayed in Figure 10.7. It is mostly straightforward: the computational product is only well-kinded for two computations of the *same* kind.

As for CBPV, we denote by **Val** the set of well-kinded value types $\{A \mid \vdash_k A : \mathbf{Val}\}$. Similarly, we write **Comp**(ε) for the set of well-kinded ε -computation types. Well-kinded contexts are defined as for CBPV.

We define arity assignments for a syntax signature as for CBPV, i.e., $\text{type} : \Pi \rightarrow \mathbf{Gnd} \times \mathbf{Gnd}$. As the ground types of MAIL are the same as CBPV, there is no need to distinguish between an arity assignment and its erasure. Given a syntax signature Σ^{syntax} , a constant type assignment is still just a function $A_- : S \rightarrow \mathbf{Val}$. Note, however, that the types of MAIL are richer than those of CBPV, so we can have constants such as:

$$\begin{aligned} ++ & : U_{\emptyset}((\mathbf{Str} \times \mathbf{Str}) \rightarrow F_{\emptyset} \mathbf{Str}) \\ + & : U_{\{\text{ArithmeticOverflow}\}}(\mathbf{Word} \times \mathbf{Word} \rightarrow F_{\{\text{ArithmeticOverflow}\}} \mathbf{Word}) \end{aligned}$$

We define type level MAIL signatures, and complete the parametrisation of our CBPV language:

Definition 10.7. A MAIL signature is a triple $\Sigma^{\text{MAIL}} = \langle \Sigma^{\text{syntax}}, \text{type}, A_- \rangle$ where:

- Σ^{syntax} is a MAIL syntax signature;
- type is an arity assignment for Σ^{syntax} ; and
- $A_- : S \rightarrow \mathbf{Val}$ is a constant type assignment for Σ^{syntax} .

The type system of MAIL, given a signature Π^{CBPV} , is displayed in Figure 10.8. It is similar to the type system of Figure 10.3, except that the single computation type judgement relation $\Gamma \vdash_c M : \underline{B}$ is replaced by a collection of computation type judgement relations $\Gamma \vdash_{\varepsilon} M : \underline{B}$, indexed by $\varepsilon \in \mathcal{E}$, where Γ, A, \underline{B} are well-kinded contexts, value types and ε -computation kinds, respectively, and where M is a computation term. As for CBPV, we call closed ground returners $\vdash_{\varepsilon} P : F_{\varepsilon} G$ programs.

The type system is similar to its CBPV counterpart, apart from the rules for coercion and effect operation symbols. The coercion rule may seem surprising — we only allow coercion of returners. However this restriction, which is semantically natural, allows sufficient generality. For if $\varepsilon \subseteq \varepsilon'$, then for any $\underline{B} : \text{Comp}_{\varepsilon}$ one can inductively define a type $\underline{B}' : \text{Comp}_{\varepsilon'}$ by: $(F_{\varepsilon} A)' = F_{\varepsilon'} A$, $(\underline{B}_1 \times \underline{B}_2)' = (\underline{B}_1)' \times (\underline{B}_2)'$, and $(A \rightarrow \underline{B})' = A \rightarrow \underline{B}'$; there is then an evident coercion from \underline{B} to \underline{B}' .

$$\begin{array}{c}
\Gamma \vdash_v c : A_c \quad \frac{\Gamma(x) = A}{\Gamma \vdash_v x : A} \quad \Gamma \vdash_v \star : \mathbf{1} \quad \frac{\Gamma \vdash_v V_1 : A_1 \quad \Gamma \vdash_v V_2 : A_2}{\Gamma \vdash_v (V_1, V_2) : A_1 \times A_2} \\
\\
\frac{\Gamma \vdash_v V : A_i}{\Gamma \vdash_v \mathbf{inj}_i^{A_1 + A_2} V : A_1 + A_2} \quad (\mathbf{i} = \mathbf{1}, \mathbf{2}) \quad \frac{\Gamma \vdash_\varepsilon M : \underline{B}}{\Gamma \vdash_v \mathbf{thunk} M : U_\varepsilon \underline{B}} \\
\\
\frac{\Gamma \vdash_{\varepsilon_1} M : F_{\varepsilon_1} A}{\Gamma \vdash_{\varepsilon_2} \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M : F_{\varepsilon_2} A} \quad \frac{\Gamma \vdash_v V : A}{\Gamma \vdash_\varepsilon \mathbf{return}_\varepsilon V : F_\varepsilon A} \\
\\
\frac{\Gamma \vdash_\varepsilon M : F_\varepsilon A \quad \Gamma, x : A \vdash_\varepsilon N : \underline{B}}{\Gamma \vdash_\varepsilon M \mathbf{to} x : A. N : \underline{B}} \\
\\
\frac{\Gamma \vdash_\varepsilon M_1 : \underline{B}_1 \quad \Gamma \vdash_\varepsilon M_2 : \underline{B}_2}{\Gamma \vdash_\varepsilon \lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \} : \underline{B}_1 \times \underline{B}_2} \quad \frac{\Gamma \vdash_\varepsilon M : \underline{B}_1 \times \underline{B}_2}{\Gamma \vdash_\varepsilon \mathbf{i}' M : \underline{B}_i} \quad (\mathbf{i} = \mathbf{1}, \mathbf{2}) \\
\\
\frac{\Gamma, x : A \vdash_\varepsilon M : \underline{B}}{\Gamma \vdash_\varepsilon \lambda x : A. M : A \rightarrow \underline{B}} \quad \frac{\Gamma \vdash_v V : A \quad \Gamma \vdash_\varepsilon M : A \rightarrow \underline{B}}{\Gamma \vdash_\varepsilon V' M : \underline{B}} \\
\\
\frac{\Gamma \vdash_v V : A_1 \times A_2 \quad \Gamma, x_1 : A_1, y : A_2 \vdash_\varepsilon M : \underline{B}}{\Gamma \vdash_\varepsilon \mathbf{match} V \mathbf{as} (x_1 : A_1, x_2 : A_2). M : \underline{B}} \quad (x_1 \neq x_2) \\
\\
\frac{\Gamma \vdash_v V : \mathbf{0}}{\Gamma \vdash_\varepsilon \mathbf{match} V : \mathbf{0} \mathbf{as} \{ \}^{\underline{B}} : \underline{B}} \\
\\
\frac{\Gamma \vdash_v V : A_1 + A_2 \quad \Gamma, x_1 : A_1 \vdash_\varepsilon M_1 : \underline{B} \quad \Gamma, x_2 : A_2 \vdash_\varepsilon M_2 : \underline{B}}{\Gamma \vdash_\varepsilon \mathbf{match} V \mathbf{as} \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} : \underline{B}} \\
\\
\frac{\Gamma \vdash_v V : U_\varepsilon \underline{B}}{\Gamma \vdash_\varepsilon \mathbf{force} V : \underline{B}} \\
\\
\frac{\Gamma \vdash_\varepsilon V : P \quad \Gamma \vdash_\varepsilon M : A \rightarrow \underline{B}}{\Gamma \vdash_\varepsilon \mathbf{op}_V^B M : \underline{B}} \quad (\mathbf{op} : A \langle P \rangle, \mathbf{op} \in \varepsilon)
\end{array}$$

Figure 10.8: MAIL type system

We note too that, from a CBPV perspective, call-by-value types are always translated into returners $F_{\varepsilon}A$ [Lev04], hence the coercion rule seems sufficient to subsume existing coercion effect systems. We do not know of any call-by-name effect systems, and we conjecture that by following Levy's call-by-name translation, we could produce a call-by-name effect system.

Also note that in the typing rule for effect operations, only operations from the current effect set ε may be used.

Theorem 10.8. *Every well-kinded type has a unique kind, and every well-typed term has a unique type.*

Proof

Standard induction. ■

We define capture avoiding substitution in the usual manner. We will only substitute value terms for variables. The variable binders are the following constructs:

- $M \text{ to } x : A. N$ binds x in N ;
- $\lambda x : A. M$ binds x in M ;
- **match** V **as** $(x_1 : A_1, x_2 : A_2). M$ binds x_1 and x_2 in M ; and
- **match** V **as** $\{\text{inj}_1 x_1 : A_1. M_1, \text{inj}_2 x_2 : A_2. M_2\}$ binds x_1 in M_1 and x_2 in M_2 .

Lemma 10.9 (substitution). *For every well-typed phrase (value term or computation term) $\Gamma \vdash P : X$, for all well-kinded contexts Δ and $\text{Dom}(\Gamma)$ -indexed family of values V_- , satisfying, for all $x \in \text{Dom}(\Gamma)$, $\Delta \vdash V_x : \Gamma(x)$, we have $\Delta \vdash P[V_x/x]_{x \in \text{Dom}(\Gamma)}$.*

Proof

Standard induction. ■

We extend the erasure function to kind and type judgements and their derivations.

Theorem 10.10. *Erasure maps well-kinded types to well-kinded types, well-typed terms to well-typed terms, and well-formed derivations to well-formed derivations.*

Proof

Straightforward induction. ■

$$\begin{array}{ll}
\mathcal{V}[\mathcal{Q}] := \mathcal{B}[\mathcal{Q}] & \mathcal{CTX}[\Gamma] := \prod_{x \in \text{Dom}(\Gamma)} [\Gamma(x)] \\
\mathcal{V}[\mathbf{1}] := \mathbf{1} & \\
\mathcal{V}[A_1 \times A_2] := [A_1] \times [A_2] & \\
\mathcal{V}[\mathbf{0}] := \mathbf{0} & \mathcal{C}_\varepsilon[F_\varepsilon A] := F[A] \\
\mathcal{V}[A_1 + A_2] := [A_1] + [A_2] & \mathcal{C}_\varepsilon[\underline{B}_1 \times \underline{B}_2] := [\underline{B}_1] \times [\underline{B}_2] \\
\mathcal{V}[U_\varepsilon \underline{B}] := |[[\underline{B}]]| & \mathcal{C}_\varepsilon[A \rightarrow \underline{B}] := [\underline{B}]^{[A]}
\end{array}$$

Figure 10.9: MAIL type interpretation

10.2.2 Categorical semantics



The denotational semantics of MAIL follows the same lines of the CBPV semantics. First we define a *base-type interpretation* $\mathcal{B}[-]$ for a syntax signature Π^{syntax} in a distributive category \mathcal{V} . Such assignments extend to *ground-type assignments* using the distributive structure. Given any functor $\mathbb{P} : \mathcal{E} \rightarrow \text{CBPV}_{\mathcal{V}}$ (cf. Definition 2.12), we have, for every ε , a strong monad T_ε . Thus, given such a \mathbb{P} , base-type assignments also extend to *type interpretations* of kind judgements (see Figure 10.9):

- value types are interpreted as \mathcal{V} -objects via $\mathcal{V}[-]$, as in CBPV;
- ε -computation types are interpreted as $\mathcal{V}^{T_\varepsilon}$ -objects, i.e., algebras, via $\mathcal{C}_\varepsilon[-]$; and
- contexts as finite products in \mathcal{V} via $\mathcal{CTX}[-]$, as in CBPV.

This interpretation treats MAIL types as a family of interpretations of CBPV types, one in each computation category \mathcal{V}^T , sharing the same value category \mathcal{V} . Note how the product of two computations is only well-defined when they are of the same kind. We define interpretations of type assignments $\mathcal{R}[\text{type}]$ given a basic-type assignment for a MAIL signature as for CBPV.

With these notions in place, we are ready to define the semantic data required for modelling MAIL:

Definition 10.11. Let Σ^{MAIL} be a MAIL signature. A Σ^{MAIL} -model \mathcal{M} is a quintuple

$$\langle \mathcal{V}, \mathbb{P}, \mathcal{B}[-], \mathcal{O}_-[-], \mathcal{K}[-] \rangle$$

where:

$$\begin{aligned}
\llbracket c \rrbracket : [\Gamma] &\xrightarrow{\dagger} \mathbb{1} \xrightarrow{\mathcal{K}[\llbracket c \rrbracket]} [A_c] & \llbracket x \rrbracket : [\Gamma] &= \prod_{x \in \text{Dom}(\Gamma)} [\Gamma(x)] \xrightarrow{\pi_x} [\Gamma(x)] & \llbracket \star \rrbracket : [\Gamma] &\xrightarrow{\dagger} \mathbb{1} \\
\llbracket (A_1, A_2) \rrbracket : [\Gamma] &\xrightarrow{\langle \llbracket A_1 \rrbracket, \llbracket A_2 \rrbracket \rangle} [A_1] \times [A_2] \\
\llbracket \mathbf{inj}_i^{A_1+A_2} V \rrbracket : [\Gamma] &\xrightarrow{\llbracket V \rrbracket} [A_i] \xrightarrow{\iota_i} [A_1] + [A_2] & \llbracket \mathbf{thunk} M \rrbracket : [\Gamma] &\xrightarrow{\llbracket M \rrbracket} \llbracket [B] \rrbracket \\
\llbracket \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M \rrbracket : [\Gamma] &\xrightarrow{\llbracket M \rrbracket} T_{\varepsilon_1} [A] \xrightarrow{m_{\varepsilon_1 \subseteq \varepsilon_2}} T_{\varepsilon_2} [A] \\
\llbracket \mathbf{return}_\varepsilon V \rrbracket : [\Gamma] &\xrightarrow{V} [A] \xrightarrow{\eta_\varepsilon} |F [A]| \\
\llbracket M \mathbf{to} x : A. N \rrbracket : [\Gamma] &\xrightarrow{\langle \text{id}, \llbracket M \rrbracket \rangle} [\Gamma] \times T [A] \xrightarrow{\llbracket N \rrbracket^\dagger} \llbracket [B] \rrbracket \\
\llbracket \lambda \{ \mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2 \} \rrbracket : [\Gamma] &\xrightarrow{\langle \llbracket M_1 \rrbracket, \llbracket M_2 \rrbracket \rangle} \llbracket [B_1] \rrbracket \times \llbracket [B_2] \rrbracket \\
\llbracket \mathbf{i} \cdot M \rrbracket : [\Gamma] &\xrightarrow{\llbracket M \rrbracket} \llbracket [B_1] \rrbracket \times \llbracket [B_2] \rrbracket \xrightarrow{\pi_i} \llbracket [B_i] \rrbracket & \llbracket \lambda x : A. M \rrbracket : [\Gamma] &\xrightarrow{\lambda[A]. \llbracket M \rrbracket} \llbracket [B] \rrbracket^{[A]} \\
\llbracket V \cdot M \rrbracket : [\Gamma] &\xrightarrow{\langle \llbracket M \rrbracket, \llbracket V \rrbracket \rangle} \llbracket [B] \rrbracket^{[A]} \times [A] \xrightarrow{\text{eval}} \llbracket [B] \rrbracket \\
\llbracket \mathbf{match} V \mathbf{as} (x_1 : A_1, x_2 : A_2). M \rrbracket : [\Gamma] &\xrightarrow{\langle \text{id}, \llbracket V \rrbracket \rangle} [\Gamma] \times [A_1] \times [A_2] \xrightarrow{\llbracket M \rrbracket} \llbracket [B] \rrbracket \\
\llbracket \mathbf{match} V : \mathbf{0} \mathbf{as} \{ \}^B \rrbracket : [\Gamma] &\xrightarrow{\llbracket V \rrbracket} 0 \xrightarrow{\dagger} \llbracket [B] \rrbracket \\
\llbracket \mathbf{match} V \mathbf{as} \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} \rrbracket : [\Gamma] &\xrightarrow{\langle \text{id}, \llbracket V \rrbracket \rangle} [\Gamma] \times ([A_1] + [A_2]) \\
&\cong ([\Gamma] \times [A_1]) + ([\Gamma] \times [A_2]) \xrightarrow{\llbracket [M_1] \rrbracket, \llbracket [M_2] \rrbracket} \llbracket [B] \rrbracket \\
&\uparrow \\
&\text{distributivity} \\
\llbracket \mathbf{force} V \rrbracket : [\Gamma] &\xrightarrow{\llbracket V \rrbracket} \llbracket [B] \rrbracket \\
\llbracket \mathbf{op}_V^B M \rrbracket : [\Gamma] &\xrightarrow{\langle \text{id}, \llbracket V \rrbracket \rangle} [\Gamma] \times [P] \xrightarrow{\text{id} \times \mathcal{G}[\text{op}]} [\Gamma] \times T [A] \xrightarrow{\llbracket M \rrbracket^\dagger} \llbracket [B] \rrbracket
\end{aligned}$$

Figure 10.10: MAIL term interpretation

- \mathcal{V} is a distributive category;
- $\mathcal{B}[-]$ is a basic-type assignment;
- $\langle \mathcal{V}, [\text{type}], \mathbb{P}, \mathcal{O}[-] \rangle$ is a Σ -model (cf. Definition 3.2);
- $\mathcal{K}[-]$ assigns to every constant $c \in S$ an arrow

$$\mathcal{K}[c] : \mathbb{1} \rightarrow [A_c]$$

Let $\langle \mathcal{V}, T, \mathbb{P}, \mathcal{B}[-], \mathcal{O}[-], \mathcal{K}[-] \rangle$ be a Σ^{MAIL} -model. Denote by $F_\varepsilon \dashv |-|$ the Eilenberg-Moore resolution of T_ε i.e., of the CBPV model $\mathbb{P}\varepsilon$. Denote by $m_{\varepsilon_1 \subseteq \varepsilon_2}$ the monad morphism $\mathbb{P}(\varepsilon_1 \subseteq \varepsilon_2)$. We interpret MAIL terms as follows (see Figure 10.5):

- value terms $\Gamma \vdash_{\mathcal{V}} V : A$ are interpreted as \mathcal{V} -morphisms

$$\mathcal{V}\mathcal{T}[V] : \mathcal{C}\mathcal{T}\mathcal{X}[\Gamma] \rightarrow \mathcal{V}[A]$$

- computation terms $\Gamma \vdash_{\varepsilon} M : \underline{B}$ are interpreted as \mathcal{V} -morphisms

$$\mathcal{C}\mathcal{T}_{\varepsilon}[M] : \mathcal{C}\mathcal{T}\mathcal{X}[\Gamma] \rightarrow |\mathcal{C}_{\varepsilon}[\underline{B}]|$$

Note that dropping the semantic function names from our denotations makes them identical to that of CBPV. As a consequence, we will see in Section 11.2 that MAIL inherits the equational theory of CBPV.

Recall that we chose our morphisms between CBPV to be monad morphisms. However, a different choice of morphisms between CBPV models may allow us to interpret richer constructs. For example, we can change our notion of morphism from T to T' to be an adjunction $F_{T \rightarrow T'} \dashv U_{T' \leftarrow T} : \mathcal{V}^{T'} \rightarrow \mathcal{V}^T$ that commute with the Eilenberg-Moore adjunctions. Such adjunctions certainly exist in the algebraic cases when $\varepsilon \subseteq \varepsilon'$. In this case, we can interpret a form of coercion at all computation types:

$$\frac{\Gamma \vdash_{\varepsilon} M : \underline{B}}{\Gamma \vdash_{\varepsilon'} \mathbf{return}_{\varepsilon \subseteq \varepsilon'} M : F_{\varepsilon \subseteq \varepsilon'} \underline{B}}$$

We can also interpret ‘partial thinking’ of effects:

$$\frac{\Gamma \vdash_{\varepsilon'} M : \underline{B}}{\Gamma \vdash_{\varepsilon} \mathbf{think}_{\varepsilon \subseteq \varepsilon'} M : U_{\varepsilon \subseteq \varepsilon'} \underline{B}}$$

However, we do not know any satisfactory computational understanding of for these constructs. Therefore we chose monad morphisms as our notion of morphism, as these lead to coercion of returners only, which have a computational interpretation as subtypes.

We omit the definition of an *algebraic* MAIL model as it is straightforward.

10.2.3 Relational semantics



We now study relational semantics for MAIL, taking advantage of our categorical formulation of the semantics. Let $\mathcal{B}[-]$ be a base-type interpretation in the category of logical relations **LogRel**, and $\mathcal{B}^1[-]$, $\mathcal{B}^2[-]$ be set-theoretic base-type interpretations. We say that $\mathcal{B}[-]$ is a lifting of $\mathcal{B}^1[-]$, $\mathcal{B}^2[-]$, when, for every basic type $Q \in \mathbf{Bsc}$, $\mathcal{B}[Q]$ is a lifting of $\mathcal{B}^1[Q]$, $\mathcal{B}^2[Q]$. Let $\mathbb{P} : \mathcal{E} \rightarrow \mathbf{CBPV}_{\mathbf{LogRel}}$, $\mathbb{P}_1, \mathbb{P}_2 : \mathcal{E} \rightarrow \mathbf{CBPV}_{\mathbf{Set}}$ be functors. We say that \mathbb{P} is a lifting of $\mathbb{P}_1, \mathbb{P}_2$ if it is a component-wise lifting, i.e., if, for every object or morphism x in \mathcal{E} , $\mathbb{P}x$ is a lifting of $\mathbb{P}_1x, \mathbb{P}_2x$.

Recall that $\mathcal{R}[\text{type}]$ assigns to every $\text{op} : A \langle P \rangle$ the type $[[A]] \langle [[P]] \rangle$.

Lemma 10.12 (basic lemma for types). *Let Σ^{MAIL} be a MAIL signature, and $\mathcal{B}[-]$ a lifting of base-type interpretations $\mathcal{B}^1[-]$, $\mathcal{B}^2[-]$.*

- *For every ground type $G \in \mathbf{Gnd}$, the relational ground type assignment $\mathcal{G}[[G]]$ is a lifting of $\mathcal{G}^1[[G]]$, $\mathcal{G}^2[[G]]$.*
- *The relational type assignment $\mathcal{R}[\text{type}]$ is a lifting of the set-theoretic type assignments $\mathcal{R}^1[\text{type}]$, $\mathcal{R}^2[\text{type}]$.*

Assume further a given lifting $\mathbb{P} : \mathcal{E} \rightarrow \mathbf{CBPV}_{\mathbf{LogRel}}$ of $\mathbb{P}_1, \mathbb{P}_2 : \mathcal{E} \rightarrow \mathbf{CBPV}_{\mathbf{Set}}$.

- *For every $\vdash_k A : \text{Val}$, $\mathcal{V}[[A]]$ is a lifting of $\mathcal{V}^1[[A]]$, $\mathcal{V}^2[[A]]$.*
- *For every $\vdash_\varepsilon \underline{B} : \text{Comp}_\varepsilon$, $\mathcal{C}_\varepsilon[[\underline{B}]]$ is a lifting of $\mathcal{C}_\varepsilon^1[[\underline{B}]]$, $\mathcal{C}_\varepsilon^2[[\underline{B}]]$.*
- *For every well-kinded context Γ , $\mathcal{CTX}[[\Gamma]]$ is a lifting of $\mathcal{CTX}^1[[\Gamma]]$, $\mathcal{CTX}^2[[\Gamma]]$.*

Proof

By induction on the various syntactic classes.

The statement for ground types holds, as ground types are interpreted using the base-type interpretation, and finite products and coproducts, which all lift. Component-wise verification validates that the type assignments lift.

For value types, we use, in addition, the fact that if \underline{B} is a T -algebra in **LogRel** lifting two algebra $\underline{B}_1, \underline{B}_2$, then its carrier set is a lifting for the two corresponding carrier sets.

We turn to computation types. For returner types, assume that for some $\vdash_k F_\varepsilon A : \text{Val}$, $\mathcal{V}[A]$ is a lifting of $\mathcal{V}^1[A]$, $\mathcal{V}^2[A]$. We have:

$$\begin{aligned} C_\varepsilon[F_\varepsilon A] &= \langle \mathbb{P}\varepsilon(\mathcal{V}[A]), \mu \rangle \\ C_\varepsilon^i[F_\varepsilon A] &= \langle \mathbb{P}_i\varepsilon(\mathcal{V}^i[A]), \mu_i \rangle \end{aligned}$$

Because $\mathbb{P}\varepsilon$ is a lifting of $\mathbb{P}_1, \mathbb{P}_2$, we can conclude that $C_\varepsilon[F_\varepsilon A]$ is a lifting of $C_\varepsilon^1[F_\varepsilon A]$, $C_\varepsilon^2[F_\varepsilon A]$. A similar argument shows the induction hypothesis also holds for computation products and function types, as the algebra structure on these is given using the cartesian closed structure and the monadic structure, which all lift.

The statement about contexts holds for the same reasons. ■

Note that we did not use the assumption that the morphism map of \mathbb{P} is a lifting.

Consider any three Σ^{MAIL} -models as follows:

$$\begin{aligned} \mathcal{M} &= \langle \mathbf{LogRel}, \mathbb{P}, \mathcal{B}[-], O_-[-], \mathcal{K}[-] \rangle \\ \mathcal{M}_1 &= \langle \mathbf{Set}, \mathbb{P}_1, \mathcal{B}^1[-], O_-^1[-], \mathcal{K}^1[-] \rangle \\ \mathcal{M}_2 &= \langle \mathbf{Set}, \mathbb{P}_2, \mathcal{B}^2[-], O_-^2[-], \mathcal{K}^2[-] \rangle \end{aligned}$$

We say that \mathcal{M} is a lifting of $\mathcal{M}_1, \mathcal{M}_2$ if:

- $\mathcal{B}[-]$ is a lifting of $\mathcal{B}^1[-], \mathcal{B}^2[-]$;
- the Σ -model $\langle \mathbf{LogRel}, \mathcal{R}[\text{type}], \mathbb{P}, O_-[-] \rangle$ is a lifting of the two Σ -models $\langle \mathbf{Set}, \mathcal{R}^1[\text{type}], \mathbb{P}_1, O_-^1[-] \rangle, \langle \mathbf{Set}, \mathcal{R}^2[\text{type}], \mathbb{P}_2, O_-^2[-] \rangle$;
- for every constant $c \in S$, $\mathcal{K}[c] : \mathbb{1} \rightarrow \mathcal{V}[A_c]$ is a lifting of $\mathcal{K}^1[c], \mathcal{K}^2[c]$.

Lemma 10.13 (basic lemma for terms). *Let Σ^{MAIL} be a MAIL signature, and \mathcal{M} a lifting of $\mathcal{M}_1, \mathcal{M}_2$.*

- For every $\Gamma \vdash_v V : A$, $\mathcal{V}\mathcal{T}[V]$ is a lifting of $\mathcal{V}\mathcal{T}^1[V], \mathcal{V}\mathcal{T}^2[V]$.
- For every $\Gamma \vdash_\varepsilon M : \underline{B}$, $C\mathcal{T}_\varepsilon[M]$ is a lifting of $C\mathcal{T}^1[M], C\mathcal{T}^2[M]$.

Note how the basic lemma for types ensures these arrows are well-typed.

Proof


Straightforward induction over value and computation terms, as our assumptions imply that all the structure involved in the denotational semantics of terms lifts. The only two liftings we have not established yet are the distributivity isomorphism used to define pattern-matching, and the generic effect interpretation. The distributivity isomorphism

$$\begin{array}{ll}
\mathcal{V}[\mathcal{Q}] := \mathcal{B}[\mathcal{Q}] & \mathcal{CTX}[\Gamma] := \prod_{x \in \text{Dom}(\Gamma)} [\Gamma(x)] \\
\mathcal{V}[\mathbf{1}] := \mathbb{1} & \\
\mathcal{V}[A_1 \times A_2] := [A_1] \times [A_2] & \\
\mathcal{V}[\mathbf{0}] := 0 & C_{\varepsilon}[F_{\varepsilon}A] := F[A] \\
\mathcal{V}[A_1 + A_2] := [A_1] + [A_2] & C_{\varepsilon}[\underline{B}_1 \times \underline{B}_2] := [\underline{B}_1] \times [\underline{B}_2] \\
\mathcal{V}[U_{\varepsilon}\underline{B}] := |[[\underline{B}]]| & C_{\varepsilon}[A \rightarrow \underline{B}] := [\underline{B}]^{[A]}
\end{array}$$

Figure 10.9 (revisited): MAIL type interpretation

lifts because **LogRel** is distributive and its products and sums are liftings. For every $\text{op} \in \Pi$, the generic effect $G_{\varepsilon}[\text{op}]$ is expressible using $O_{\varepsilon}[\text{op}]$, the monadic structure, and the cartesian closed structure (see Theorem 2.4). ■

10.2.4 Set-theoretic semantics

 We again specialise to sets and functions. The set-theoretic denotational semantics of MAIL follow the same lines of the CBPV semantics. First we define a *set-theoretic base-type interpretation* $\mathcal{B}[-]$ for a syntax signature Π^{syntax} . Such assignments extend to *ground-type assignments* using products and sums. Given any family of monads T_- indexed by \mathcal{E} , such assignments also extend to *type interpretations* of kind judgements as sets (see Figure 10.9 (revisited) on page 240). This interpretation treats MAIL types as a family of interpretations of CBPV types, one for each effect set, sharing the same interpretations for values. We define interpretations of type assignments $[\text{type}]$ given a basic-type assignment for a MAIL signature as for CBPV.

With these notions in place, we are ready to define the semantic data required for modelling MAIL using sets and functions:

Definition 10.11 (revisited). Let Σ^{MAIL} be a MAIL signature. A set-theoretic Σ^{MAIL} -model \mathcal{M} is a quintuple

$$\langle \mathcal{B}[-], T_-, m_-, \mathcal{G}[-], \mathcal{K}[-] \rangle$$

where:

- $\mathcal{B}[-]$ is a basic-type assignment;

$$\begin{aligned}
\mathcal{V}\mathcal{T} \llbracket b \rrbracket (\gamma) &:= \llbracket b \rrbracket & \mathcal{V}\mathcal{T} \llbracket x \rrbracket (\gamma) &:= \pi_x(\gamma) & \mathcal{V}\mathcal{T} \llbracket \star \rrbracket (\gamma) &:= \star \\
\mathcal{V}\mathcal{T} \llbracket (A_1, A_2) \rrbracket (\gamma) &:= \langle \llbracket A_1 \rrbracket (\gamma), \llbracket A_2 \rrbracket (\gamma) \rangle \\
\mathcal{V}\mathcal{T} \llbracket \mathbf{inj}_i^{A_1+A_2} A \rrbracket (\gamma) &:= \iota_i(\llbracket A \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T}_{\varepsilon_2} \llbracket \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M \rrbracket (\gamma) &:= m_{\varepsilon_1 \subseteq \varepsilon_2}(\llbracket M \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket \mathbf{return}_{\varepsilon} A \rrbracket (\gamma) &:= \eta_{\varepsilon}(\llbracket A \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket M \mathbf{to} x : A. N \rrbracket (\gamma) &:= \llbracket M \rrbracket (\gamma) \ggg (\lambda a. \llbracket N \rrbracket (\gamma[x \mapsto a])) \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket \mathbf{i}' M \rrbracket (\gamma) &:= \pi_i(\llbracket M \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket A' M \rrbracket (\gamma) &:= (\llbracket M \rrbracket (\gamma)) (\llbracket A \rrbracket (\gamma)) \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket \mathbf{match} A : \mathbf{0} \mathbf{as} \{ \}^B \rrbracket &:= i_{\llbracket B \rrbracket} \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket \mathbf{match} A \mathbf{as} \{ \mathbf{inj}_1 x_1 : A_1. M_1, \mathbf{inj}_2 x_2 : A_2. M_2 \} \rrbracket (\gamma) \\
&:= \begin{cases} \llbracket M_1 \rrbracket (\gamma[x_1 \mapsto a_1]) & \llbracket A \rrbracket (\gamma) = \iota_1 a_1 \\ \llbracket M_2 \rrbracket (\gamma[x_2 \mapsto a_2]) & \llbracket A \rrbracket (\gamma) = \iota_2 a_2 \end{cases} \\
\mathcal{C}\mathcal{T}_{\varepsilon} \llbracket \mathbf{op}_A^B M \rrbracket (\gamma) &:= \mathcal{G}_{\varepsilon}[\mathbf{op}] (\llbracket A \rrbracket (\gamma)) \ggg (\llbracket M \rrbracket (\gamma))
\end{aligned}$$

Figure 10.10 (revisited): MAIL term interpretation

- $\langle \llbracket \text{type} \rrbracket, T_{-}, m_{-}, \mathcal{G}[\llbracket - \rrbracket] \rangle$ is a Σ -model; and
- $\mathcal{K}[\llbracket - \rrbracket]$ assigns to every constant $c \in S$ an element $\mathcal{K}[\llbracket c \rrbracket]$ in $\llbracket A_c \rrbracket$.

Let $\langle \mathcal{B}[\llbracket - \rrbracket], T_{-}, m_{-}, \mathcal{G}[\llbracket - \rrbracket], \mathcal{K}[\llbracket - \rrbracket] \rangle$ be a Σ^{MAIL} -model. We interpret MAIL terms $\Gamma \vdash P : X$ as functions $\llbracket P \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket X \rrbracket$ (see Figure 10.5 (revisited) on page 241). Note that dropping the semantic function names from our denotations makes them identical to that of CBPV. As a consequence, we will see in Section 11.2 that MAIL inherits the equational theory of CBPV.

By replacing the Σ -model with a presentation Σ -model we obtain the notion of a *presentation Σ^{MAIL} -model*. Each presentation model yields a set-theoretic model, and hence can be used to interpret MAIL.

10.3 Model generation

Our goal is to generate most or all of the structure needed by MAIL models from a given CBPV model. Most of the work has been done in previous chapters, but the choice of built-in constants complicates matters slightly.



One case in which no complication arises is for the *benchmark models*, defined in Example 7-2. Because these models ignore the effect annotations, the CBPV semantics and the MAIL semantics agree on the nose.

Lemma 10.14. *Let Π^{CBPV} be a CBPV signature, $\mathcal{M} = \langle \mathcal{V}, \mathcal{B}[-], T, O[-], \mathcal{K}[-] \rangle$ a CBPV Π^{CBPV} -model, and Σ an effect hierarchy. Let $\langle \mathcal{V}, [\text{type}], \mathbb{P}_b, O_b^[-] \rangle$ be the benchmark model corresponding to the CBPV model $\langle \mathcal{V}, [\text{type}], T, O[-] \rangle$ (see Example 7-2).*

The benchmark MAIL model \mathcal{M}_b is given by

$$\langle \mathcal{V}, \mathcal{B}[-], \mathbb{P}_b, O_b^[-], \mathcal{K}[-] \rangle$$

and for every MAIL type, context, and term X , we have $\mathcal{M}_b[X] = \mathcal{M}[X_b]$.

Proof

Straightforward calculation. ■

We can always define benchmark models. However, they completely ignore effect annotations. We are interested in obtaining a similar construction for the *conservative restriction models* (see Theorem 7.12). However, as the built-in constants may have arbitrary types and involve arbitrary effects, we do not have a canonical choice of their types and interpretations.

For example, the arithmetic addition constant in CBPV:

$$+ : \mathbf{U}(\mathbf{Word} \times \mathbf{Word} \rightarrow \mathbf{F Word})$$

can have an effect-dependent type in MAIL:

$$+ : \mathbf{U}_{\{\text{ArithmeticOverflow}\}}(\mathbf{Word} \times \mathbf{Word} \rightarrow \mathbf{F}_{\{\text{ArithmeticOverflow}\}} \mathbf{Word})$$

The situation becomes more complicated with higher-order built-in constants, due to contravariance.

We have no general solution to this problem. However, the amount of work involved in choosing the effect annotations and interpretations for the built-in constants is linear in the number of the built-in constants. Therefore, we foresee no problems in

leaving them unspecified by our general account, and manually choosing the appropriate effect annotation and interpretation in any concrete case. We formulate our results to be of use in this general case.

Nevertheless, under some simplifying assumptions on the MAIL and CBPV signatures, we can guarantee a suitable choice of types and interpretations for the built-in constants. We will describe the simplest such method, by restricting to a sub-class of signatures, which we call *simple signatures*.

Definition 10.15. A simple MAIL signature is a MAIL signature Σ^{MAIL} such that $\Pi \in \mathcal{E}$, and for each built-in constant c , the only effect set appearing in A_c is Π .

Thus, a simple signature may include number and string literals $1, 0x\text{FEED} : \mathbf{Word}$, and boolean operations $=, >, < : U_{\Pi}(\mathbf{Word} \times \mathbf{Word} \rightarrow F_{\Pi}\mathbf{2})$ but not a pure string concatenation function $++ : U_{\emptyset}(\mathbf{Str} \times \mathbf{Str} \rightarrow F_{\emptyset}\mathbf{Str})$.

Our simplifying assumptions are:

- the CBPV model is algebraic, i.e., given by an enriched Lawvere theory \mathcal{L} ;
- the operation set Π is surjective, in the sense that the (unique) morphism from the initial $\langle \Pi, \llbracket \text{type} \rrbracket \rangle$ theory $\mathcal{L}_{\langle \Pi, \llbracket \text{type} \rrbracket \rangle}$ to \mathcal{L} is an \mathcal{E} -morphism, i.e., surjective in the set-theoretic case; and
- the MAIL signature is simple.

The first assumption is essential, as we cannot construct the conservative restriction model otherwise. The second assumption is reasonable, as it means we include *all* the effects in our effect analysis. If the initial morphism is not an \mathcal{E} -morphism, the theory \mathcal{L} may include effects that lie beyond the reach of our type-and-effect analysis. As we discussed, the last assumption is restrictive, but non-essential. We keep it as it simplifies the account greatly.

Lemma 10.16. Let Σ^{MAIL} be a simple MAIL signature, $\mathcal{M} = \langle \mathcal{V}, \llbracket \text{type} \rrbracket, T, O[-] \rangle$ a CBPV $\Sigma_{\natural}^{\text{CBPV}}$ -model, and $\langle \mathcal{V}, \llbracket \text{type} \rrbracket, \mathbb{P}, O_-[-] \rangle$ a Σ -model. If $\mathbb{P}\Pi = T$, then for every built-in constant $c \in S$, the type interpretation of A_c induced by \mathbb{P} is $\mathcal{M} \llbracket A_c \rrbracket$.

Let \mathcal{E}^{law} be a class of morphisms in $\mathbf{Law}_{\lambda}\mathcal{V}$. We say that an algebraic CBPV Π^{CBPV} -model is \mathcal{E}^{law} -covered if the initial morphism $\mathfrak{T} : \mathcal{L}_{\langle \Pi, \llbracket \text{type} \rrbracket \rangle} \rightarrow \mathcal{L}$ is an \mathcal{E}^{law} -morphism.

Corollary 10.17. *Let Σ^{MAIL} be a simple MAIL signature, let*

$$\mathcal{M} = \langle \lambda, \mathcal{V}, \mathcal{B}[-], \mathcal{L}, \mathcal{L}[-], \mathcal{K}[-] \rangle$$

be an algebraic CBPV $\Sigma_{\sharp}^{\text{CBPV}}$ -model, and let $\langle \mathcal{E}^{\text{law}}, \mathcal{M}^{\text{law}} \rangle$ be a factorisation system of $\mathbf{Law}_{\lambda} \mathcal{V}$, such that \mathcal{M} is \mathcal{E}^{law} -covered. Let

$$\langle \lambda, \mathcal{V}, \llbracket \text{type} \rrbracket, \mathcal{L}_-, \mathcal{O}_-[-] \rangle$$

be the conservative restriction model corresponding to the CBPV model

$$\langle \lambda, \mathcal{V}, \llbracket \text{type} \rrbracket, \mathcal{L}, \mathcal{L}[-] \rangle$$

(see Theorem 7.12).

The conservative restriction MAIL model \mathcal{M}_{\sharp} is given as the algebraic MAIL Σ^{MAIL} -model

$$\langle \mathcal{V}, \mathcal{B}[-], \mathcal{L}_-, \mathcal{O}_-[-], \mathcal{K}[-] \rangle$$

Proof

From the assumptions follows that $\mathcal{L}_{\Pi} \cong \mathcal{L}$, hence by the previous lemma, the interpretation for the built-in constants from \mathcal{M} can be used in \mathcal{M}_{\sharp} . ■

We now turn to construction of logical relations MAIL models.

Lemma 10.18. *Let Σ^{MAIL} be a simple MAIL signature, and let $\mathcal{B}[-]$ be the diagonal base-type interpretation in \mathbf{LogRel} , i.e., for every $Q \in \mathbf{Bsc}$, $\mathcal{B}[Q]$ is the diagonal relation. Then, for every ground type $G \in \mathbf{Gnd}$, $\mathcal{G}[G]$ is the diagonal relation.*

Proof

By induction over ground types, noting that the diagonal relations are closed under products and coproducts. ■

We will use the following construction to relate different MAIL semantics.

Theorem 10.19. *Given a MAIL signature Σ^{MAIL} , consider any two MAIL Σ^{MAIL} -models sharing the same base-type interpretation:*

$$\mathcal{M}_1 = \langle \mathbf{Set}, \mathbb{P}_1, \mathcal{B}^0[-], \mathcal{O}_-^1[-], \mathcal{K}^1[-] \rangle$$

$$\mathcal{M}_2 = \langle \mathbf{Set}, \mathbb{P}_2, \mathcal{B}^0[-], \mathcal{O}_-^2[-], \mathcal{K}^2[-] \rangle$$

Let $\langle \mathbf{LogRel}, \llbracket \text{type} \rrbracket, \mathbb{P}, \mathcal{O}_-[-] \rangle$ be the free lifting of the induced Σ -models

$$\langle \mathbf{Set}, \llbracket \text{type} \rrbracket, \mathbb{P}_1, \mathcal{O}_-^1[-] \rangle, \quad \langle \mathbf{Set}, \llbracket \text{type} \rrbracket, \mathbb{P}_2, \mathcal{O}_-^2[-] \rangle$$

via the diagonal lifting of $\llbracket \text{type} \rrbracket$.

Denote by $\mathcal{B} \llbracket - \rrbracket$ the diagonal lifting of the shared base-type interpretation $\mathcal{B}^0 \llbracket - \rrbracket$. The assignment $\mathcal{B} \llbracket - \rrbracket$ and the functor \mathbb{P} then induce a logical relations MAIL type interpretation $\mathcal{V} \llbracket - \rrbracket$.

If, for every $c \in S$, the pair $\langle \mathcal{M}_1 \llbracket c \rrbracket, \mathcal{M}_2 \llbracket c \rrbracket \rangle$ lifts to a logical relations morphism $\mathcal{K} \llbracket - \rrbracket : \mathbb{1} \rightarrow \mathcal{V} \llbracket A_c \rrbracket$, then

$$\mathcal{M} = \langle \mathbf{LogRel}, \mathbb{P}, \mathcal{B} \llbracket - \rrbracket, O_- \llbracket - \rrbracket, \mathcal{K} \llbracket - \rrbracket \rangle$$

is a logical relation MAIL Σ^{MAIL} -model. We call \mathcal{M} the free lifting of $\mathcal{M}_1, \mathcal{M}_2$.

Proof

By the previous lemma, $\mathcal{M} \llbracket \text{type} \rrbracket$ is the diagonal lifting of $\llbracket \text{type} \rrbracket$. Thus, by fiat, \mathcal{M} is a MAIL Σ^{MAIL} -model. \blacksquare



To aid accessibility, we recast the model generation process in terms of presentations.

Definition 10.15 (revisited). A simple MAIL signature is a MAIL signature Σ^{MAIL} such that $\Pi \in \mathcal{E}$, and for each built-in constant c , the only effect set appearing in A_c is Π .

We say that a presentation CBPV Π^{CBPV} -model is *fully-covered* if the initial translation from the free presentation whose signature is generated by $\mathcal{R} \llbracket \text{type} \rrbracket$ is surjective. In simpler terms, a model is fully-covered if every term in its presentation can be expressed using the indexed terms that define the generic effects of this model.

Corollary 10.17 (revisited). Let Σ^{MAIL} be a simple MAIL signature, let

$$\mathcal{M} = \langle \mathcal{B} \llbracket - \rrbracket, \text{Ax}, O \llbracket - \rrbracket, \mathcal{K} \llbracket - \rrbracket \rangle$$

be a fully-covered presentation CBPV $\Sigma_{\sharp}^{\text{CBPV}}$ -model. Let

$$\langle \llbracket \text{type} \rrbracket, \text{Ax}_-, O_- \llbracket - \rrbracket \rangle$$

be the conservative restriction model corresponding to the pair $\text{Ax}, O \llbracket - \rrbracket$ (see Theorem 7.12 (revisited)).

The conservative restriction MAIL model \mathcal{M}_{\sharp} is given as the presentation MAIL Σ^{MAIL} -model

$$\langle \mathcal{B} \llbracket - \rrbracket, \text{Ax}_-, O_- \llbracket - \rrbracket, \mathcal{K} \llbracket - \rrbracket \rangle$$

To summarise, we defined a general type-and-effect system and its semantics, and in the algebraic case we constructed the hierarchical semantics from the CBPV semantics and related them to each other.

Chapter 11

Optimisations

The transformation was amazing

—Bon Jovi



In this chapter we develop a general account of semantics preserving, effect-dependent program transformations. These are also known in the literature as effect-dependent *optimisations*, although we make no attempt to establish that these lead to more efficient code (see, for example, the “dragon book” [ALSU06, Subsection 1.4.2] for a similar usage of the term ‘optimisation’).

The optimisations we consider are conditional equations between MAIL phrases, i.e., effect annotated CBPV terms. Such equations are *valid* in a MAIL model if their denotations are equal. These equations consist of a simple, equational, program logic which is strictly more powerful than its unannotated counterpart: erasing the effect annotations from the terms may make a valid transformation invalid. We also show that the annotated logic is sound and complete, and show its use as a formal basis for the correctness of program optimisation based on these equations.

With the foundational notions in place, we then validate the optimisations in the literature. Our general account describes the optimisations as falling into three natural classes, based on which properties of the models validate them. Thus we locate the semantic source of these optimisations. As a consequence, by searching these sources for additional optimisations, we come across optimisations not covered by the current literature.

First, in Section 11.1, we define the equational logic and its semantics. We then show that the conservative restriction model is sound and complete with respect to the original CBPV semantics. Next, we formally justify optimisation by effect-dependent transformation. Then, in Section 11.2, we classify the optimisations existing in the

literature. We show how the algebraic viewpoint explains the semantic origins of these optimisations, and allows us to discover new ones.

11.1 Validity

First, we define denotational equivalence:

Definition 11.1. *Let Π^{CBPV} be a CBPV signature, let \mathcal{M} be a CBPV Π^{CBPV} -model, and let $\Gamma \vdash P_1 : X$ and $\Gamma \vdash P_2 : X$ be two well-typed CBPV phrases (value or computation terms). We say that the optimisation*

$$\Gamma \vdash P_1 = P_2 : X$$

is valid in \mathcal{M} if $\llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$.

When the optimisation is valid, we write $\mathcal{M} \models \Gamma \vdash P_1 = P_2 : X$. When Γ and X can be inferred, we simply write $\mathcal{M} \models P_1 = P_2$.

We define validity of optimisations in MAIL models similarly.

Our goal is to validate equivalences between CBPV *programs*. The validation of these equivalences may rely on effect analysis information, i.e., use MAIL models. We thus relate MAIL semantics to CBPV semantics. Recall the erasure function $(-)_\natural$ from Section 10.2. We begin with the benchmark model.

Theorem 11.2. *Let Π^{CBPV} be a CBPV signature, \mathcal{M} a CBPV Π^{CBPV} -model, and Σ an effect hierarchy. Let \mathcal{M}_b be the benchmark model (cf. Lemma 10.14). Then for all well-typed MAIL terms $\Gamma \vdash P : X$,*

$$\mathcal{M}_b \llbracket P \rrbracket = \mathcal{M} \llbracket P_\natural \rrbracket$$

As a consequence, for all MAIL terms $\Gamma \vdash P^1, P^2 : X$:

$$\mathcal{M} \models P_\natural^1 = P_\natural^2 : X_\natural \quad \iff \quad \mathcal{M}_b \models P^1 = P^2 : X$$

Proof

Straightforward induction. ■

However, using the benchmark model on its own serves no purpose, as we could instead just validate the transformations in the CBPV model. Rather, we use the benchmark model to infer semantic equivalence of CBPV terms using conservative restriction MAIL models.

Lemma 11.3. *Let Σ^{MAIL} be a MAIL signature, and \mathcal{M} be any set-theoretic algebraic CBPV Σ^{CBPV} -model.*

Let \mathcal{M}_\sharp be any presentation MAIL Σ^{MAIL} -model such that the underlying Σ -model of \mathcal{M}_\sharp is the conservative restriction model generated from the underlying Π -model of \mathcal{M} . (see Theorem 7.12 (revisited)). Let \mathcal{M}_\flat be the benchmark MAIL Σ^{MAIL} -model arising from \mathcal{M} .



Assume that the condition from Theorem 10.19 holds for the built-in constants:

for every built-in constant $c \in S$, $\langle \mathcal{M}_\sharp \llbracket c \rrbracket, \mathcal{M} \llbracket c \rrbracket \rangle$ lifts to a logical relations morphism:

$$\mathcal{K}^{\text{LogRel}} \llbracket c \rrbracket : \mathbb{1} \rightarrow \mathcal{M}_{\text{LogRel}} \llbracket A_c \rrbracket$$



For every pair of well-typed MAIL programs $\vdash_\varepsilon P^1, P^2 : F_\varepsilon G$,

$$\mathcal{M} \models P^1_\sharp = P^2_\sharp : FG \quad \iff \quad \mathcal{M}_\sharp \models P^1 = P^2 : F_\varepsilon G$$

Proof



By Theorem 10.19, we have a free lifting logical relations MAIL Σ^{MAIL} -model $\mathcal{M}_{\text{LogRel}}$. For every ε , let $m_\varepsilon : T_{\mathcal{L}_\varepsilon} \rightarrow T_{\mathcal{L}}$ be the componentwise injective monad morphism from the construction of the conservative restriction model (see Theorem 7.12).

Consider any MAIL program $\vdash_\varepsilon P : F_\varepsilon G$. By the basic lemma for terms (Lemma 10.13), we deduce that:

$$\langle \mathcal{M}_\sharp \llbracket P \rrbracket, \mathcal{M}_\flat \llbracket P \rrbracket \rangle \in \dot{T}_\varepsilon \mathcal{M}_{\text{LogRel}} \llbracket G \rrbracket$$

By Lemma 10.18, $\mathcal{M}_{\text{LogRel}} \llbracket G \rrbracket$ is the diagonal relation over $\llbracket G \rrbracket$. Therefore, by the definition of the conservative comparison logical relations model (see Corollary 9.12),

$$m_\varepsilon(\mathcal{M}_\sharp \llbracket P \rrbracket) = \mathcal{M}_\flat \llbracket P \rrbracket \stackrel{\text{Theorem 11.2}}{\downarrow} \mathcal{M} \llbracket P_\sharp \rrbracket$$

Consider any pair of well-typed MAIL programs $\vdash_\varepsilon P^1, P^2 : F_\varepsilon G$. By functionality of m_ε we deduce that

$$\mathcal{M} \llbracket P^1_\sharp \rrbracket = \mathcal{M} \llbracket P^2_\sharp \rrbracket \quad \iff \quad \mathcal{M}_\sharp \llbracket P^1 \rrbracket = \mathcal{M}_\sharp \llbracket P^2 \rrbracket$$

and by injectivity of m_ε we deduce that

$$\mathcal{M} \llbracket P^1_\sharp \rrbracket = \mathcal{M} \llbracket P^2_\sharp \rrbracket \quad \implies \quad \mathcal{M}_\sharp \llbracket P^1 \rrbracket = \mathcal{M}_\sharp \llbracket P^2 \rrbracket$$

as desired. ■

The need to incorporate the built-in constants into the conservative restriction model complicates our formulation of this lemma. Indeed, we expose the conditions on the effect-annotated constants that enable to lift the two MAIL models. When we uniformly incorporate these constants into the conservative restriction model, such as with simple signatures (see Corollary 10.17), we obtain a more succinct result:

Theorem 11.4. *Let Σ^{MAIL} be a simple MAIL signature, and \mathcal{M} be any fully-covered presentation CBPV $\Sigma_{\sharp}^{\text{CBPV}}$ -model. Let \mathcal{M}_{\sharp} be the conservative restriction MAIL Σ^{MAIL} -model of Corollary 10.17.*

For every pair of well-typed MAIL programs $\vdash_{\varepsilon} P^1, P^2 : F_{\varepsilon}G$,

$$\mathcal{M} \models P_{\sharp}^1 = P_{\sharp}^2 : FG \quad \iff \quad \mathcal{M}_{\sharp} \models P^1 = P^2 : F_{\varepsilon}G$$

Proof



Our assumptions ensure that Corollary 10.17 is applicable hence \mathcal{M}_{\sharp} is well-defined. The surjectivity of the initial morphism ensures that the component-wise injective monad morphism $m_{\Pi} : T_{\Pi} \rightarrow T$ is the identity function.

Let \mathcal{M}_{\sharp} be the conservative restriction Σ -model induced by \mathcal{M} . Let $\mathcal{M}_{\text{LogRel}}$ be the free lifting model induced by the conservative restriction and the benchmark models, as in the last lemma. A straightforward inductive argument shows that, as $m_{\Pi} = \text{id}$, the interpretation of every value type and computation type containing only Π as an effect set is the diagonal relation.

Therefore, for every built-in constant $c \in S$, the pair $\langle \llbracket c \rrbracket, \llbracket c \rrbracket \rangle$ is in $\mathcal{M}_{\text{LogRel}}A_c$. The previous lemma is applicable. \blacksquare

We justify the process of effect-dependent program optimisation using effect-based equational reasoning. We begin with a complete effect-annotated source code program P , which we wish to transform, i.e., to optimise. To do so, we wish to apply a certain effect-dependent transformation, such as the following Discard optimisation:

$$\frac{\Gamma \vdash_{\varepsilon} M : F_{\varepsilon}A \quad \Gamma \vdash_{\varepsilon'} N : \underline{B}}{\mathcal{M} \models \mathbf{coerce}_{\varepsilon \subseteq \varepsilon'} M \mathbf{to} x : A.N = N} \quad (\varepsilon \subseteq \varepsilon') \quad (11.1)$$

This optimisations states that if N does not directly depend on the result of the returner M , then the computation M may be discarded without affecting the meaning. This optimisation is valid if, for example, $\mathcal{L}_{\varepsilon}$ is the environment theory $\mathcal{L}_{\text{Env}(\mathbb{V})}$. Note that this transformation is not sound in general, by which we mean that, if we erase the effect annotations, we obtain an optimisation that is not valid in the CBPV model for global state:

$$\frac{\Gamma \vdash M : FA \quad \Gamma \vdash N : \underline{B}}{\mathcal{M} \models M \mathbf{to} x : A.N = N} \quad (11.1')$$

Interestingly, transforming Discard (11.1) into a higher-order equation

$$\lambda_m : U_\varepsilon F_\varepsilon A. \lambda_n : U_{\varepsilon'} \underline{B}. \mathbf{force}_m \mathbf{to} x : A. \mathbf{force}_n = \lambda_m : U_\varepsilon F_\varepsilon A. \lambda_n : U_{\varepsilon'} \underline{B}. \mathbf{force}_n$$

yields a valid MAIL equation whose CBPV erasure may be invalid

$$\lambda_m : UFA. \lambda_n : U\underline{B}. \mathbf{force}_m \mathbf{to} x : A. \mathbf{force}_n = \lambda_m : UFA. \lambda_n : U\underline{B}. \mathbf{force}_n$$

which shows that Theorem 11.4 fails at higher types.

We validate such effect-annotated transformations in the conservative restriction model by straightforward calculation. Therefore, applying the transformation to the effect-annotated source program, we obtain a transformed program P' . Standard techniques show our denotational semantics is *compositional*, hence $P = P'$ in the conservative restriction model. By appeal to Theorem 11.4, the unannotated source code of the original program and the transformed program are equal in the original CBPV model, i.e. $P_{\dagger} = P'_{\dagger}$. We can then continue to optimise the annotated source code, or even perform further static analysis, adding tighter effect annotations in the source code, enabling further transformations to occur. For example, if in the above example $N = \mathbf{coerce}_{\varepsilon'' \subseteq \varepsilon'} N'$ and $\varepsilon' = \varepsilon \cup \varepsilon''$, then after M is discarded, we may not need to coerce N' at all, and maintain the tighter effect-set ε'' . Thus, we formally justified effect-dependent optimisation of programs.

First, note our notion of validity, *denotational equivalence*, is natural and, in any adequate model, implies contextual equivalence. In contrast, Benton's work validated the same optimisations only up to *contextual* equivalence with respect to a *restricted* class of ground contexts. Next, note that while the original semantics can prove the same transformations the effect-annotated program can, the benefit of using the effect analysis is in the *locality* of our reasoning. The use of the annotated transformation (11.1) is validated based on local information, facilitated by the available effect sets. In contrast, justifying the same transformation without the annotations requires some form of global analysis, as demonstrated by the unsoundness of the unannotated transformation (11.1'). Also note the bi-implication in Theorem 11.4. So far, we only used this implication for deducing transformation validity in CBPV. The converse implication means that, while in essence our semantics of interest are the CBPV semantics, we lose no expressiveness by moving to the MAIL semantics. However, we may make use of incomplete models to validate optimisations. For example, in previous work [KP12] we used the *axiomatic restriction models*, whose explicit presentation is straightforward, but may produce incomplete MAIL semantics. Finally, despite Theorem 11.4

referring to *programs* only, we foresee no inherent difficulty in using it to justify optimisation of library modules. We outline how to do so. If effect analysis is performed on all system modules and the interface signatures match-up, then Theorem 11.4 justifies the validity of the transformation to the entire system. If other modules are not analysed, then the inter-module interfaces need to be annotated with the full effect-set Π . For example, if an unanalysed module exposes the interface $f : U\mathbf{1} \rightarrow F\mathbf{1}$, we will treat this interface as if it had the annotation $f : U_{\Pi}\mathbf{1} \rightarrow F_{\Pi}\mathbf{1}$. Theorem 11.2 guarantees we can safely assume these annotations, and then invoke Theorem 11.4. We leave a formal treatment of programming languages with explicit module support to further work.

11.2 Optimisation taxonomy

We turn to validating effect-dependent optimisations, formulated as MAIL equations. This is done semantically, using Theorem 11.4. We divide optimisations into *structural*, *local algebraic*, and *global algebraic* groups. We validate versions of almost all the optimisations in Benton et al. [BK99, BKHB06, BB07, BKBH07, BKBH09], and also some others, not previously considered in the semantics literature. The only optimisations considered by Benton et al. that we do not treat deal with exception handlers.

Structural optimisations reflect the general structure of our models. Let Σ^{MAIL} be a MAIL signature. Then a structural optimisation is one that is valid in all MAIL Σ^{MAIL} -models. Figure 11.1 shows example schemes for such optimisations. Note that the β , η , and sequencing laws are the standard CBPV equational laws, found in the CBPV literature [Lev04, PP08]. As the fragment of the language they deal with and their denotational semantics are identical to CBPV, they are indeed true in all MAIL models. The equations for effect interaction are validated by straightforward calculation using the monadic structure and the definition of the free, product, and exponential algebras for a monad. Finally, the equations for coercion follow from functoriality of the monad hierarchy, from basic properties of strong monad morphisms, and from the definition of effect operation preservation. The structural optimisations are the basic transformations in program optimisation, and they include well-known optimisations, such as: constant unfolding and propagation, code inlining, common subexpression elimination, and others.

Local algebraic optimisations originate from particular equations in the Lawvere theory \mathcal{L}_{ϵ} associated to a given MAIL model. Each equation yields such an optimisa-

<p>β laws:</p> <p>match (A, A') as $(x : A, y : A') . M = M [A/x, A'/y]$</p> <p>match inj$_i^{A_1+A_2} A$ as $\{\mathbf{inj}_1 x_1 : A_1 . M_1, \mathbf{inj}_2 x_2 : A_2 . M_2\} = M_i [A/x_i]$</p> <p>force $(\mathbf{thunk} M) = M$ $(\mathbf{return}_{\epsilon} A) \mathbf{to} x : A . M = M [A/x]$</p> <p>$\mathbf{i} \cdot \lambda \{\mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2\} = M_i$ $A \cdot \lambda x : A . M = M [A/x]$</p> <p>$\eta$ laws:</p> <p>$A = \star$</p> <p>$M [A/z] = \mathbf{match} A \mathbf{as} (x : A, y : A') . M [(x, y)/z]$ x, y fresh in M</p> <p>$M = \mathbf{match} A : \mathbf{0} \mathbf{as} \{\}^B$</p> <p>$M [A/z] = \mathbf{match} A \mathbf{as} \begin{cases} \mathbf{inj}_1 x : A . M [\mathbf{inj}_1 x/z], \\ \mathbf{inj}_2 y : A' . M [\mathbf{inj}_2 y/z] \end{cases}$ x, y fresh in M</p> <p>$A = \mathbf{thunk} (\mathbf{force} A)$ $M = M \mathbf{to} x : A . \mathbf{return}_{\epsilon} x$</p> <p>$M = \lambda \{\mathbf{1} \mapsto \mathbf{1}'M, \mathbf{2} \mapsto \mathbf{2}'M\}$ $N = \lambda x : A . x'N$ x fresh in N</p> <p>Sequencing:</p> <p>$M \mathbf{to} x : A . (N \mathbf{to} y : A' . N') = (M \mathbf{to} x : A . N) \mathbf{to} y : A' . N'$ x fresh in N'</p> <p>$M \mathbf{to} x : A . \lambda \{\mathbf{1} \mapsto N_1, \mathbf{2} \mapsto N_2\} = \lambda \begin{cases} \mathbf{1} \mapsto M \mathbf{to} x : A . N_1, \\ \mathbf{2} \mapsto M \mathbf{to} x : A . N_2 \end{cases}$</p> <p>$M \mathbf{to} x : A . \lambda y : A' . N = \lambda y : A' . (M \mathbf{to} x : A . N)$ y fresh in M</p> <p>Effects:</p> <p>$\mathbf{op}_A^B \lambda x : A . (M \mathbf{to} y : A' . N) = (\mathbf{op}_A^{F_{\epsilon} A'} \lambda x : A . M) \mathbf{to} y : A' . N$ x fresh in N</p> <p>$\mathbf{op}_A^{B_1 \times B_2} \lambda x : A . \lambda \begin{cases} \mathbf{1} \mapsto M_1, \\ \mathbf{2} \mapsto M_2 \end{cases} = \lambda \begin{cases} \mathbf{1} \mapsto \mathbf{op}_A^{B_1} \lambda x : A . M_1, \\ \mathbf{2} \mapsto \mathbf{op}_A^{B_2} \lambda x : A . M_2 \end{cases}$</p> <p>$\mathbf{op}_A^{A' \rightarrow B} \lambda x : A . \lambda y : A' . M = \lambda y : A' . \mathbf{op}_A^B \lambda x : A . M$ $x \neq y$</p> <p>Coercion:</p> <p>$\mathbf{coerce}_{\epsilon_2 \subseteq \epsilon_3} (\mathbf{coerce}_{\epsilon_1 \subseteq \epsilon_2} M) = \mathbf{coerce}_{\epsilon_1 \subseteq \epsilon_3} M$</p> <p>$\mathbf{coerce}_{\epsilon_1 \subseteq \epsilon_2} (\mathbf{return}_{\epsilon_1} M) = \mathbf{return}_{\epsilon_2} M$</p> <p>$\mathbf{coerce}_{\epsilon \subseteq \epsilon'} (M \mathbf{to} x : A . N) = (\mathbf{coerce}_{\epsilon \subseteq \epsilon'} M) \mathbf{to} x : A . \mathbf{coerce}_{\epsilon \subseteq \epsilon'} N$</p> <p>$\mathbf{coerce}_{\epsilon_1 \subseteq \epsilon_2} (\mathbf{op}_A^{F_{\epsilon_1} A'} \lambda x : A . M) = \mathbf{op}_A^{F_{\epsilon_2} A'} \lambda x : A . \mathbf{coerce}_{\epsilon_1 \subseteq \epsilon_2} M$</p>

Figure 11.1: structural optimisations

tion. To derive these optimisations from the equations, one follows Plotkin and Pretnar [PP08]. We just give an example here. The global state theory for a finite set of storable values $\mathbb{V} = \{v_1, \dots, v_n\}$ includes the axiom scheme

$$\text{update}_{v_i}(\text{lookup}(\mathbf{x}_1, \dots, \mathbf{x}_n)) = \text{update}_{v_i}(\mathbf{x}_{v_i})$$

The corresponding algebraic optimisation is

$$\text{update}_{\mathbb{V}}^B(\text{lookup } M) = \text{update}_{\mathbb{V}}^B(V \cdot M)$$

Such transformations are the bread-and-butter of effectful program optimisation.

Global algebraic optimisations follow from overall, global properties of the theories. Each appears in two forms, which we call *utilitarian* and *pristine*. The utilitarian form readily applies to program optimisation; the pristine form is shorter and easier to validate, but perhaps less useful. For example, the utilitarian form of Discard is

$$\frac{\Gamma \vdash_{\varepsilon} M : F_{\varepsilon}A \quad \Gamma \vdash_{\varepsilon'} N : B}{\mathcal{M} \models \mathbf{coerce}_{\varepsilon \subseteq \varepsilon'} M \text{ to } x : A. N = N} \quad (\varepsilon \subseteq \varepsilon')$$

and its pristine form is

$$\frac{\Gamma \vdash_{\varepsilon} M : F_{\varepsilon}A}{\mathcal{M} \models M \text{ to } x : A. \mathbf{return}_{\varepsilon^*} = \mathbf{return}_{\varepsilon^*}}$$

The two forms can always be shown equivalent using structural optimisations. Instances of both appear in the work of Benton et al.

Validating the global algebraic optimisations denotationally reveals an intimate connection¹ to Führmann's work on the structure of call-by-value [Füh00, Füh02]: each has a characterisation in semantic terms. For example, the Discard optimisation holds iff $\mathcal{L}_{\varepsilon}$ is an *affine* theory [Koc71, Jac94, Füh00], that is, iff $\eta_{\perp}^{\varepsilon} : \perp \rightarrow T\perp$ is an isomorphism.

There are also *algebraic* characterisations of these semantic properties. For example, a theory \mathcal{L} is affine iff for every term t , $t(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$, that is, iff a *global absorption law* holds. (Here, and below, we are displaying *all* the variables of the terms at hand.) Such algebraic properties can be investigated using the equational presentation of the theory. As an example, the theory for environments $\mathcal{L}_{\text{Env}(\mathbb{V})}$ and the semilattice theory are affine.

A summary of our results appears in Figure 11.2. In every row, all the conditions are equivalent in any MAIL model, taking the condition in the first column as universally quantified over all ε' such that $\varepsilon' \supseteq \varepsilon, \varepsilon_1, \varepsilon_2$. As fully justifying these equivalences

¹Alex Simpson, private communication.

name	utilitarian form	pristine form	abstract side condition	algebraic equivalent	example basic theories
Discard	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{(\text{coerce } M) \text{ to } x : A.N = N}$	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A}{M \text{ to } x : A. \text{return}_* = \text{return}_*}$	$\mathcal{L}_{\mathcal{E}}$ affine: $\eta_{\mathbb{I}}^{\varepsilon} : \mathbb{I} \rightarrow T_{\mathcal{E}}\mathbb{I}$ has a continuous inverse	For all ε -terms t : $t(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$	read-only state, semilattices
Copy	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{\text{coerce } M \text{ to } x : A. \text{coerce } M \text{ to } y : A.N = \text{coerce } M \text{ to } x : A.N[x/y]}$	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A}{M \text{ to } x : A.M \text{ to } y : A. \text{return}_{\mathcal{E}}(x,y) = M \text{ to } x : A. \text{return}_{\mathcal{E}}(x,x)}$	$\mathcal{L}_{\mathcal{E}}$ relevant: $\Psi_{\mathcal{E}} \circ \delta = T_{\mathcal{E}}\delta$	For all ε -terms t : $t(t(\mathbf{x}_1, \dots, \mathbf{x}_n), \dots, \mathbf{x}_n) = t(\mathbf{x}_1, \dots, \mathbf{x}_n)$	exceptions, read-only state, write-only state
Weak Copy	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{\text{coerce } M \text{ to } x : A. \text{coerce } M \text{ to } y : A.N = \text{coerce } M \text{ to } y : A.N}$	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A}{M \text{ to } x : A.M = M}$	$\mu_{\mathcal{E}} \circ T_{\mathcal{E}}\pi_1 \circ \text{str}_{\mathcal{E}} \circ \delta = \text{id}$	For all ε -terms t : $t(\mathbf{x}_1, \dots, \mathbf{x}_n) = t(\mathbf{x}_1, \dots, \mathbf{x}_n)$	any affine or relevant theory: exceptions, read-only and write-only state, semilattices
Swap	$\frac{\Gamma \vdash_{\mathcal{E}_1} M_1 : F_{\mathcal{E}_1}A_1 \quad \Gamma \vdash_{\mathcal{E}_2} M_2 : F_{\mathcal{E}_2}A_2 \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{\text{coerce } M_1 \text{ to } x_1 : A_1. \text{coerce } M_2 \text{ to } x_2 : A_2.N = \text{coerce } M_1 \text{ to } x_1 : A_1.N \text{coerce } M_2 \text{ to } x_2 : A_2.N}$	$\frac{\Gamma \vdash_{\mathcal{E}_1} M_1 : F_{\mathcal{E}_1}A_1 \quad \Gamma \vdash_{\mathcal{E}_2} M_2 : F_{\mathcal{E}_2}A_2}{\text{coerce } M_1 \text{ to } x_1 : A_1. \text{coerce } M_2 \text{ to } x_2 : A_2. \text{return}_{\mathcal{E}}(x_1, x_2) = \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}}(x_1, x_2)}$	$\mathcal{L}_{\mathcal{E}_1} \subseteq \varepsilon, \mathcal{L}_{\mathcal{E}_2} \subseteq \varepsilon$ commute: $\Psi_{\mathcal{E}} \circ (m_{\mathcal{E}_1} \subseteq \varepsilon \times m_{\mathcal{E}_2} \subseteq \varepsilon) = \Psi_{\mathcal{E}} \circ (m_{\mathcal{E}_1} \subseteq \varepsilon \times m_{\mathcal{E}_2} \subseteq \varepsilon)$	$\mathcal{L}_{\mathcal{E}_1} \subseteq \varepsilon$ translations commute with $\mathcal{L}_{\mathcal{E}_2} \subseteq \varepsilon$ translations (see tensor equations)	distinct global memory cells
Weak Swap	$\frac{\Gamma \vdash_{\mathcal{E}_1} M_1 : F_{\mathcal{E}_1}A_1 \quad \Gamma \vdash_{\mathcal{E}_2} M_2 : F_{\mathcal{E}_2}A_2 \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{\text{coerce } M_1 \text{ to } x_1 : A_1. \text{coerce } M_2 \text{ to } x_2 : A_2. \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}} x_1 = \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}} x_1}$	$\frac{\Gamma \vdash_{\mathcal{E}_1} M_1 : F_{\mathcal{E}_1}A_1 \quad \Gamma \vdash_{\mathcal{E}_2} M_2 : F_{\mathcal{E}_2}A_2}{\text{coerce } M_1 \text{ to } x_1 : A_1. \text{coerce } M_2 \text{ to } x_2 : A_2. \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}} x_1}$	$\Psi_{\mathcal{E}} \circ (m_{\mathcal{E}_1} \subseteq \varepsilon \times m_{\mathcal{E}_2} \subseteq \varepsilon) \circ (\text{id} \times \eta_{\mathbb{I}}^{\varepsilon}) = \Psi_{\mathcal{E}} \circ (m_{\mathcal{E}_1} \subseteq \varepsilon \times m_{\mathcal{E}_2} \subseteq \varepsilon) \circ (\text{id} \times \eta_{\mathbb{I}}^{\varepsilon})$	For all ε -terms $t = \mathcal{L}_{\mathcal{E}_1} \subseteq \varepsilon(t')$, $s = \mathcal{L}_{\mathcal{E}_2} \subseteq \varepsilon(s')$: $t(s(\mathbf{x}_1, \dots, \mathbf{x}_n), \dots, \mathbf{x}_n) = s(\mathbf{x}_n, \dots, \mathbf{x}_n) = s(t(\mathbf{x}_1, \dots, \mathbf{x}_n), \dots, \mathbf{x}_n)$	when $\mathcal{L}_{\mathcal{E}_2}$ is affine, e.g.: read-only state and semilattices.
Isolated Swap	$\frac{\Gamma \vdash_{\mathcal{E}_1} M_1 : F_{\mathcal{E}_1}A_1 \quad \Gamma \vdash_{\mathcal{E}_2} M_2 : F_{\mathcal{E}_2}A_2 \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{\text{coerce } M_1 \text{ to } x_1 : A_1. \text{coerce } M_2 \text{ to } x_2 : A_2. \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}} x_1 = \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}} x_1}$	$\frac{\Gamma \vdash_{\mathcal{E}_1} M_1 : F_{\mathcal{E}_1}A_1 \quad \Gamma \vdash_{\mathcal{E}_2} M_2 : F_{\mathcal{E}_2}A_2}{\text{coerce } M_1 \text{ to } x_1 : A_1. \text{coerce } M_2 \text{ to } x_2 : A_2. \text{coerce } M_1 \text{ to } x_1 : A_1. \text{return}_{\mathcal{E}} x_1}$	$\Psi_{\mathcal{E}} \circ (m_{\mathcal{E}_1} \times m_{\mathcal{E}_2}) \circ (\eta_{\mathbb{I}}^{\varepsilon_1} \times \eta_{\mathbb{I}}^{\varepsilon_2}) = \Psi_{\mathcal{E}} \circ (T_{\mathcal{E}_1} \times T_{\mathcal{E}_2}) \circ (\eta_{\mathbb{I}}^{\varepsilon_1} \times \eta_{\mathbb{I}}^{\varepsilon_2})$	For all ε -terms $t = \mathcal{L}_{\mathcal{E}_1} \subseteq \varepsilon(t')$, $s = \mathcal{L}_{\mathcal{E}_2} \subseteq \varepsilon(s')$: $t(s(\mathbf{x}, \dots, \mathbf{x}), \dots, \mathbf{x}) = s(\mathbf{x}, \dots, \mathbf{x}) = s(t(\mathbf{x}, \dots, \mathbf{x}), \dots, \mathbf{x})$	when $\mathcal{L}_{\mathcal{E}_1}$ is affine: read-only state and semilattices.
Unique	$\frac{\Gamma \vdash_{\mathcal{E}} M_i : F_{\mathcal{E}}\mathbf{0}, i = 1, 2 \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{M_1 = M_2}$	(same as utilitarian form)	$T\mathbf{0} = \mathbf{0}, \mathbb{I}$	$\mathcal{L}_{\mathcal{E}}$ equates all ε -constants	all three state theories, semilattices, a single unparametrised exception.
Pure Hoist	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{\text{return}_{\mathcal{E}} \text{think}(\text{coerce } M \text{ to } x : A.N) = M \text{ to } x : A. \text{return}_{\mathcal{E}} \text{think return}_{\mathcal{E}} \text{think } N}$	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A}{\text{return}_{\mathcal{E}} \text{think } M = M \text{ to } x : A. \text{return}_{\mathcal{E}} \text{think return}_{\mathcal{E}} x}$	$T_{\mathcal{E}}\eta_{\mathbb{I}}^{\varepsilon} = \eta_{\mathbb{I}}^{\varepsilon} B$	all ε -terms are equal to variables in $\mathcal{L}_{\mathcal{E}}$	the empty theory, inconsistent theories
Hoist	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A \quad \Gamma \vdash_{\mathcal{E}'} N : \underline{B}}{M \text{ to } x : A. \text{return}_{\mathcal{E}} \text{think}(\text{coerce } M \text{ to } x : A.N) = M \text{ to } x : A. \text{return}_{\mathcal{E}} \text{think return}_{\mathcal{E}} \text{think } N}$	$\frac{\Gamma \vdash_{\mathcal{E}} M : F_{\mathcal{E}}A}{M \text{ to } x : A. \text{think return}_{\mathcal{E}}(x, \text{think } M) = M \text{ to } x : A. \text{think return}_{\mathcal{E}}(x, \text{think return}_{\mathcal{E}} x)}$	$T_{\mathcal{E}}(\eta^{\varepsilon}, \text{id}) = \text{str}_{\mathcal{E}} \circ \delta$	all ε -terms are either a variable or independent of their variables via $\mathcal{L}_{\mathcal{E}}$	exceptions

Figure 11.2: Global algebraic optimisations

will require us to include 36 proofs, we omit them. Benton et al. (Führmann [Füh00, Füh02]) considered call-by-value analogues of the optimisations labelled by **B** (respectively **F**) in Figure 11.2, albeit for particular combinations of effects (respectively for a fixed monad); Führmann also showed the equivalence of the pristine and utilitarian form and the abstract side condition in his setting.

The abstract conditions in Figure 11.2 use the following standard categorical notions. The diagonal function $\delta_B : B \rightarrow B \times B$ is given by $\delta(b) := \langle b, b \rangle$. The two *double strength* functions $\psi, \tilde{\psi} : TA \times TB \rightarrow T(A \times B)$ are defined by $\psi = \theta \circ T\text{str}' \circ \text{str}$ and by $\tilde{\psi} = \theta \circ T\text{str} \circ \text{str}'$.

The algebraic characterisations follow from the algebraic understanding of the monadic structure. Note how Copy corresponds to a global *idempotency* law, and how Swap corresponds to *commutativity*. This algebraic presentation using laws is malleable to manipulation. Slight variations on these two laws generate the Weak Copy², and Weak and Isolated Swap optimisations, which are new in the formal methods optimisation literature. Thus, by understanding the semantic origins of these optimisations in algebraic laws we were able to discover new ones. Also note the algebraic condition for Pure Hoist. It means that \mathcal{L}_ε is either inconsistent, or the operations project one of their arguments without effect.

Note too the two hoisting optimisations and compare them to the structural optimisations dealing with effect operations (Figure 11.1). The simplicity of the latter over the former suggests that the complications arise from the process of *thinking* rather than abstracting over variables. This clean separation between thinks and abstraction also supports our use of CBPV.

To summarise, we formalised effect-dependent optimisation and justified the conservative restriction model construction, in the set-theoretic case, as a sound and complete model of type-and-effect analysis. We then used our models to validate and classify existing optimisations, discover new optimisations, and connect different strands of the semantics literature.

²Paul B. Levy, private communication.

Chapter 12

Combining effects

We could be victims of a lethal combination

—The Bee Gees



A key advantage of the algebraic approach to semantics is *modularity*, where several simple theories are put together via known combinators to give semantics to a complex language. Hyland et al. [HPP06] study two such combinators, the *sum* and *tensor* of two algebraic theories. These operations cover most of the known combinations of effects. In terms of presentations, the sum and the tensor have simple descriptions. In this chapter we will see how set-theoretic presentations of theories assist in deriving the validity of optimisations in the combination of theories from their validity in each component.

We introduce and investigate a novel class of optimisations that allow a highly modular treatment, the *operation-wise valid* optimisations. We show which of the global algebraic optimisations are operation-wise valid and which are not. For the non-operation-wise valid optimisations, we achieve modularity by proving ad-hoc combination theorems.

In this chapter we will use the category of Lawvere theories $\mathbf{Law}_{\mathbb{X}_0}\mathbf{Set}$. However, this category is equivalent to the category of theories \mathbf{Theory} , i.e., the category of presentations and \sim -equivalence classes of translations. Therefore, we do not mark this chapter with the “beware cats” sign, as each use of a Lawvere theory can be replaced with a presentation. In particular, we denote by \mathbf{Th}_{Ax} the Lawvere theory induced by a presentation Ax .

First, in Section 12.1, we begin by recapitulating the known definitions of the sum and the tensor of theories and express their known set-theoretic presentations, summarising the relevant results from [HPP06]. Next, in Section 12.2, we turn to isolating

the operation-wise valid optimisations. Finally, in Section 12.3, we conclude by ad-hoc treatment of the rest of the optimisations.

12.1 Combining theories

The first way to combine theories is by taking their sum, i.e., coproduct, in the category of (enriched) Lawvere theories. Viewed syntactically as presentations (cf. Chapter 8), the sum of two theories has the following well-known straightforward description (see Hyland et al. [HPP06]).

Let $Ax = \langle \sigma_1, E_1 \rangle$, $Ax = \langle \sigma_2, E_2 \rangle$ be two presentations. Denote by $\sigma_1 + \sigma_2$ the signature consisting of the disjoint union of the two operation sets, maintaining their original arities, i.e., assigning $\iota_i \text{op} : n$, for each $\text{op} : n$ in σ_i , $i = 1, 2$. The signature $\sigma_1 + \sigma_2$ induces, for $i = 1, 2$, a family of relabelling maps $\iota_i : \text{Terms}_{\sigma_i}(X) \rightarrow \text{Terms}_{(\sigma_1 + \sigma_2)}(X)$, homomorphically mapping each $\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ to $\iota_i \text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n)$. This relabelling map extends componentwise to equations $t = t'$. Define $Ax_1 + Ax_2$ as the presentation $\langle \sigma_1 + \sigma_2, \iota_1[E_1] \cup \iota_2[E_2] \rangle$. Simply put, $Ax_1 + Ax_2$ consists of the union of the operations of Ax_1 and Ax_2 , possibly renamed to avoid conflict, and the union of the equations, suitably renamed.

With this notation, we have:

Theorem 12.1 (see Hyland et al. [HPP06]). *For every two presentations Ax_1, Ax_2 ,*

$$\text{Th}_{Ax_1} + \text{Th}_{Ax_2} \cong \text{Th}_{(Ax_1 + Ax_2)}$$

Example 12-1. In the proof of Theorem 7.11 we constructed the initial theory \mathcal{L}_σ of a given signature σ as the coproduct $\sum_{\text{op} \in \sigma} \mathcal{L}_{\text{op}}$ where \mathcal{L}_{op} is the initial theory with a single operation op with the same type as in σ . In light of the last theorem, in terms of presentations this construction manifests as the following isomorphisms:

$$\begin{aligned} \mathcal{L}_{\{\text{op}_1:n_1, \dots, \text{op}_n:n_k\}} &\cong \text{Th}_{\langle \{\text{op}_1:n_1, \dots, \text{op}_n:n_k\}, \emptyset \rangle} \\ &\cong \text{Th}_{\langle \sum_{i=1}^k \{\text{op}_i:n_i\}, \emptyset \rangle} \\ &\stackrel{\text{Theorem 12.1}}{\downarrow} \\ &\cong \sum_{i=1}^k \text{Th}_{\langle \{\text{op}_i:n_i\}, \emptyset \rangle} \\ &\cong \sum_{i=1}^k \mathcal{L}_{\{\text{op}_i:n_i\}} \end{aligned}$$

□

The explicit description of the coproduct of two monads, even in the set-theoretic case, is complicated, see Adámek et al. [AMBL12]. For our purposes it suffices to refer to the special case where one of the theories is free, which Hyland et al. [HPP06] treat. The full details of the general description are beyond the scope of this thesis, and we merely instantiate it to obtain the following two examples.

Example 12-2. We describe the monad for combining terminal I/O with non-deterministic choice. Let $\mathbb{C}\text{har}$ be a finite set denoting terminal characters. The signature for terminal I/O (cf. Example 8-23) is given by

$$\sigma_{\text{IO}(\mathbb{C}\text{har})} := \left\{ \text{input} : |\mathbb{C}\text{har}|, \text{input}_c : 1 \mid c \in \mathbb{C}\text{har} \right\}$$

and the theory for terminal I/O is the free theory $\mathcal{L}_{\text{IO}(\mathbb{C}\text{har})} \cong \text{Th}_{\langle \sigma_{\text{IO}(\mathbb{C}\text{har})}, \emptyset \rangle}$. Let $\mathcal{L}_{\text{ND}} \cong \text{Th}_{\langle \{\vee:2\}, E_{\text{ND}} \rangle}$ be the theory for non-determinism, i.e., E_{ND} is the theory of semi-lattices (cf. Example 8-4):

$$\mathbf{x} \vee (\mathbf{y} \vee \mathbf{z}) = (\mathbf{x} \vee \mathbf{y}) \vee \mathbf{z}, \quad \mathbf{x} \vee \mathbf{y} = \mathbf{y} \vee \mathbf{x}, \quad \mathbf{x} \vee \mathbf{x} = \mathbf{x}$$

Recall that the corresponding monad to \mathcal{L}_{ND} is the non-empty, finite powerset monad $\mathcal{P}_+^{\mathbf{x}0}(-)$.

We follow Hyland et al. [HLPP07] and combine terminal I/O with nondeterminism using the sum $\mathcal{L}_{\text{IO}(\mathbb{C}\text{har})} + \mathcal{L}_{\text{ND}} \cong \text{Th}_{\langle \sigma_+, E_+ \rangle}$, where

$$\begin{aligned} \sigma_+ &= \sigma_{\text{IO}(\mathbb{C}\text{har})} + \sigma_{\text{ND}} = \left\{ \vee, \text{input} : |\mathbb{C}\text{har}|, \text{output}_c \mid c \in \mathbb{C}\text{har} \right\} \\ E_+ &= E_{\text{ND}} = \{ \mathbf{x} \vee (\mathbf{y} \vee \mathbf{z}) = (\mathbf{x} \vee \mathbf{y}) \vee \mathbf{z}, \mathbf{x} \vee \mathbf{y} = \mathbf{y} \vee \mathbf{x}, \mathbf{x} \vee \mathbf{x} = \mathbf{x} \} \end{aligned}$$

We calculated the following monad T_+ corresponding to $\mathcal{L}_{\text{IO}(\mathbb{C}\text{har})} + \mathcal{L}_{\text{ND}}$ using the techniques of Hyland et al. [HPP06], which are beyond the scope of this thesis.

Let X be any set. The set T_+X is given by $\mathcal{P}_+^{\mathbf{x}0}(SX)$, where SX is defined inductively as follows.

- For every $x \in X$, $x \in SX$.
- For every family $\langle \tau_c \rangle_{c \in \mathbb{C}\text{har}}$ of non-empty, finite subsets $\tau_c \subseteq SX$, $\langle I, \langle \tau_c \rangle \rangle \in SX$.
- For every $c \in \mathbb{C}\text{har}$, and every non-empty, finite subset $\tau \subseteq SX$, $\langle O, c, \tau \rangle \in SX$.

Let $f : X \rightarrow Y$ be any function. The function $T_+f : T_+X \rightarrow T_+Y$ is given by the direct image $T_+f : \tau \mapsto \tilde{f}[\tau]$, where $\tilde{f} : SX \rightarrow SY$ is defined inductively as follows.

- For every $x \in X$, $\tilde{f}(x) := f(x)$.

- For every family $\langle \tau_c \rangle_{c \in \text{Char}}$, $\tilde{f}(\langle I, \langle \tau_c \rangle \rangle) := \langle I, \langle \tilde{f}[\tau_c] \rangle \rangle$.
- For every $c \in \text{Char}$, and every τ , $\tilde{f}(\langle O, c, \tau \rangle) := \langle O, c, \tilde{f}[\tau] \rangle$.

The monadic unit is given by $\eta^+ : x \mapsto \{x\}$. The monadic multiplication is given by the direct image $\mu^+ : \kappa \mapsto \bigcup \tilde{\mu}[\kappa]$, where $\tilde{\mu} : ST_+X \rightarrow T_+X$ is defined inductively as follows.

- For every $\tau \in T_+X$, $\tilde{\mu}(\tau) := \tau$.
- For every family $\langle \tau_c \rangle_{c \in \text{Char}}$, $\tilde{\mu}(\langle I, \langle \tau_c \rangle \rangle) := \{ \langle I, \langle \tilde{f}[\tau_c] \rangle \rangle \}$.
- For every $c \in \text{Char}$, and every τ , $\tilde{\mu}(\langle O, c, \tau \rangle) := \{ \langle O, c, \tilde{f}[\tau] \rangle \}$.

Finally, we define the generic effects for this monad:

$$\begin{aligned} \text{toss} &: \mathbb{1} \rightarrow T_+2 \\ \text{toss} &: \star \mapsto \{\iota_1 \star, \iota_2 \star\} \end{aligned}$$

$$\begin{aligned} \text{get} &: \mathbb{1} \rightarrow T_+\text{Char} & \text{put} &: \text{Char} \rightarrow T_+\mathbb{1} \\ \text{get} &: \star \mapsto \{ \langle I, \langle \{c\} \rangle_{c \in \text{Char}} \rangle \} & \text{put} &: c \mapsto \{ \langle O, c, \{ \star \} \rangle \} \end{aligned}$$

We can also recover these effects by lifting via the coproduct injection monad morphisms. The injection $\iota_2 : \mathcal{P}_+^{\mathbb{N}_0}(-) \rightarrow T_+$ is given by the inclusions $\mathcal{P}_+^{\mathbb{N}_0}(X) \subseteq T_+X$. The coproduct injection $\iota_1 : T_{1/O}(\text{Char}) \rightarrow T_+$ is given, for every set X , inductively by

- For every $x \in X$, $\iota_1(x) := \{x\}$.
- For every family $\langle t_c \rangle_{c \in \text{Char}}$, $\iota_1(\langle I, \langle t_c \rangle \rangle) := \{ \langle I, \langle \iota_1(t_c) \rangle \rangle \}$.
- For every $c \in \text{Char}$, and $t \in T_{1/O}(\text{Char})X$, $\iota_1(\langle O, c, t \rangle) := \{ \langle O, c, \iota_1(t) \rangle \}$.

Straightforward calculation shows these injections are component-wise injective. \square

The following example is taken from Hyland et al. [HPP06]).

Example 12-3. We combine exceptions with other theories by summing. Let \mathbb{E} be a set denoting possible exceptions, for example 64-bit words, integers, or a set of identifiers. Recall from Example 8-23 that the theory for exception raising is the free theory $\mathcal{L}_{\text{Exc}(\mathbb{E})} \cong \text{Th}_{\langle \sigma_{\text{Exc}(\mathbb{E})}, \emptyset \rangle}$, where

$$\sigma_{\text{Exc}(\mathbb{E})} := \{ \text{throw}_e : 0 \mid e \in \mathbb{E} \}$$

Recall that the corresponding monad $T_{\text{Exc}(\mathbb{E})}$ is $\mathbb{E} + -$.

Levy¹ noted that summing with $\mathcal{L}_{\text{Exc}(\mathbb{E})}$ yields the exceptions monad transformer [KW93]. More precisely, given any theory \mathcal{L} whose corresponding monad is T , the monad corresponding to $\mathcal{L}_{\text{Exc}(\mathbb{E})} + \mathcal{L}$ is $T(\mathbb{E} + -)$. The monadic unit is given by

$$X \xrightarrow{\eta^{\text{Exc}(\mathbb{E})}} \mathbb{E} + X \xrightarrow{\eta} T(\mathbb{E} + X)$$

The monadic multiplication is given by

$$T(\mathbb{E} + T(\mathbb{E} + X)) \xrightarrow{T[\eta \circ \text{id}]} T^2(\mathbb{E} + X) \xrightarrow{\mu} T(\mathbb{E} + X)$$

The injections are given by

$$T \xrightarrow{T\iota_2} T(\mathbb{E} + -) \xleftarrow{\eta} \mathbb{E} + -$$

We can use them to transport the operations from each monad into the combined monad using Corollary 2.9. □

We now turn to combinations in which effects from different theories *commute*. The following universal characterisation of this combination is one of the main technical contributions of Hyland et al. [HPP06]:

Theorem 12.2 ([HPP06, Appendix A]). *Let $\mathcal{L}_1, \mathcal{L}_2$ be (finitary) Lawvere (set-theoretic) theories. There exists a Lawvere theory $\mathcal{L}_1 \otimes \mathcal{L}_2$, unique up to a canonical isomorphism, such that, for every category \mathcal{V} with finite products:*

$$\mathbf{Mod}(\mathcal{L}_1 \otimes \mathcal{L}_2, \mathcal{V}) \simeq \mathbf{Mod}(\mathcal{L}_1, \mathbf{Mod}(\mathcal{L}_2, \mathcal{V}))$$

We call $\mathcal{L}_1 \otimes \mathcal{L}_2$ the tensor product of \mathcal{L}_1 and \mathcal{L}_2 . Moreover, there exist a pair of Lawvere theory morphisms

$$\mathcal{L}_1 \xrightarrow{\overset{\otimes}{\iota}_1} \mathcal{L}_1 \otimes \mathcal{L}_2 \xleftarrow{\overset{\otimes}{\iota}_2} \mathcal{L}_2$$

such that, for every pair of morphisms $\mathfrak{T}_1 : \mathcal{L}_1 \rightarrow \mathcal{L}'_1$, $\mathfrak{T}_2 : \mathcal{L}_2 \rightarrow \mathcal{L}'_2$, there exists a unique morphism $\mathfrak{T}_1 \otimes \mathfrak{T}_2 : \mathcal{L}_1 \otimes \mathcal{L}_2 \rightarrow \mathcal{L}'_1 \otimes \mathcal{L}'_2$, satisfying:

$$\begin{array}{ccccc}
 \mathcal{L}_1 & \xrightarrow{\overset{\otimes}{\iota}_1} & \mathcal{L}_1 \otimes \mathcal{L}_2 & \xleftarrow{\overset{\otimes}{\iota}_2} & \mathcal{L}_2 \\
 \mathfrak{T}_1 \downarrow & & \downarrow \mathfrak{T}_1 \otimes \mathfrak{T}_2 & & \downarrow \mathfrak{T}_2 \\
 \mathcal{L}'_1 & \xrightarrow{\overset{\otimes}{\iota}_1} & \mathcal{L}'_1 \otimes \mathcal{L}'_2 & \xleftarrow{\overset{\otimes}{\iota}_2} & \mathcal{L}'_2
 \end{array}$$

¹Paul B. Levy, unpublished work.

The tensor \otimes equips the category of Lawvere theories with a symmetric monoidal structure whose monoidal unit is the initial Lawvere theory $\mathbf{Pres}_{\mathbf{x}_0}^{\text{op}} \cong \text{Th}_{\langle \emptyset, \emptyset \rangle}$.

In terms of presentations, the tensor product has a straightforward characterisation. Let σ_1, σ_2 be two signatures. Define $E_{\sigma_1 \otimes \sigma_2}$ to be the following set of equations over the combined signature $\sigma_1 + \sigma_2$ consisting of the all the equations

$$\begin{aligned} \text{op}_1(\text{op}_2(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{1,n}), \dots, \text{op}_2(\mathbf{x}_{m,1}, \dots, \mathbf{x}_{m,n})) \\ = \\ \text{op}_2(\text{op}_1(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{m,1}), \dots, \text{op}_1(\mathbf{x}_{1,n}, \dots, \mathbf{x}_{m,n})) \end{aligned}$$

for every $\text{op}_1 : m$ in σ_1 , and $\text{op}_2 : n$ in σ_2 . We can write these equations more succinctly using tuples:

$$\text{op}_1 \left\langle \text{op}_2 \langle \mathbf{x}_{i,j} \rangle_{j=1}^n \right\rangle_{i=1}^m = \text{op}_2 \left\langle \text{op}_1 \langle \mathbf{x}_{i,j} \rangle_{i=1}^m \right\rangle_{j=1}^n$$

We now define the tensor product of two presentations:

$$\mathbf{Ax}_1 \otimes \mathbf{Ax}_2 := \text{Th}_{\langle \sigma_1 + \sigma_2, \iota_1[E_1] \cup \iota_2[E_2] \cup E_{\sigma_1 \otimes \sigma_2} \rangle}$$

The tensor product has the following characterisation, commonly used to define it:

Theorem 12.3 (see Hyland et al. [HPP06]). *For every two presentations $\mathbf{Ax}_1, \mathbf{Ax}_2$,*

$$\text{Th}_{\mathbf{Ax}_1} \otimes \text{Th}_{\mathbf{Ax}_2} \cong \text{Th}_{(\mathbf{Ax}_1 \otimes \mathbf{Ax}_2)}$$

The tensor morphisms $\mathcal{L}_1 \xrightarrow{\overset{\otimes}{\iota_1}} \mathcal{L} \xleftarrow{\overset{\otimes}{\iota_2}} \mathcal{L}_2$ translate by relabelling the operations by i . The action of \otimes on translations $\mathfrak{T}_1 : \mathbf{Ax}_1 \rightarrow \mathbf{Ax}'_1, \mathfrak{T}_2 : \mathbf{Ax}_2 \rightarrow \mathbf{Ax}'_2$ acts by combining the translation as for the sum.

Corollary 12.4. $\text{Th}_{(\mathbf{Ax}_1 \otimes \mathbf{Ax}_2)}$ is the initial theory \mathcal{L} with two specified morphisms $\mathcal{L}_1 \xrightarrow{\overset{\otimes}{\iota_1}} \mathcal{L} \xleftarrow{\overset{\otimes}{\iota_2}} \mathcal{L}_2$ satisfying all the tensor equations in $E_{\sigma_1 \otimes \sigma_2}$.

The next two examples are two of the main contributions of Hyland et al. [HPP06].

Example 12-4 (see Hyland et al. [HPP06, Theorem 10]). The state monad transformer arises from tensoring with global state. Let $\mathbb{V}, |\mathbb{V}| \geq 2$ be a finite set denoting storable values, and $\mathcal{L}_{\text{GS}(\mathbb{V})}$ be the corresponding global state theory. Let \mathcal{L} be any Lawvere theory whose corresponding monad is T . Then the monad corresponding to $\mathcal{L}_{\text{GS}(\mathbb{V})} \otimes \mathcal{L}$ is the monad resulting from applying the *global state monad transformer* [Mog90, Subsection 4.1.2] to T .

Explicitly, the combined monad is given by $(T(\mathbb{V} \times -))^{\mathbb{V}}$. The monadic unit is given by $x \mapsto \lambda v. \eta \langle v, x \rangle$, and the monadic multiplication is given by

$$\begin{aligned} & \left(T \left(\mathbb{V} \times (T(\mathbb{V} \times X))^{\mathbb{V}} \right) \right)^{\mathbb{V}} \\ & \cong \left(T \left((T(\mathbb{V} \times X))^{\mathbb{V}} \times \mathbb{V} \right) \right)^{\mathbb{V}} \xrightarrow{(T\text{eval})^{\mathbb{V}}} (T^2(\mathbb{V} \times X))^{\mathbb{V}} \xrightarrow{(\mu)^{\mathbb{V}}} (T(\mathbb{V} \times X))^{\mathbb{V}} \end{aligned}$$

Note how instantiating with the identity monad, i.e., the monad resulting from the initial Lawvere theory $\mathbf{Pres}_{\mathbf{x}_0}^{\text{op}}$, yields the global state monad.

The morphism $\overset{\otimes}{\mathbf{i}}_1 : T_{\text{GS}(\mathbb{V})} \rightarrow (T(\mathbb{V} \times -))^{\mathbb{V}}$ is given by

$$(\mathbb{V} \times X)^{\mathbb{V}} \xrightarrow{(\eta)^{\mathbb{V}}} (T(\mathbb{V} \times X))^{\mathbb{V}}$$

or, more explicitly,

$$\overset{\otimes}{\mathbf{i}}_1 : f \mapsto \eta \circ f$$

The morphism $\overset{\otimes}{\mathbf{i}}_2 : T \rightarrow (T(\mathbb{V} \times -))^{\mathbb{V}}$ is given by

$$\lambda \mathbb{V}. \left((TX) \times \mathbb{V} \cong \mathbb{V} \times TX \xrightarrow{\text{str}} T(\mathbb{V} \times X) \right)$$

or, more explicitly,

$$\overset{\otimes}{\mathbf{i}}_2 : k \mapsto \lambda v. (T(\lambda x. \langle v, x \rangle))(k)$$

We can use these maps to transport the algebraic operations from each component to the tensor using Corollary 2.9.

For every finitary monad morphism $m : T \rightarrow T'$, straightforward calculations show that the tensor monad morphism $T_{\text{GS}(\mathbb{V})} \otimes m$ is given by

$$T_{\text{GS}(\mathbb{V})} \otimes m : (T(\mathbb{V} \times -))^{\mathbb{V}} \xrightarrow{m^{\mathbb{V}}} (T'(\mathbb{V} \times -))^{\mathbb{V}}$$

To obtain the theory of finitely many memory locations $\ell \in \mathbb{L}$, we take the \mathbb{L} -fold tensor the global state theory with itself, $\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\mathbb{V})}$. \square

Example 12-5 (see Hyland et al. [HPP06, Theorem 12]). The writer monad transformer arises from tensoring with the theory for the writer monad. Let $M = \langle |M|, \cdot, 1 \rangle$ be any monoid. The theory for the writer monad is $\mathcal{L}_{\text{W}(M)} = \text{Th}_{\langle \sigma_{\text{W}(M)}, E_{\text{W}(M)} \rangle}$ where

$$\sigma_{\text{W}(M)} := \{ \text{act}_m : 1 \mid m \in |M| \}$$

and $E_{\text{W}(M)}$ consists of the following equations, for every $m, m' \in |M|$:

$$\text{act}_1(\mathbf{x}) = \mathbf{x}, \quad \text{act}_m(\text{act}_{m'}(\mathbf{x})) = \text{act}_{m \cdot m'} \mathbf{x}$$

The corresponding monad $T_{\mathbb{W}(M)}$ is $|M| \times -$.

Let \mathcal{L} be any theory and T the corresponding monad. Then the monad corresponding to $\mathcal{L}_{\mathbb{W}(M)} \otimes \mathcal{L}$ is the monad resulting from applying the *writer monad transformer* to \mathcal{L} . This transformer was first introduced in Gurr's thesis [Gur91, Subsection 4.2.3] as the *complexity monad constructor*.

Explicitly, the combined monad is given by $T(|M| \times -)$. The unit is given by $x \mapsto \eta \langle 1, x \rangle$. The monadic multiplication is given by

$$T(|M| \times T(|M| \times X)) \xrightarrow{T_{\text{str}}} T^2(|M| \times |M| \times X) \xrightarrow{\mu} T(|M| \times |M| \times X) \xrightarrow{T(\cdot \times \text{id})} T(|M| \times X)$$

Note how instantiating with the identity monad yields the writer monad.

The morphism $\overset{\otimes}{\mathfrak{i}}_1 : T_{\mathbb{W}(M)} \rightarrow T(|M| \times -)$ is given by the monadic unit

$$|M| \times X \xrightarrow{\eta} T(|M| \times X)$$

The morphism $\overset{\otimes}{\mathfrak{i}}_2 : T \rightarrow (T(|M| \times -))$ is given by

$$TX \xrightarrow{T(x \mapsto \langle 1, x \rangle)} T(|M| \times X)$$

We can use these maps to transport the algebraic operations from each component to the tensor.

For every finitary monad morphism $m : T \rightarrow T'$, straightforward calculations show that the tensor monad morphism $T_{\mathbb{W}(M)} \otimes m$ is given by

$$T_{\mathbb{W}(M)} \otimes m : T(|M| \times -) \xrightarrow{m} T'(|M| \times -)$$

To obtain the theory for finitely many write-only memory locations $\ell \in \mathbb{L}$, we instantiate to the overwriting monoid over some finite set of storable values \mathbb{V} , and take the \mathbb{L} -fold tensor of the overwrite theory with itself, $\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{OW}(\mathbb{V})}$. \square

The following example involving the *environment monad transformer* [LHJ95] does not appear in Hyland et al.'s work [HPP06], but follows the same techniques, and was known to its authors².

Example 12-6. The environment monad transformer arises from tensoring with the environment theory. Let \mathbb{V} , $|\mathbb{V}| = n \geq 2$ be any finite set denoting storable values. Recall the theory for the environment monad (cf. Example 8-17) $\mathcal{L}_{\text{Env}(\mathbb{V})} = \text{Th}\langle \{\text{lookup}_n\}, E_{\text{Env}(\mathbb{V})} \rangle$ where $E_{\text{Env}(\mathbb{V})}$ consists of the following two equations,

$$\text{lookup}(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$$

$$\text{lookup}(\text{lookup}(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{1,n}), \dots, \text{lookup}(\mathbf{x}_{n,1}, \dots, \mathbf{x}_{n,n})) = \text{lookup}(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{n,n})$$

²Gordon D. Plotkin, private communication, 2009.

The second equation may be more succinctly presented using tupling:

$$\text{lookup} \left\langle \text{lookup} \langle \mathbf{x}_{i,j} \rangle_{j=1}^n \right\rangle_{i=1}^n = \text{lookup} \langle \mathbf{x}_{k,k} \rangle_{k=1}^n$$

The corresponding monad $T_{\text{Env}(\mathbb{V})}$ is $-^{\mathbb{V}}$.

Let \mathcal{L} be any theory and T the corresponding monad. Then the monad corresponding to $\mathcal{L}_{\text{Env}(\mathbb{V})} \otimes \mathcal{L}$ is the monad resulting from applying the reader monad transformer to \mathcal{L} .

Explicitly, the combined monad is given by $(T-)^{\mathbb{V}}$. The unit is given by $x \mapsto \lambda v.x$. The monadic multiplication is given by

$$\lambda v. \left(\begin{array}{c} (T(TX)^{\mathbb{V}})^{\mathbb{V}} \times \mathbb{V} \xrightarrow{\delta} (T(TX)^{\mathbb{V}})^{\mathbb{V}} \times \mathbb{V} \times \mathbb{V} \xrightarrow{\text{eval}} (T(TX)^{\mathbb{V}}) \times \mathbb{V} \\ \xrightarrow{\text{costr}} T((TX)^{\mathbb{V}} \times \mathbb{V}) \xrightarrow{T\text{eval}} T^2X \xrightarrow{\mu} TX \end{array} \right)$$

Note how instantiating with the identity monad yields the environment monad.

The morphism $\mathfrak{i}_1^{\otimes} : T_{\text{Env}(\mathbb{V})} \rightarrow (T-)^{\mathbb{V}}$ is given by the monadic unit

$$X^{\mathbb{V}} \xrightarrow{\eta^{\mathbb{V}}} (TX)^{\mathbb{V}}$$

or, more explicitly,

$$\mathfrak{i}_1^{\otimes} : f \mapsto \eta \circ f$$

The morphism $\mathfrak{i}_2^{\otimes} : T \rightarrow (T-)^{\mathbb{V}}$ is given by

$$TX \xrightarrow{\lambda v.\pi_1} (TX)^{\mathbb{V}}$$

or, more explicitly,

$$\mathfrak{i}_2^{\otimes} : k \mapsto \lambda v.k$$

We can use these maps to transport the algebraic operations from each component to the tensor using Corollary 2.9.

For every finitary monad morphism $m : T \rightarrow T'$, straightforward calculations show that the tensor monad morphism $T_{\text{Env}(\mathbb{V})} \otimes m$ is given by

$$T_{\text{Env}(\mathbb{V})} \otimes m : (T-)^{\mathbb{V}} \xrightarrow{m^{\mathbb{V}}} (T'-)^{\mathbb{V}}$$

To obtain the theory for finitely many read-only memory locations $\ell \in \mathbb{L}$, take the \mathbb{L} -fold tensor the environment theory with itself, $\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{Env}(\mathbb{V})}$. \square

Example 12-7. We describe, in terms of monads, the action of tensoring any theory \mathcal{L} with the injective morphisms

$$\mathcal{L}_{\text{Env}(\mathbb{V})} \xrightarrow{m_{\text{Env}(\mathbb{V})}} \mathcal{L}_{\text{GS}(\mathbb{V})} \xleftarrow{m_{\text{OW}(\mathbb{V})}} \mathcal{L}_{\text{OW}(\mathbb{V})}$$

Straightforward calculations show that the morphism $m_{\text{Env}(\mathbb{V})}$ is given by:

$$\begin{aligned} m_{\text{Env}(\mathbb{V})} \otimes T : (TX)^{\mathbb{V}} &\rightarrow (T(\mathbb{V} \times X))^{\mathbb{V}} \\ \kappa &\mapsto \lambda v. (T(\lambda x. \langle v, x \rangle)) (\kappa(v)) \end{aligned}$$

Analogous calculations show that the morphism $m_{\text{OW}(\mathbb{V})}$ is given by:

$$\begin{aligned} m_{\text{OW}(\mathbb{V})} \otimes T : T((\mathbb{1} + \mathbb{V}) \times X) &\rightarrow (T(\mathbb{V} \times X))^{\mathbb{V}} \\ k &\mapsto \lambda v. \left(T \left(\begin{array}{l} \langle \mathbf{1}_1 \star, x \rangle \mapsto \langle v, x \rangle \\ \langle \mathbf{1}_2 v_0, x \rangle \mapsto \langle v_0, x \rangle \end{array} \right) (k) \right) \end{aligned}$$

□

12.2 Operation-wise validity

Note that the algebraic conditions in the first three rows and the last two rows of Figure 11.2 have the same form: we quantify over all terms t of a theory and require that they satisfy an algebraic condition. We say that such an optimisation is *operation-wise valid* if its validity follows from validating this algebraic condition for the terms of the form $t = \text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n)$, for every $\text{op} : n$ in the theory. We do not provide a rigorous definition of operation-wise validity. Rather, we will formulate an operation-wise validity condition for each optimisation separately.

Theorem 12.5. *The Discard optimisation is operation-wise valid: for every presentation $\text{Ax} = \langle \sigma, E \rangle$, the theory Th_{Ax} satisfies the Discard optimisation if and only if for every $\text{op} : n$ in σ ,*

$$\text{op}(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$$

Proof

The ‘only if’ implication is immediate. Let $\text{Ax} = \langle \sigma, E \rangle$ be a presentation such that for every $\text{op} : n$ in σ , $\text{op}(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$. We prove by induction that all terms t satisfy the *absorption law*

$$t(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$$

If $t = \mathbf{x}$ is a variable, we are done. Consider any

$$t(\mathbf{x}_1, \dots, \mathbf{x}_k) = \text{op}(s^1(\mathbf{y}_{1,1}, \dots, \mathbf{y}_{1,m_1}), \dots, s^n(\mathbf{y}_{n,1}, \dots, \mathbf{y}_{n,m_n}))$$

where $\text{op} : n$ in σ ,

$$\{\mathbf{y}_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \subseteq \{\mathbf{x}_i \mid 1 \leq i \leq k\}$$

and s^1, \dots, s^n satisfy the absorption law, then

$$t(\mathbf{x}, \dots, \mathbf{x}) = \text{op}(s^1(\mathbf{x}, \dots, \mathbf{x}), \dots, s^n(\mathbf{x}, \dots, \mathbf{x})) \stackrel{\text{induction hypothesis}}{\downarrow} \text{op}(\mathbf{x}, \dots, \mathbf{x}) \stackrel{\text{assumption}}{\downarrow} \mathbf{x}$$

Thus, by induction, Th_{Ax} satisfies the Discard optimisation. ■

This last theorem gives an immediate proof that the theories for read-only state, and semilattices validate the Discard optimisation.

The natural candidate for the operation-wise validity condition for the Weak Copy is: each $\text{op} : n$ satisfies the equation:

$$\text{op}(\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n), \dots, \text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n)) = \text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n)$$

The natural candidate for the Copy optimisation is similar, but with an additional index.

Counter-example 12-8. The Copy and Weak Copy optimisations are *not* operation-wise valid. As validity of Copy implies that of Weak Copy, it is enough to show that, for the theory of global state, Weak Copy satisfies the condition for operation-wise validity, yet is not valid.

Indeed, if we denote $\mathbb{V} = \{v_1, \dots, v_n\}$, $n \geq 2$, we have:

$$\begin{aligned} \text{lookup}(\text{lookup}(\mathbf{x}_1, \dots, \mathbf{x}_n), \dots, \text{lookup}(\mathbf{x}_1, \dots, \mathbf{x}_n)) &= \text{lookup}(\mathbf{x}_1, \dots, \mathbf{x}_n) \\ \text{update}_{v_i}(\text{update}_{v_i}(\mathbf{x})) &= \text{update}_{v_i}(\mathbf{x}) \end{aligned}$$

hence Weak Copy is operation-wise valid.

However, Weak Copy is *not* valid for global state. Indeed, consider the term

$$t(\mathbf{x}) := \text{lookup}(\text{update}_{v_n}(\mathbf{x}), \dots, \text{update}_{v_1}(\mathbf{x}))$$

Intuitively, t modifies the memory cell according to $v_i \mapsto v_{n+1-i}$. Then $t(t(\mathbf{x})) = \mathbf{x}$ follows from the global state equations. However, the equation $t(\mathbf{x}) = \mathbf{x}$ does not follow from the global state equations. Indeed its interpretation in $T_{\text{GS}(\mathbb{V})} \mathbb{1}$ in the environment $\mathbf{x} \mapsto \star$ is $\lambda v_i. \langle v_{n+1-i}, \star \rangle$. As $n \geq 2$, this function is different from the function $\lambda v. \langle v, \star \rangle$, which is the interpretation of \mathbf{x} in the same environment. □

We do now know any reasonable condition that can act as an operation-wise validity condition for the Unique optimisation.

Theorem 12.6. *The Pure Hoist optimisation is operation-wise valid. Explicitly, for every presentation $Ax = \langle \sigma, E \rangle$, the theory Th_{Ax} satisfies the Pure Hoist optimisation if and only if for every $\text{op} : n$ in σ there exists an $1 \leq i \leq n$ such that*

$$\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \mathbf{x}_i$$

Proof

The ‘only if’ implication is immediate. Let $Ax = \langle \sigma, E \rangle$ be a presentation such that for every $\text{op} : n$ in σ , there exists an some $1 \leq i \leq n$ such that $\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \mathbf{x}_i$. We prove by induction that for all terms $t(\mathbf{x}_1, \dots, \mathbf{x}_m)$ there exists some $1 \leq j \leq m$ such that $t(\mathbf{x}_1, \dots, \mathbf{x}_m) = \mathbf{x}_j$.

If $t = \mathbf{x}$ is a variable, we are done. Consider any

$$t(\mathbf{x}_1, \dots, \mathbf{x}_k) = \text{op}(s^1, \dots, s^n)$$

where $\text{op} : n$ in σ , and s^1, \dots, s^n satisfy the induction hypothesis. By our assumption, there exists some $1 \leq i \leq n$ such that $\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \mathbf{x}_i$. By our induction hypothesis for s^i , there exists some $1 \leq j \leq m_i$ such that $s^i(\mathbf{x}_1, \dots, \mathbf{x}_k) = \mathbf{x}_j$. Therefore:

$$t(\mathbf{x}_1, \dots, \mathbf{x}_k) = s^i(\mathbf{x}_1, \dots, \mathbf{y}_k) = \mathbf{x}_j$$

Thus, t also satisfies the induction hypothesis. ■

This theorem gives an immediate proof that the empty theory (and, of course, both of the inconsistent theories) validate the Pure Hoist optimisation.

Theorem 12.7. *The Hoist optimisation is operation-wise valid: for every presentation $Ax = \langle \sigma, E \rangle$, the theory Th_{Ax} satisfies the Hoist optimisation if and only if for every $\text{op} : n$ in σ , either there exists some $1 \leq i \leq n$, such that*

$$\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \mathbf{x}_i$$

or, for every $\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x}'_1, \dots, \mathbf{x}'_n$,

$$\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_n) = \text{op}(\mathbf{x}'_1, \dots, \mathbf{x}'_n)$$

Proof

The ‘only if’ direction is immediate. Let $Ax = \langle \sigma, E \rangle$ be a presentation satisfying the condition of the theorem. We prove by induction that for all terms $t(\mathbf{x}_1, \dots, \mathbf{x}_k)$ either

there exists some $1 \leq j \leq k$ such that $t(\mathbf{x}_1, \dots, \mathbf{x}_k) = \mathbf{x}_j$, or, for all $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k$, $t(\mathbf{x}_1, \dots, \mathbf{x}_k) = t(\mathbf{x}'_1, \dots, \mathbf{x}'_k)$.

If $t = \mathbf{x}$, we are done. Consider any

$$t = \text{op}(s_1, \dots, s_n)$$

where $\text{op} : n$ in σ , and s_1, \dots, s_n satisfy the induction hypothesis. We proceed by case analysis on our assumption.

Assume that for all $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k$, $\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_k) = \text{op}(\mathbf{x}'_1, \dots, \mathbf{x}'_k)$. Consider any $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k$, and denote $s'_i = s_i(\mathbf{x}'_1, \dots, \mathbf{x}'_k)$. Then:

$$t(\mathbf{x}_1, \dots, \mathbf{x}_k) = \text{op}(s_1, \dots, s_n) = \text{op}(s'_1, \dots, s'_n) = t(\mathbf{x}'_1, \dots, \mathbf{x}'_k)$$

Thus, in this case, the induction hypothesis holds for t .

Assume, therefore, that, $\text{op}(\mathbf{x}_1, \dots, \mathbf{x}_k) = \mathbf{x}_i$ for some i . As s_i also satisfies the induction hypothesis, we can proceed by case analysis. If there exists some j such that $s_i(\mathbf{x}_1, \dots, \mathbf{x}_k) = \mathbf{y}_j$, we conclude as in the previous proof. Otherwise, for all $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k$, we have $s_i(\mathbf{x}_1, \dots, \mathbf{x}_k) = s_i(\mathbf{x}'_1, \dots, \mathbf{x}'_k)$. Consider any $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k$. Then:

$$t(\mathbf{x}_1, \dots, \mathbf{x}_k) = s_i(\mathbf{x}_1, \dots, \mathbf{x}_k) = s_i(\mathbf{x}'_1, \dots, \mathbf{x}'_k) = t(\mathbf{x}'_1, \dots, \mathbf{x}'_k)$$

Thus, t also satisfies the induction hypothesis. ■

This theorem gives an immediate proof that theories involving only constants validate the Hoist optimisation.

Similarly, in the algebraic condition for the various Swap optimisations, we quantify over *pairs* of terms t' and t' from a pair of theories. These three optimisations are also component-wise valid.

Theorem 12.8. *The Swap optimisation is operation-wise valid: for every triple of presentations $\text{Ax}_1 = \langle \sigma_1, E_1 \rangle$, $\text{Ax}_2 = \langle \sigma_2, E_2 \rangle$, and $\text{Ax} = \langle \sigma, E \rangle$ and for every pair of translations $\text{Ax}_1 \xrightarrow{\mathfrak{T}_1} \text{Ax} \xleftarrow{\mathfrak{T}_2} \text{Ax}_2$:*

- $\text{Th}_{\text{Ax}_1} \xrightarrow{\text{Th}_{\mathfrak{T}_1}} \text{Th}_{\text{Ax}} \xleftarrow{\text{Th}_{\mathfrak{T}_2}} \text{Th}_{\text{Ax}_2}$ satisfies the Swap optimisation if and only if for every $\text{op}_1 : m$ in σ_1 and $\text{op}_2 : n$ in σ_2 ,

$$\begin{aligned} & \mathfrak{T}_1(\text{op}_1)(\mathfrak{T}_2(\text{op}_2)(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{1,n}), \dots, \mathfrak{T}_2(\text{op}_2)(\mathbf{x}_{m,1}, \dots, \mathbf{x}_{m,n})) \\ & \qquad \qquad \qquad = \\ & \mathfrak{T}_2(\text{op}_2)(\mathfrak{T}_1(\text{op}_1)(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{m,1}), \dots, \mathfrak{T}_1(\text{op}_1)(\mathbf{x}_{1,n}, \dots, \mathbf{x}_{m,n})) \end{aligned}$$

- $\text{Th}_{\text{Ax}_1} \xrightarrow{\text{Th}_{\mathfrak{T}_1}} \text{Th}_{\text{Ax}} \xleftarrow{\text{Th}_{\mathfrak{T}_2}} \text{Th}_{\text{Ax}_2}$ satisfies the Weak Swap optimisation if and only if for every $\text{op}_1 : m$ in σ_1 and $\text{op}_2 : n$ in σ_2 ,

$$\begin{aligned} & \mathfrak{T}_1(\text{op}_1)(\mathfrak{T}_2(\text{op}_2)(\mathbf{x}_1, \dots, \mathbf{x}_1), \dots, \mathfrak{T}_2(\text{op}_2)(\mathbf{x}_m, \dots, \mathbf{x}_m)) \\ & \qquad \qquad \qquad = \\ & \mathfrak{T}_2(\text{op}_2)(\mathfrak{T}_1(\text{op}_1)(\mathbf{x}_1, \dots, \mathbf{x}_m), \dots, \mathfrak{T}_1(\text{op}_1)(\mathbf{x}_1, \dots, \mathbf{x}_m)) \end{aligned}$$

- $\text{Th}_{\text{Ax}_1} \xrightarrow{\text{Th}_{\mathfrak{T}_1}} \text{Th}_{\text{Ax}} \xleftarrow{\text{Th}_{\mathfrak{T}_2}} \text{Th}_{\text{Ax}_2}$ satisfies the Isolated Swap optimisation if and only if for every $\text{op}_1 : m$ in σ_1 and $\text{op}_2 : n$ in σ_2 ,

$$\begin{aligned} & \mathfrak{T}_1(\text{op}_1)(\mathfrak{T}_2(\text{op}_2)(\mathbf{x}, \dots, \mathbf{x}), \dots, \mathfrak{T}_2(\text{op}_2)(\mathbf{x}, \dots, \mathbf{x})) \\ & \qquad \qquad \qquad = \\ & \mathfrak{T}_2(\text{op}_2)(\mathfrak{T}_1(\text{op}_1)(\mathbf{x}, \dots, \mathbf{x}), \dots, \mathfrak{T}_1(\text{op}_1)(\mathbf{x}, \dots, \mathbf{x})) \end{aligned}$$

Proof

Consider any triple of presentations $\text{Ax}_1 = \langle \sigma_1, E_1 \rangle$, $\text{Ax}_2 = \langle \sigma_2, E_2 \rangle$, and $\text{Ax} = \langle \sigma, E \rangle$ and translations $\text{Ax}_1 \xrightarrow{\mathfrak{T}_1} \text{Ax} \xleftarrow{\mathfrak{T}_2} \text{Ax}_2$. In all three statements, the ‘only if’ implications are immediate, and it remains to establish the converse implications. We begin with Swap, and assume the first condition.

To reduce syntactic clutter, we consider the presentation Ax_{Swap} whose signature is $\sigma_1 + \sigma_2$, and consisting of all the equations:

$$\begin{aligned} & \text{op}_1(\text{op}_2(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{1,n}), \dots, \text{op}_2(\mathbf{x}_{m,1}, \dots, \mathbf{x}_{m,n})) \\ & \qquad \qquad \qquad = \\ & \text{op}_2(\text{op}_1(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{m,1}), \dots, \text{op}_1(\mathbf{x}_{1,n}, \dots, \mathbf{x}_{m,n})) \end{aligned}$$

for every $\text{op}_1 \in \sigma_1$, $\text{op}_2 \in \sigma_2$. Consider the translation $\mathfrak{T} : \text{Ax}_{\text{Swap}} \rightarrow \text{Ax}$ extending \mathfrak{T}_1 , \mathfrak{T}_2 , i.e., $\mathfrak{T}(\text{op}) = \mathfrak{T}_i(\text{op})$ for every op in σ_i . This \mathfrak{T} is indeed a translation, as by our assumption, Th_{Ax} satisfies all the \mathfrak{T} -translations of the equations in Ax_0 .

First, note that for every σ_2 -term $s(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and every $\text{op}_1 : m$ in σ_1 ,

$$\begin{aligned} & \text{op}_1(s(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{1,n}), \dots, s(\mathbf{x}_{m,1}, \dots, \mathbf{x}_{m,n})) \\ & \qquad \qquad \qquad = \\ & s(\text{op}_1(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{m,1}), \dots, \text{op}_1(\mathbf{x}_{1,n}, \dots, \mathbf{x}_{m,n})) \end{aligned}$$

This fact is well-known, and its proof follows by straightforward induction over σ_2 -terms. A similar inductive argument shows that for every σ_1 -term $t(\mathbf{x}_1, \dots, \mathbf{x}_m)$ and

every σ_2 -term $s(\mathbf{x}_1, \dots, \mathbf{x}_n)$,

$$\begin{aligned} & t(s(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{1,n}), \dots, s(\mathbf{x}_{m,1}, \dots, \mathbf{x}_{m,n})) \\ & = \\ & s(t(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{m,1}), \dots, t(\mathbf{x}_{1,n}, \dots, \mathbf{x}_{m,n})) \end{aligned}$$

Applying \mathfrak{T} to the last equation shows that $\text{Th}_{\text{Ax}_1} \xrightarrow{\text{Th}_{\mathfrak{T}_1}} \text{Th}_{\text{Ax}} \xleftarrow{\text{Th}_{\mathfrak{T}_2}} \text{Th}_{\text{Ax}_2}$ satisfies the Swap optimisation.

Repetition of the above argument with a careful modification of the indices proves the statements for the remaining pair of Swap optimisations. ■

We thus addressed the operation-wise validity of all the global algebraic optimisations in Figure 11.2.

12.3 Ad-hoc combination

Unfortunately, not all the global algebraic optimisations are operation-wise valid. The Copy, Weak Copy, and Unique optimisations are not operation-wise valid. Thus, in order to validate them in a combined theory, we need to employ ad-hoc methods. We begin with a general tool: if a theory validates a global algebraic optimisation, then every super-theory of it with the same signature also validates the same optimisation.

Proposition 12.9. *If $\mathcal{L} = \text{Th}_{\langle \sigma, E \rangle}$ validates one of the Discard, Copy, Weak Copy, Unique, Pure Hoise, or Hoist optimisations, and if $\mathcal{L}' = \text{Th}_{\langle \sigma', E' \rangle}$ is any theory with $E \subseteq E'$, then \mathcal{L}' validates the same optimisation.*

Analogously, if $\text{Th}_{\langle \sigma_1, E_1 \rangle} \xrightarrow{\text{Th}_{\mathfrak{T}_1}} \text{Th}_{\langle \sigma, E \rangle} \xleftarrow{\text{Th}_{\mathfrak{T}_2}} \text{Th}_{\langle \sigma_2, bE_2 \rangle}$ satisfy any the various Swap optimisations, and if, further, $\text{Th}_{\langle \sigma_1, E_1' \rangle} \xrightarrow{\text{Th}_{\mathfrak{T}'_1}} \text{Th}_{\langle \sigma', E' \rangle} \xleftarrow{\text{Th}_{\mathfrak{T}'_2}} \text{Th}_{\langle \sigma'_2, E'_2 \rangle}$ are such that $\sigma \subseteq \sigma'$, $E \subseteq E'$, for each $i = 1, 2$, $E_i \subseteq E$, and for each op in σ_i , $\mathfrak{T}_i(\text{op}) = \mathfrak{T}'_i(\text{op})$, then $\text{Th}_{\langle \sigma_1, E_1' \rangle} \xrightarrow{\text{Th}_{\mathfrak{T}'_1}} \text{Th}_{\langle \sigma', E' \rangle} \xleftarrow{\text{Th}_{\mathfrak{T}'_2}} \text{Th}_{\langle \sigma'_2, E'_2 \rangle}$ validate the same swap optimisation.

Proof

Each of the algebraic characterisations of the optimisations only involves the provability of certain equations over the sets of terms. Adding more axioms without changing the signature maintains the provability of these characterisations, and the theorem follows. ■

The first two optimisation we consider are Copy and Weak Copy.

Theorem 12.10. *Let $\mathcal{L} = \text{Th}_{\langle\sigma, E\rangle}$ and \mathcal{L}' be two Lawvere theories satisfying the (Weak) Copy optimisation. If, for every $\text{op} \in \sigma$, $\text{op} : 0$, then $\mathcal{L} + \mathcal{L}'$ and $\mathcal{L} \otimes \mathcal{L}'$ satisfy the same optimisation.*

Note that, as \mathcal{L} only has constant operations, it necessarily satisfies the (Weak) Copy optimisation.

Proof

First, we assume \mathcal{L}' validates the Copy optimisation, and prove the statement for the sum $\mathcal{L} + \mathcal{L}'$. Denote $\mathcal{L}' = \text{Th}_{\langle\sigma', E'\rangle}$, and consider any $\sigma + \sigma'$ -term $u(\mathbf{x}_1, \dots, \mathbf{x}_n)$. As σ consists solely of constants, u must be of the form $t(\mathbf{x}_1, \dots, \mathbf{x}_n, c_1, \dots, c_k)$, where $t(\mathbf{x}_1, \dots, \mathbf{x}_{n+k})$ is a σ' -term, and $c_i : 0$ in σ , for every $1 \leq i \leq k$. Note that t , as a σ' -term, satisfies the idempotency law.

First, we demonstrate the algebraic manipulation in the proof for the case $n = k = 2$.

$$\begin{aligned}
& u(u(\mathbf{x}_1^1, \mathbf{x}_2^1), u(\mathbf{x}_1^2, \mathbf{x}_2^2)) \\
&= t(t(\mathbf{x}_1^1, \mathbf{x}_2^1, c_1, c_2), t(\mathbf{x}_1^2, \mathbf{x}_2^2, c_1, c_2), c_1, c_2) \\
&= t(\overbrace{t(t(\mathbf{x}_1^1, \mathbf{x}_2^1, c_1, c_2), t(\mathbf{x}_1^2, \mathbf{x}_2^2, c_1, c_2), t(\mathbf{x}_1^3, \mathbf{x}_2^3, c_1, c_2), t(\mathbf{x}_1^4, \mathbf{x}_2^4, c_1, c_2))} \\
&\quad \uparrow \\
&\quad (*) \quad \overbrace{t(t(\mathbf{x}_1^1, \mathbf{x}_2^1, c_1, c_2), t(\mathbf{x}_1^2, \mathbf{x}_2^2, c_1, c_2), t(\mathbf{x}_1^3, \mathbf{x}_2^3, c_1, c_2), t(\mathbf{x}_1^4, \mathbf{x}_2^4, c_1, c_2))} \\
&\quad \quad t(\mathbf{x}_1^3, \mathbf{x}_2^3, c_1, c_2), \\
&\quad (**) \quad t(\mathbf{x}_1^4, \mathbf{x}_2^4, c_1, c_2)) \\
&\quad \downarrow \\
&= t(t(\mathbf{x}_1^1, \mathbf{x}_2^2, c_1, c_2), \\
&\quad \quad t(\mathbf{x}_1^1, \mathbf{x}_2^2, c_1, c_2), \\
&\quad \quad t(\mathbf{x}_1^3, \mathbf{x}_2^3, c_1, c_2), \\
&\quad \quad t(\mathbf{x}_1^4, \mathbf{x}_2^4, c_1, c_2)) = t(\mathbf{x}_1^1, \mathbf{x}_2^2, c_1, c_2) = u(\mathbf{x}_1^1, \mathbf{x}_2^2) \\
&\quad \quad \quad \uparrow \\
&\quad \quad \quad (***)
\end{aligned}$$

First, note how in transition (*) we use the idempotency law to introduce new subterms. In particular, we introduce four fresh variables, \mathbf{x}_1^3 , \mathbf{x}_2^3 , \mathbf{x}_1^4 , and \mathbf{x}_2^4 . The freshness of these variables is not important, but merely illustrates that we may choose arbitrary terms in their stead. In transition (**), we simplify the first two arguments in the outermost term using the idempotency law. We then simplify the entire term using the idempotency law in transition (***) to obtain the right-hand side of the idempotency law.

This example generalises to arbitrary n and k , but requires more complex index manipulations. To clarify those manipulations, we write $\langle_{i=1}^n x_i \rangle$ for the sequence

$\langle x_i \rangle_{i=1}^n = \langle x_1, \dots, x_n \rangle$. This notation has the benefit of binding the index *before* usage, while maintaining the lightweight syntax of the sequencing brackets. We suppress string concatenation, by writing $t(\langle_{i=1}^n \mathbf{x}_i \rangle, \langle_{j=1}^k c_j \rangle)$ for $t(\mathbf{x}_1, \dots, \mathbf{x}_n, c_1, \dots, c_k)$.

With these conventions, calculate:

$$\begin{aligned}
& u \left\langle_{i=1}^n u \left\langle_{i'=1}^n \mathbf{x}_{i'}^i \right\rangle \right\rangle \\
&= t \left(\left\langle_{i=1}^n t \left(\left\langle_{i'=1}^n \mathbf{x}_{i'}^i \right\rangle, \left\langle_{j'=1}^k c_{j'} \right\rangle \right) \right\rangle, \left\langle_{j=1}^k c_j \right\rangle \right) \\
&\stackrel{(*)}{=} t \left(\left\langle_{i''=1}^n \left[t \left(\left\langle_{\ell=1}^{n+k} t \left(\left\langle_{i'=1}^n \mathbf{x}_{i'}^\ell \right\rangle, \left\langle_{j'=1}^k c_{j'} \right\rangle \right) \right) \right] \right\rangle, \right. \\
&\quad \left. \left\langle_{j''=1}^k t \left(\left\langle_{i=1}^n \mathbf{x}_i^{n+j''} \right\rangle, \left\langle_{j=1}^k c_j \right\rangle \right) \right\rangle \right) \\
&\stackrel{(**)}{=} t \left(\left\langle_{i''=1}^n t \left(\left\langle_{i'=1}^n \mathbf{x}_{i'}^{i''} \right\rangle, \left\langle_{j'=1}^k c_{j'} \right\rangle \right) \right\rangle, \right. \\
&\quad \left. \left\langle_{j''=1}^k t \left(\left\langle_{i=1}^n \mathbf{x}_i^{n+j''} \right\rangle, \left\langle_{j=1}^k c_j \right\rangle \right) \right\rangle \right) \\
&\stackrel{(***)}{=} t \left(\left\langle_{i=1}^n \mathbf{x}_i^i \right\rangle, \right. \\
&\quad \left. \left\langle_{j=1}^k c_j \right\rangle \right) = u \left\langle_{i=1}^n \mathbf{x}_i^i \right\rangle
\end{aligned}$$

Thus in $\mathcal{L} + \mathcal{L}'$ the idempotency law holds, hence it validates Copy. To show that the tensor also validates Copy, we appeal to Proposition 12.9.

To prove the statement for Weak Copy, erase the superscript from all the variables $\mathbf{x}_{i'}^i$ in the proof. Note how every appeal to Copy can be soundly replaced with an appeal to Weak Copy. Also note we no longer add any fresh variables in the first transition of the calculation. \blacksquare

Our proof consists of two steps. First, we use our assumption on \mathcal{L} to identify a special form to which all $\sigma + \sigma'$ can be brought. In the last proof, it was $t(\mathbf{x}_1, \dots, \mathbf{x}_n, c_1, \dots, c_k)$. Then we establish the idempotency law by direct calculation using the idempotency laws in each component theory. We use this tactic in all our ad-hoc combination theorems for Copy and Weak Copy.

Theorem 12.11. *Let $Ax = \langle \sigma, E \rangle$, $Ax' = \langle \sigma', E \rangle$ be two presentations such that Th_{Ax} and $\text{Th}_{Ax'}$ validate the Copy (resp. Weak Copy) optimisation. If σ assigns arity 1 to all operation symbols, then $\text{Th}_{(Ax \otimes Ax')}$ satisfies the Copy (resp. Weak Copy) optimisation.*

The proof is straightforward, but technically involved. Therefore we first illustrate

it with an example. First, a *unary signature* is a signature that assigns to every operation symbol the arity 1, i.e., in which all operations are unary. A *unary presentation* is a presentation with a unary signature. The first crucial observation is that if Ax is unary, then every $Ax + Ax'$ -term can be separated into an Ax' -term with Ax -terms substituted for its variables.

For example, let σ' , and σ be given by $\{f' : 3\}$ and $\{g, h : 1\}$, respectively. The tensor equation for g and f' in this case is

$$g(f'(\mathbf{x}, \mathbf{y}, \mathbf{z})) = f'(g(\mathbf{x}), g(\mathbf{y}), g(\mathbf{z}))$$

Using the tensor equations, we can separate any $\sigma + \sigma'$ -term by cascading the unary σ operations towards the variables. For example,

$$g(f'(\mathbf{x}, h(\mathbf{y}), \mathbf{z})) = f'(g(\mathbf{x}), g(h(\mathbf{y})), g(\mathbf{z}))$$

In this separated form, we have a σ' -term, $f'(\mathbf{x}, \mathbf{y}, \mathbf{z})$, in which we substitute the σ -terms $g(\mathbf{x})$, $g(h(\mathbf{y}))$, $g(\mathbf{z})$.

With this observation in place, we prove the theorem by directly establishing the idempotency law. For example, consider the term $u(\mathbf{x}_1, \mathbf{x}_2) := f'(\mathbf{x}_1, g(\mathbf{x}_2), h(\mathbf{x}_1))$. We have:

$$\begin{aligned} u(u(\mathbf{x}_1^1, \mathbf{x}_2^1), u(\mathbf{x}_1^2, \mathbf{x}_2^2)) &= f'(f'(\mathbf{x}_1^1, g(\mathbf{x}_2^1), h(\mathbf{x}_2^1))), \\ &\quad \text{tensor} \quad \boxed{g(f'(\mathbf{x}_1^2, g(\mathbf{x}_2^2), h(\mathbf{x}_1^2)))}, \\ &\quad \text{equations} \quad \boxed{h(f'(\mathbf{x}_1^1, g(\mathbf{x}_2^1), h(\mathbf{x}_1^1)))}) \\ &\quad \downarrow \\ &= f'(f'(\mathbf{x}_1^1, g(\mathbf{x}_2^1), h(\mathbf{x}_2^1))), \\ &\quad \text{idempotency} \quad f'(g(\mathbf{x}_1^2), g(g(\mathbf{x}_2^2)), g(h(\mathbf{x}_1^2))), \\ &\quad \text{in } \mathcal{L}' \quad f'(h(\mathbf{x}_1^1), h(g(\mathbf{x}_2^1)), h(h(\mathbf{x}_1^1))) \\ &\quad \downarrow \\ &= f'(\mathbf{x}_1^1, \\ &\quad \quad \boxed{g(g(\mathbf{x}_2^2))}, \\ &\quad \quad \boxed{h(h(\mathbf{x}_1^1))}) = f'(\mathbf{x}_1^1, g(\mathbf{x}_2^2), h(\mathbf{x}_2^1)) = u(\mathbf{x}_1^1, \mathbf{x}_2^2) \\ &\quad \quad \uparrow \\ &\quad \quad \text{idempotency in } \mathcal{L} \end{aligned}$$

The full proof generalises these two steps.

Lemma 12.12. *Let σ be a unary signature, and σ' be any other signature. For every $\sigma + \sigma'$ -term $u(\mathbf{x}_1, \dots, \mathbf{x}_k)$ there exist:*

- a natural number m and a sequence i_1, \dots, i_m from $\{1, \dots, k\}$;

- a σ' -term $t(\mathbf{x}_1, \dots, \mathbf{x}_m)$; and
- σ -terms $s_1(\mathbf{x}), \dots, s_m(\mathbf{x})$

such that $\mathcal{L}_\sigma \otimes \mathcal{L}_{\sigma'}$ proves

$$u(\mathbf{x}_1, \dots, \mathbf{x}_k) = t(s_1(\mathbf{x}_{i_1}), \dots, s_m(\mathbf{x}_{i_m}))$$

Proof

Given any k , we show by induction that $u(\mathbf{x}_1, \dots, \mathbf{x}_k)$ satisfies the lemma.

For $u(\mathbf{x}_1, \dots, \mathbf{x}_k) = \mathbf{x}_c$, for some $1 \leq c \leq k$, choose $m := 1$, $i_1 := c$, $t(\mathbf{x}) := \mathbf{x}$, and $s_1(\mathbf{x}) := \mathbf{x}$, and then

$$t(s_1(\mathbf{x}_{i_1})) = \mathbf{x}_c = u(\mathbf{x}_1, \dots, \mathbf{x}_k)$$

Consider any $u(\mathbf{x}_1, \dots, \mathbf{x}_k) = \text{op}(u^1, \dots, u^\ell)$, such that, for every $1 \leq d \leq \ell$, the $\sigma + \sigma'$ -term $u^d(\mathbf{x}_1, \dots, \mathbf{x}_k)$ satisfies the induction hypothesis. As op is in $\sigma + \sigma'$, we may split into two cases.

Assume op is a σ -operation. Therefore, op is unary, hence $\ell = 1$, and $u = \text{op}(u')$, where $u'(\mathbf{x}_1, \dots, \mathbf{x}_k)$ satisfies the induction hypothesis, i.e., there exist:

- a natural number m and a sequence i_1, \dots, i_m from $\{1, \dots, k\}$;
- a σ' -term $t(\mathbf{x}_1, \dots, \mathbf{x}_m)$; and
- σ -terms $s'_1(\mathbf{x}), \dots, s'_m(\mathbf{x})$

such that $\mathcal{L}_\sigma \otimes \mathcal{L}_{\sigma'}$ proves

$$u' \langle \langle_{c=1}^k \mathbf{x}_c \rangle \rangle = t' \langle \langle_{a=1}^m s'_a(\mathbf{x}_{i_a}) \rangle \rangle$$

To establish the induction hypothesis, take m , $\langle \langle_{a=1}^m i_a \rangle \rangle$, and $t \langle \langle_{a=1}^m \mathbf{x}_a \rangle \rangle$ as themselves, and for every $1 \leq a \leq m$, take $s_a(\mathbf{x}) := \text{op}(s'_a(\mathbf{x}))$. We then have:

$$t \langle \langle_{a=1}^m s(\mathbf{x}_{i_a}) \rangle \rangle = t \langle \langle_{a=1}^m \text{op}(s'_a(\mathbf{x}_{i_a})) \rangle \rangle \stackrel{\text{tensor equations}}{\downarrow} = \text{op}(t \langle \langle_{a=1}^m s'_a(\mathbf{x}_{i_a}) \rangle \rangle) = u \langle \langle_{c=1}^k \mathbf{x}_c \rangle \rangle$$

Thus, in this case, the induction hypothesis holds.

Assume op is a σ' operation. Therefore, for every $1 \leq d \leq \ell$, there exist

- a natural number m^d and a sequence $i_1^d, \dots, i_{m^d}^d$ from $\{1, \dots, k\}$;
- a σ' -term $t^d(\mathbf{x}_1, \dots, \mathbf{x}_{m^d})$; and

- a σ -terms $s_1^d(\mathbf{x}), \dots, s_m^d(\mathbf{x})$

such that $\mathcal{L}_\sigma \otimes \mathcal{L}_{\sigma'}$ proves

$$u^d \langle_{c=1}^k \mathbf{x}_c \rangle = t^d \langle_{a'=1}^{m^d} s_{a'}^d(\mathbf{x}_{i_{a'}^d}) \rangle$$

We establish the induction hypothesis.

- Take $m := \sum_{d=1}^{\ell} m^d$. Denote by $\iota_d : \{1, \dots, m^d\} \rightarrow \{1, \dots, m\}$ the canonical injection. For every $1 \leq d \leq \ell$, and $1 \leq a' \leq m^d$ take $i_{\iota_d a'} := i_{a'}^d$.

- Take

$$t \langle_{a=1}^m \mathbf{x}_a \rangle := \text{op} \langle_{d=1}^{\ell} t^d \langle_{a'=1}^{m^d} \mathbf{x}_{\iota_d a'} \rangle \rangle$$

- Take, for every $1 \leq d \leq \ell$ and $1 \leq a' \leq m^d$:

$$s_{\iota_d a'}(\mathbf{x}) := s_{a'}^d(\mathbf{x})$$

We then have:

$$\begin{aligned} t \langle_{a=1}^m s_a(\mathbf{x}_{i_a}) \rangle &= \text{op} \langle_{d=1}^{\ell} t^d \langle_{a'=1}^{m^d} s_{\iota_d a'}(\mathbf{x}_{i_{\iota_d a'}}) \rangle \rangle \\ &= \text{op} \langle_{d=1}^{\ell} t^d \langle_{a'=1}^{m^d} s_{a'}^d(\mathbf{x}_{i_{a'}^d}) \rangle \rangle \\ &\stackrel{\text{induction hypothesis}}{\downarrow} \\ &= \text{op} \langle_{d=1}^{\ell} u^d \langle_{c=1}^k \mathbf{x}_c \rangle \rangle \\ &= u \langle_{c=1}^k \mathbf{x}_c \rangle \end{aligned}$$

And the induction hypothesis holds. ■

We return to the proof at hand:

Proof of Theorem 12.11

Let $Ax = \langle \sigma, E \rangle$, $Ax' = \langle \sigma', E \rangle$ be two presentations such that Th_{Ax} and $\text{Th}_{Ax'}$ validate the Copy (resp. Weak Copy) optimisation. Assume further that σ is unary.

Consider any $\sigma + \sigma'$ -term $u \langle_{c=1}^k \mathbf{x}_c \rangle$. By Lemma 12.12, there exist:

- a natural number m and a sequence i_1, \dots, i_m from $\{1, \dots, k\}$;
- a σ' -term $t(\mathbf{x}_1, \dots, \mathbf{x}_m)$; and
- σ -terms $s_1(\mathbf{x}), \dots, s_m(\mathbf{x})$

such that $\mathcal{L}_\sigma \otimes \mathcal{L}_{\sigma'}$ proves, and hence $\text{Th}_{(\text{Ax} \otimes \text{Ax}')}$ also proves,

$$u \langle_{c=1}^k \mathbf{x}_c \rangle = t \langle_{a=1}^m s_a(\mathbf{x}_{i_a}) \rangle$$

Calculate:

$$\begin{aligned} u \langle_{c=1}^k u \langle_{c'=1}^k \mathbf{x}_{c'}^c \rangle \rangle &= u \langle_{c=1}^k t \langle_{a'=1}^m s_{a'}(\mathbf{x}_{i_{a'}}^c) \rangle \rangle \\ &= t \langle_{a=1}^m \boxed{s_a \left(t \langle_{a'=1}^m s_{a'}(\mathbf{x}_{i_{a'}}^{i_a}) \rangle \right)} \rangle \\ &\text{tensor equations} \\ &\downarrow \\ &= t \langle_{a=1}^m t \langle_{a'=1}^m s_a(s_{a'}(\mathbf{x}_{i_{a'}}^{i_a})) \rangle \rangle \\ &\text{idempotency in } \mathcal{L}' \\ &\downarrow \\ &= t \langle_{a=1}^m \boxed{s_a(s_a(\mathbf{x}_{i_a}^{i_a}))} \rangle \\ &\text{idempotency in } \mathcal{L} \\ &\downarrow \\ &= t \langle_{a=1}^m s_a(\mathbf{x}_{i_a}^{i_a}) \rangle \\ &= u \langle_{c=1}^k \mathbf{x}_c^c \rangle \end{aligned}$$

Therefore $\text{Th}_{(\text{Ax} \otimes \text{Ax}')}$ proves the idempotency law. Note that by erasing the superscripts from variables in the proof we obtain a proof for the corresponding statement for Weak Copy. \blacksquare

The previous proof depended on the ability to push the operations from σ deeper into the term, towards the variables. In the next theorem we depend on the ability to pull them towards the root of the term.

Theorem 12.13. *Let \mathcal{L} , \mathcal{L}' be two theories that validate the Copy (resp. Weak Copy) optimisation. If \mathcal{L} also validates the Discard optimisation, then $\mathcal{L} \otimes \mathcal{L}'$ validates the Copy (resp. Weak Copy) optimisation.*

We first demonstrate how we separate each $\sigma + \sigma'$ -term when $\text{Th}_{\langle \sigma, E \rangle}$ validates the Discard optimisation, i.e., satisfies the absorption law

$$u(\mathbf{x}, \dots, \mathbf{x}) = \mathbf{x}$$

Consider a theory \mathcal{L} consisting of two operations $\sigma := \{g : 1, h : 3\}$ in which the idempotency and absorption laws hold. Take \mathcal{L}' to be any theory with a single binary operation $\mathcal{L}_{\{f':2\}}$ in which the idempotency law hold. In every $\sigma + \sigma'$ -term we can bubble the σ -operations towards the root of the term. For example:

$$\begin{aligned} &\text{absorption in } \mathcal{L} \\ &\downarrow \\ &f'(g(\mathbf{x}), \boxed{h(\mathbf{y}, \mathbf{z}, \mathbf{y})}) = f'(g(\mathbf{x}), g(h(\mathbf{y}, \mathbf{z}, \mathbf{y}))) \end{aligned}$$

- a σ -term $s(\mathbf{x}_1, \dots, \mathbf{x}_m)$,

such that

- for all $1 \leq d \leq \ell$ and $1 \leq a \leq m$, j_a^d is in $\{1, \dots, m^d\}$; and
- $\mathcal{L} \otimes \mathcal{L}'$ proves

$$\text{op} \left\langle \left\langle_{d=1}^{\ell} s^d \left\langle \left\langle_{a'=1}^{m^d} \mathbf{x}_{a'}^d \right\rangle \right\rangle \right\rangle = s \left\langle \left\langle_{a=1}^m \text{op} \left\langle \left\langle_{d=1}^{\ell} \mathbf{x}_{j_a^d}^d \right\rangle \right\rangle \right\rangle$$

Proof

Consider any \mathcal{L} , \mathcal{L}' , $\text{op} : \ell$, and $\left\langle \left\langle_{d=1}^{\ell} s^d \left\langle \left\langle_{a'=1}^{m^d} \mathbf{x}_{a'}^d \right\rangle \right\rangle \right\rangle$ as in the Lemma's statement. Denote

$$u := \text{op} \left\langle \left\langle_{d=1}^{\ell} s^d \left\langle \left\langle_{a'=1}^{m^d} \mathbf{x}_{a'}^d \right\rangle \right\rangle \right\rangle$$

For every $1 \leq k \leq \ell$, denote by Φ_k the following invariant: there exist

- a natural number m_k and a doubly-indexed sequence $\left\langle \left\langle_{d=1}^k \left\langle \left\langle_{a''=1}^{m_k} k j_{a''}^d \right\rangle \right\rangle \right\rangle$; and
- a σ -term $s_k(\mathbf{x}_1, \dots, \mathbf{x}_{m_k})$,

such that

- for all $1 \leq d \leq k$ and $1 \leq a'' \leq m_k$, $k j_{a''}^d$ is in $\{1, \dots, m^d\}$; and
- $\mathcal{L} \otimes \mathcal{L}'$ proves

$$u = s_k \left\langle \left\langle_{a''=1}^{m_k} \text{op} \left(\left\langle \left\langle_{d=1}^k \mathbf{x}_{k j_{a''}^d}^d \right\rangle, \left\langle \left\langle_{d=k+1}^{\ell} s^d \left\langle \left\langle_{a'=1}^{m^d} \mathbf{x}_{a'}^d \right\rangle \right\rangle \right\rangle \right) \right\rangle \right\rangle$$

Note that Φ_0 holds, as, by taking $m^0 := 1$ and $t_0(\mathbf{x}) := \mathbf{x}$, the conclusion of Φ_0 amounts to $u = u$. Also note that Φ_ℓ is the Lemma's statement. Therefore, it suffices to establish that, for every $1 \leq k < \ell$, Φ_k implies Φ_{k+1} .

Consider any $1 \leq k < \ell$ and assume Φ_k . Therefore, we have some number m_k , sequence $\left\langle \left\langle_{d=1}^k \left\langle \left\langle_{a''=1}^{m_k} k j_{a''}^d \right\rangle \right\rangle \right\rangle$, and term $s_k(\mathbf{x}_1, \dots, \mathbf{x}_{m_k})$ witnessing Φ_k .

Take $m_{k+1} := m_k \times m^{k+1}$. Denote by $\langle -, - \rangle$ the canonical bijection

$$\langle -, - \rangle : \{1, \dots, m_k\} \times \{1, \dots, m^{k+1}\} \cong \{1, \dots, m_{k+1}\}$$

Take, for every $1 \leq a'' \leq m_k$, $1 \leq a \leq m^{k+1}$, and $1 \leq d \leq k$,

$${}^{k+1}j_{\langle a'', a \rangle}^d := {}^k j_{a''}^d$$

and, for $d = k + 1$, take:

$${}^{k+1}j_{\langle a'', a \rangle}^{k+1} := a$$

Note that in both cases indeed ${}^{k+1}j_{a''}^d \leq m^k$. Finally, take

$$s_{k+1} \langle {}_{a'=1}^{m^{k+1}} \mathbf{x}_{a'} \rangle := s_k \langle {}_{a'=1}^{m^k} s^{k+1} \langle {}_{a=1}^{m^{k+1}} \mathbf{x}_{\langle a'', a \rangle} \rangle \rangle$$

Calculate, with care:

$$\begin{aligned}
& s_{k+1} \left\langle {}_{a=1}^{m_{k+1}} \text{op} \left(\left\langle {}_{d=1}^{k+1} \mathbf{x}_{k+1; j_a^d} \right\rangle, \left\langle {}_{d=k+2}^{\ell} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \right\rangle \right) \right\rangle \\
& \text{reindex} \\
& \downarrow \\
& = s_{k+1} \left\langle {}_{a=1}^{m_{k+1}} \text{op} \left(\left\langle {}_{d=1}^k \mathbf{x}_{k+1; j_a^d} \right\rangle, \mathbf{x}_{k+1; j_a^d}, \left\langle {}_{d=k+2}^{\ell} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \right\rangle \right) \right\rangle \\
& = s_k \left\langle {}_{a''=1}^{m^k} s^{k+1} \left\langle {}_{a=1}^{m^{k+1}} \text{op} \left(\left\langle {}_{d=1}^k \mathbf{x}_{k+1; j_{\langle a'', a \rangle}^d} \right\rangle, \mathbf{x}_{k+1; j_{\langle a'', a \rangle}^d}, \left\langle {}_{d=k+2}^{\ell} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \right\rangle \right) \right\rangle \right\rangle \\
& = s_k \left\langle {}_{a''=1}^{m^k} \left[s^{k+1} \left\langle {}_{a=1}^{m^{k+1}} \text{op} \left(\left\langle {}_{d=1}^k \mathbf{x}_{k+1; j_{a''}^d} \right\rangle, \mathbf{x}_a, \left\langle {}_{d=k+2}^{\ell} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \right\rangle \right) \right] \right\rangle \\
& \text{tensor} \\
& \text{equations} \\
& \downarrow \\
& = s_k \left\langle {}_{a''=1}^{m^k} \text{op} \left(\left\langle {}_{d=1}^k \left[s^{k+1} \langle {}_{a=1}^{m^{k+1}} \mathbf{x}_{k+1; j_{a''}^d} \rangle \right] \right\rangle, s^{k+1} \langle {}_{a=1}^{m^{k+1}} \mathbf{x}_a \rangle, \left\langle {}_{d=k+2}^{\ell} \left[s^{k+1} \langle {}_{a=1}^{m^{k+1}} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \rangle \right] \right\rangle \right) \right\rangle \\
& \text{absorption} \\
& \text{law} \\
& \downarrow \\
& = s_k \left\langle {}_{a''=1}^{m^k} \text{op} \left(\left\langle {}_{d=1}^k \mathbf{x}_{k+1; j_{a''}^d} \right\rangle, \left[s^{k+1} \langle {}_{a=1}^{m^{k+1}} \mathbf{x}_a \rangle, \left\langle {}_{d=k+2}^{\ell} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \right\rangle \right] \right) \right\rangle \\
& \text{reindex} \\
& \downarrow \\
& = s_k \left\langle {}_{a''=1}^{m^k} \text{op} \left(\left\langle {}_{d=1}^k \mathbf{x}_{k+1; j_{a''}^d} \right\rangle, \left\langle {}_{d=k+1}^{\ell} s^d \langle {}_{a'=1}^{m^d} \mathbf{x}_{a'} \rangle \right\rangle \right) \right\rangle \\
& \Phi_k \\
& \downarrow \\
& = u
\end{aligned}$$

Therefore Φ_{k+1} holds. ■

Based on this result, we separate arbitrary $\sigma + \sigma'$ terms:

Lemma 12.15. *Let $\mathcal{L} = \text{Th}_{\langle \sigma, E \rangle}$, $\mathcal{L}' = \text{Th}_{\langle \sigma', E' \rangle}$ be two theories. If \mathcal{L} validates the Discard optimisation, then for every $\sigma + \sigma'$ -term $u(\mathbf{x}_1, \dots, \mathbf{x}_c)$ there exist:*

- a natural number m , a sequence of natural numbers $\langle_{a=1}^m n_a \rangle$, and a doubly-indexed sequence $\langle_{a=1}^m \langle_{b'=1}^{n_a} i_{b'}^a \rangle \rangle$ over $\{1, \dots, k\}$;
- a σ -term $s(\mathbf{x}_1, \dots, \mathbf{x}_m)$; and
- a sequence of σ' -terms $\langle_{a=1}^m t_a \langle_{b'=1}^{n_a} \mathbf{x}_{b'}^a \rangle \rangle$

such that $\mathcal{L} \otimes \mathcal{L}'$ proves:

$$u(\mathbf{x}_1, \dots, \mathbf{x}_k) = s \left\langle_{a=1}^m t_a \left\langle_{b'=1}^{n_a} \mathbf{x}_{b'}^a \right\rangle \right\rangle$$

Proof

Consider \mathcal{L} and \mathcal{L}' as in the Lemma's statement, and any natural number k . We prove the Lemma by induction over terms $u(\mathbf{x}_1, \dots, \mathbf{x}_k)$.

For $u(\mathbf{x}_1, \dots, \mathbf{x}_k) = \mathbf{x}_i$, take $m := 1$, $n^1 := 1$, $i_1^1 := i$, $s(\mathbf{x}) := \mathbf{x}$, and $t_1(\mathbf{x}) := \mathbf{x}$. We indeed have:

$$s(t_1(\mathbf{x}_i)) = \mathbf{x}_i = u(\mathbf{x}_1, \dots, \mathbf{x}_k)$$

Consider any $u = \text{op} \langle_{d=1}^{\ell} u^d \rangle$ such that, for every $1 \leq d \leq \ell$, u satisfies the induction hypothesis, i.e., there exist:

- a natural number m^d , a sequence of natural numbers $\langle_{a'=1}^{m^d} n_{a'}^d \rangle$, and a doubly-indexed sequence $\langle_{a'=1}^{m^d} \langle_{b''=1}^{n_{a'}^d} i_{b''}^{a'} \rangle \rangle$ over $\{1, \dots, k\}$;
- a σ -term $s^d(\mathbf{x}_1, \dots, \mathbf{x}_{m^d})$; and
- a sequence of σ' -terms $\langle_{a'=1}^{m^d} t_{a'}^d \langle_{b''=1}^{n_{a'}^d} \mathbf{x}_{b''} \rangle \rangle$

such that $\mathcal{L} \otimes \mathcal{L}'$ proves:

$$u^d(\mathbf{x}_1, \dots, \mathbf{x}_k) = s^d \left\langle_{a'=1}^{m^d} t_{a'}^d \left\langle_{b''=1}^{n_{a'}^d} \mathbf{x}_{b''} \right\rangle \right\rangle$$

As op is in $\sigma + \sigma'$, we may split into two cases.

Assume op is in σ .

- Take $m := \sum_{d=1}^{\ell} m^d$. For every $1 \leq d \leq \ell$ and $1 \leq a' \leq m^d$, take $n_{1_d a'} := n_{a'}^d$, and for every $1 \leq b'' \leq n_{1_d a'}^d$, take $i_{b''}^{1_d a'} := i_{b''}^{a'}$.
- Take $s(\mathbf{x}_1, \dots, m) := \text{op} \left\langle_{d=1}^{\ell} s^d \left\langle_{a'=1}^{m^d} \mathbf{x}_{1_d a'} \right\rangle \right\rangle$.

- For every $1 \leq d \leq \ell$ and $1 \leq a' \leq m^d$, take $t_{\iota_{da'}}(\mathbf{x}_1, \dots, \mathbf{x}_{n_{a'}^d}) := t_{a'}^d \langle_{b''=1}^{n_{a'}^d} \mathbf{x}_{b''} \rangle$.

Calculate:

$$\begin{aligned}
s \langle_{a=1}^m t_a \langle_{b'=1}^{n_a} \mathbf{x}_{i_{b'}^a} \rangle \rangle &= \text{op} \left\langle_{d=1}^{\ell} s^d \left\langle_{a'=1}^{m^d} \boxed{t_{\iota_{da'}} \left\langle_{b''=1}^{n_{\iota_{da'}}} \mathbf{x}_{\boxed{i_{b''}^{\iota_{da'}}}} \right\rangle} \right\rangle \right\rangle \\
&= \text{op} \left\langle_{d=1}^{\ell} \boxed{s^d \left\langle_{a'=1}^{m^d} t_{a'}^d \left\langle_{b''=1}^{n_{\iota_{da'}}} \mathbf{x}_{i_{b''}^{\iota_{da'}}} \right\rangle} \right\rangle \right\rangle \\
&\quad \text{induction hypothesis} \\
&\quad \downarrow \\
&= \text{op} \left\langle_{d=1}^{\ell} u^d \right\rangle = u
\end{aligned}$$

Thus we established the induction hypothesis in this case.

Assume op is in σ' . We invoke Lemma 12.14, and deduce there exist:

- a natural number m and a doubly-indexed sequence $\langle_{d=1}^{\ell} \langle_{a=1}^m j_a^d \rangle$; and
- a σ -term $s(\mathbf{x}_1, \dots, \mathbf{x}_m)$,

such that

- for all $1 \leq d \leq \ell$ and $1 \leq a \leq m$, j_a^d is in $\{1, \dots, m^d\}$; and
- $\mathcal{L} \otimes \mathcal{L}'$ proves

$$\text{op} \left\langle_{d=1}^{\ell} s^d \left\langle_{a'=1}^{m^d} \mathbf{x}_{a'}^d \right\rangle \right\rangle = s \left\langle_{a=1}^m \text{op} \left\langle_{d=1}^{\ell} \mathbf{x}_{j_a^d}^d \right\rangle \right\rangle$$

To establish the induction hypothesis,

- Take m as m from Lemma 12.14. For every $1 \leq a \leq m$, take $n_a := \sum_{d=1}^{\ell} n_{j_a^d}^d$. For every $1 \leq b \leq n_{j_a^d}^d$, take $i_{\iota_{da}}^a := d_{\iota_{da}}^{j_a^d}$, and indeed $i_{\iota_{da}}^a$ is in $\{1, \dots, k\}$.
- Take $s(\mathbf{x}_1, \dots, \mathbf{x}_m)$ as the same s from Lemma 12.14.
- For every $1 \leq a \leq m$, take:

$$t_a \langle_{b'=1}^{n_a} \mathbf{x}_{b'} \rangle := \text{op} \left\langle_{d=1}^{\ell} t_{j_a^d}^d \left\langle_{b=1}^{n_{j_a^d}^d} \mathbf{x}_{\iota_{da}}^d \right\rangle \right\rangle$$

Calculate:

$$\begin{aligned}
s \left\langle_{a=1}^m \left[t_a \left\langle_{b'=1}^{n_a} \mathbf{x}_{i_{b'}^a} \right\rangle \right] \right\rangle &= s \left\langle_{a=1}^m \text{op} \left\langle_{d=1}^{\ell} t_{j_a^d} \left\langle_{b=1}^{n_{j_a^d}^d} \mathbf{x}_{i_{j_a^d}^d} \right\rangle \right\rangle \right\rangle \\
&= s \left\langle_{a=1}^m \text{op} \left\langle_{d=1}^{\ell} t_{j_a^d} \left\langle_{b=1}^{n_{j_a^d}^d} \mathbf{x}_{i_{j_a^d}^d} \right\rangle \right\rangle \right\rangle \\
&\stackrel{\text{Lemma 12.14}}{\downarrow} = \text{op} \left\langle_{d=1}^{\ell} \left[s^d \left\langle_{a'=1}^{m^d} t_{a'}^d \left\langle_{b''=1}^{n_{a'}^d} \mathbf{x}_{i_{b''}^{a'} } \right\rangle \right] \right\rangle \\
&\stackrel{\text{induction hypothesis}}{\downarrow} = \text{op} \left\langle_{d=1}^{\ell} u^d \right\rangle = u
\end{aligned}$$

Thus the induction hypothesis holds in this case too. ■

We are ready to prove our theorem:

Proof of Theorem 12.13

Consider theories $\mathcal{L} = \text{Th}_{\langle \sigma, E \rangle}$, $\mathcal{L}' = \text{Th}_{\langle \sigma', E' \rangle}$ that validate the Copy optimisation, and assume \mathcal{L} also validates the Discard optimisation. Consider any $\sigma + \sigma'$ -term $u(\mathbf{x}_1, \dots, \mathbf{x}_k)$.

By Lemma 12.15 there exist:

- a natural number m , a sequence of natural numbers $\langle_{a=1}^m n_a \rangle$, and a doubly-indexed sequence $\langle_{a=1}^m \langle_{b'=1}^{n_a} i_{b'}^a \rangle \rangle$ over $\{1, \dots, k\}$;
- a σ -term $s(\mathbf{x}_1, \dots, \mathbf{x}_m)$; and
- a sequence of σ' -terms $\langle_{a=1}^m t_a \langle_{b'=1}^{n_a} \mathbf{x}_{i_{b'}^a} \rangle \rangle$

such that $\mathcal{L} \otimes \mathcal{L}'$ proves:

$$u(\mathbf{x}_1, \dots, \mathbf{x}_k) = s \left\langle_{a=1}^m t_a \left\langle_{b'=1}^{n_a} \mathbf{x}_{i_{b'}^a} \right\rangle \right\rangle$$

Calculate:

$$\begin{aligned}
u \langle \langle_{c=1}^k u \langle \langle_{c'=1}^k \mathbf{x}_{c'}^c \rangle \rangle \rangle &= u \langle \langle_{c=1}^k s \langle \langle_{a=1}^m t_{a'} \langle \langle_{b''=1}^{n_{a'}} \mathbf{x}_{i_{b''}^{a'}}^c \rangle \rangle \rangle \rangle \\
&= s \langle \langle_{a=1}^m \boxed{t_a \langle \langle_{b'=1}^{n_a} s \langle \langle_{a'=1}^m t_{a'} \langle \langle_{b''=1}^{n_{a'}} \mathbf{x}_{i_{b''}^{a'}}^{i_a} \rangle \rangle \rangle \rangle} \rangle \rangle \\
&\text{tensor equations} \\
&\downarrow \\
&= s \langle \langle_{a=1}^m s \langle \langle_{a'=1}^m t_a \langle \langle_{b'=1}^{n_a} t_a \langle \langle_{b''=1}^{n_{a'}} \mathbf{x}_{i_{b''}^{a'}}^{i_a} \rangle \rangle \rangle \rangle \rangle \rangle \\
&\text{idempotency} \\
&\downarrow \text{in } \mathcal{L} \\
&= s \langle \langle_{a=1}^m \boxed{t_a \langle \langle_{b'=1}^{n_a} t_a \langle \langle_{b''=1}^{n_{a'}} \mathbf{x}_{i_{b''}^{a'}}^{i_a} \rangle \rangle \rangle} \rangle \rangle \\
&\text{idempotency} \\
&\downarrow \text{in } \mathcal{L}' \\
&= s \langle \langle_{a=1}^m t_a \langle \langle_{b'=1}^{n_a} \mathbf{x}_{i_{b'}^{a'}}^{i_a} \rangle \rangle \rangle \\
&= u \langle \langle_{c=1}^k \mathbf{x}_c^c \rangle \rangle
\end{aligned}$$

Thus $\mathcal{L} \otimes \mathcal{L}'$ validates the Copy optimisation. Note that by erasing the superscripts from variables in the proof we obtain a proof for the corresponding statement for Weak Copy. ■

Finally, we consider the Unique optimisation:

Theorem 12.16. *Let $\mathcal{L} = \text{Th}_{\langle \sigma, E \rangle}$ be a theory. Then \mathcal{L} validates the Unique optimisation if and only if every nullary operation in σ commutes with every operation in σ .*

The above condition, stated explicitly, requires that, for every $c : 0$ and $\text{op} : n$ in σ , \mathcal{L} proves

$$\text{op}(c, \dots, c) = c$$

In particular, for every two nullary operations c, c' , we have $c = c'$.

Proof

The ‘only if’ implication is immediate. For the converse, first note that if σ contains no nullary operations, then it has no constant terms and the algebraic condition for the Unique optimisation is vacuously true.

Assume σ contains at least one nullary operation $c_0 : 0$. We prove by induction that, for every nullary term u with no variables, \mathcal{L} proves $u = c_0$.

Assume $u = \text{op}(u^1, \dots, u^\ell)$ where, for all $1 \leq d \leq \ell$, \mathcal{L} proves $u^d = c_0$. Therefore:

$$t = \text{op}(c_0, \dots, c_0) = c_0$$

Therefore, \mathcal{L} proves all nullary terms equal to each other, and \mathcal{L} validates the Unique optimisation. ■

To summarise, we investigated the two common ways to combine effects, sum and tensor. Six out of the nine global algebraic optimisation from Chapter 11 are operation-wise valid and admit an extremely modular treatment: validity of an optimisation in a combined theory follows from its validity in all component theories. The remaining three optimisations also admit some modular treatment via ad-hoc combination results. As we saw, the proofs of these ad-hoc results involve delicate algebraic manipulation of the equational theories. It is therefore implausible that these arguments can be applied rigorously and without error to concrete programs without our general algebraic scaffolding. Thus our abstract, algebraic, and general approach to effect-dependent optimisations facilitates novel analysis.

Chapter 13

Use case

Let me demonstrate...

—No Doubt



We demonstrate how to use our rigorous tools to deal with the complexity of a non-trivial language.

First, we validate the various optimisations from Chapter 11 and formulate procedures for deciding when an optimisation holds for a given effect set. The modularity results of Chapter 12 produce high-level proofs justifying our conditions.

We show our conditions *necessary/complete*: a compiler implementing our decision procedures will *not* miss an opportunity to apply one of the effect-dependent optimisations. We begin by establishing the necessity of the conditions via traditional pencil-and-paper proofs, which involve some technical difficulties. We propose an alternative treatment via a computational representation of our denotational models. We construct such a model using the HASKELL programming language. We use this model to demonstrate the necessity of our characterisations, under appropriate modelling assumptions.

First, in Section 13.1, we present a language involving global state, exceptions, and non-determinism, and its derived type-and-effect system. Next, in Section 13.2, we present our validity decision procedures and their soundness. Then, in Section 13.3, we establish the completeness of our procedures using pencil-and-paper proofs. Finally, in Section 13.4, we establish their completeness using our HASKELL model.

13.1 Language and semantics

Assume the memory has been partitioned into a finite set of disjoint regions [LG88]

Reg. The set **Reg** is partitioned into three subsets:

- read-only regions **Reg_{RO}**;
- write-only regions **Reg_{WO}**; and
- read-write regions **Reg_{RW}**.

We denote the following two sets:

- the read-able regions **Reg_R** := **Reg_{RO}** \cup **Reg_{RW}**; and
- the write-able regions **Reg_W** := **Reg_{WO}** \cup **Reg_{RW}**.

The MAIL signature in question Σ^{MAIL} is given as follows. The basic types **Bsc** are **Char**, **Word**, **Str** and **Loc**. The effect operations Π and their arities are:

- input : **Char** for terminal input;
- output : **1** \langle **Char** \rangle for terminal output;
- raise : **0** for causing an exception;
- throw : **0** \langle **Str** \rangle for causing an exception with an error message;
- rollback : **0** for causing a rollback exception;
- abort : **0** \langle **Str** \rangle for causing a rollback exception with an error message;
- for all write-able regions $\rho \in \mathbf{Reg}_W$, lookup ^{ρ} : **Word** \langle **Loc** \rangle , note that each region consists of **Loc**-many locations;
- for all read-able regions ρ in **Reg_R**, update ^{ρ} : **1** \langle **Loc** \times **Word** \rangle ; and
- \vee : **2** for non-deterministic choice.

The effect sets \mathcal{E} are given by all (necessarily finite) effect operation subsets $\mathcal{P}(\Pi)$.

Finally, the built-in constants S we choose are:

- ' c ' : **Char** for each ASCII character c ;
- n : **Word** for each 64-bit number n ;

- "s" : **Str** for each character string s ; and
- l : **Loc** for each 64-bit memory address l .

The resulting signature Σ^{MAIL} is indeed simple (see Definition 10.15).

The CBPV signature $\Sigma_{\natural}^{\text{CBPV}}$ is $\langle\langle \mathbf{Bsc}, \Pi, S \rangle, \text{type}, A_{-}\rangle$. The chosen CBPV model \mathcal{M} interprets the basic types as follows:

- $\llbracket \mathbf{Char} \rrbracket$ is 2^8 , the usual ASCII encoding;
- $\llbracket \mathbf{Word} \rrbracket$ is 2^{64} , which we also denote by \mathbb{V} ;
- $\llbracket \mathbf{Str} \rrbracket$ is $\llbracket \mathbf{Char} \rrbracket^{80}$ (see below regarding the length restriction) and
- $\llbracket \mathbf{Loc} \rrbracket$ is 2^{64} , which we also denote by \mathbb{L} .

The theory \mathcal{L} is given as follows (cf. Hyland et al. [HPP06]):

$$\begin{aligned} \mathcal{L}_{\{\text{raise,throw}\}} + & \left(\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{Env}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\mathbb{V})} \right. \\ & \left. \begin{array}{ccc} \rho \in \mathbf{Reg}_{\text{RO}} & \rho \in \mathbf{Reg}_{\text{WO}} & \rho \in \mathbf{Reg}_{\text{RW}} \end{array} \right) \\ & \otimes (\mathcal{L}_{\{\text{rollback,abort}\}} + \mathcal{L}_{\{\text{input}\}} + \mathcal{L}_{\{\text{output}\}} + \mathcal{L}_{\text{ND}}) \end{aligned}$$

where the lookup operations in the signature in the $\langle \rho, \ell \rangle$ -th component of the folded tensor $\bigotimes_{\rho \in \mathbf{Reg}_{\text{RO}}} \mathcal{L}_{\text{Env}(\mathbb{V})}$ are tagged with ρ and ℓ , i.e. $\text{lookup}_{\rho}^{\ell}$, and similarly for the other folded tensors. Finally, the constants are given the obvious interpretations.

We restricted the length of strings to 80 characters to ensure their parameter type has a finite denotation. In this way we may use finitary Lawvere theories and presentations interchangeably (cf. page 196).

Note our choice to combine the theory for non-deterministic choice with write-only memory using the tensor. It is a natural design choice based on the current literature [HPP06]). In Section 13.3, however, our analysis of the conservative restrictions of \mathcal{L} shows that the tensor may be inadequate for combining these two effects, by exhibiting an equation involving write-only memory and non-deterministic choice that is *not* included in the tensor of the two theories.

By Corollary 10.17, we have a conservative restriction model \mathcal{M}_{\natural} . By Theorem 11.4, validating optimisations in this model yields valid optimisations for CBPV.

13.2 Optimisation validation

We turn to validating optimisations, and begin with the operation-wise valid ones.

For each optimisation o in $\{\text{Discard}, \text{Pure Hoist}, \text{Hoist}\}$, we define a set $\zeta^o \subseteq \Pi$ of the operations that satisfy its algebraic condition:

$$\begin{aligned} \zeta^{\text{Discard}} &:= \{\vee, \text{lookup}^\rho \mid \rho \in \mathbf{Reg}_R\} & \zeta^{\text{Pure Hoist}} &:= \emptyset \\ \zeta^{\text{Hoist}} &:= \{\text{raise}, \text{throw}, \text{abort}, \text{rollback}\} \end{aligned}$$

Proposition 13.1. *For each optimisation o in $\{\text{Discard}, \text{Pure Hoist}, \text{Hoist}\}$, if $\varepsilon \subseteq \zeta^o$ then o is valid in the theory \mathcal{L}_ε of $\mathcal{M}_\#$.*

Proof

Note that $\mathbf{x} \vee \mathbf{x} = \mathbf{x}$ in \mathcal{L}_{ND} and \mathcal{L}_{ND} is a subtheory of \mathcal{L} , not necessarily conservative. Therefore $\mathbf{x} \vee \mathbf{x} = \mathbf{x}$ holds in \mathcal{L} . Similarly, for every ρ in \mathbf{Reg}_R , lookup^ρ satisfies the absorption law in \mathcal{L}_{Env} or \mathcal{L}_{GS} and hence \mathcal{L} . Thus all operations in ζ^{Discard} satisfy the absorption law in \mathcal{L} . If $\varepsilon \subseteq \zeta^{\text{Discard}}$, then every op in ε satisfies the absorption law in \mathcal{L} , and hence in the conservative restriction \mathcal{L}_ε . By Theorem 12.5 \mathcal{L}_ε satisfies the Discard optimisation.

A similar argument using Theorem 12.7 shows the corresponding statement for Hoist. Theorem 12.6 shows the statement for Pure Hoist holds. \blacksquare

Analogously, for each o in $\{\text{Swap}, \text{Weak Swap (WSwap)}, \text{isolated swap (ISwap)}\}$ and for each op $\in \Pi$ define the set $\zeta^o(\text{op})$ of effect operations that o -commute with op. For the non-symmetric Weak Swap, this condition is that for op : $A \langle P \rangle$ and op' : $A \langle P' \rangle$, op' $\in \zeta^{\text{WSwap}}(\text{op})$ if and only if for all $p \in \llbracket P \rrbracket$ and $p' \in \llbracket P' \rrbracket$, \mathcal{L} proves that:

$$\begin{aligned} \text{op}_p(\text{op}'_{p'}(\mathbf{x}_1, \dots, \mathbf{x}_1), \text{op}'_{p'}(\mathbf{x}_n, \dots, \mathbf{x}_n)) \\ = \\ \text{op}'_{p'}(\text{op}_p(\mathbf{x}_1, \dots, \mathbf{x}_n), \text{op}_p(\mathbf{x}_1, \dots, \mathbf{x}_n)) \end{aligned}$$

where n is the cardinality of $\llbracket A \rrbracket$.

Note that because Swap implies Weak Swap, which implies Isolated Swap, we have $\zeta^{\text{Swap}}(\text{op}) \subseteq \zeta^{\text{WSwap}}(\text{op}) \subseteq \zeta^{\text{ISwap}}(\text{op})$. These sets are given in Figure 13.1.

From Theorem 12.8 we deduce the validity of swapping:

Proposition 13.2. *Let o be one of the three Swap optimisations. Let $\varepsilon_1, \varepsilon_2 \subseteq \varepsilon$ be three effect sets. If $\varepsilon_2 \subseteq \bigcap_{\text{op} \in \varepsilon_1} \zeta^o(\text{op})$ then the optimisation o is valid for $\mathcal{L}_{\varepsilon_1} \rightarrow \mathcal{L}_\varepsilon \leftarrow \mathcal{L}_{\varepsilon_2}$.*

op	ζ^{Swap}	$\zeta^{\text{WSwap}} \setminus \zeta^{\text{Swap}}$	$\zeta^{\text{ISwap}} \setminus \zeta^{\text{WSwap}}$	$\Pi \setminus \zeta^{\text{ISwap}}$
input	lookup ^p , update ^p	∅	∅	input, output, raise, throw, rollback, abort
output	lookup ^p , update ^p	∅	∅	input, output, raise, throw, rollback, abort
raise	lookup ^p , raise, ∅	∅	∅	input, output, throw, rollback, abort, update ^p
throw	lookup ^p , ∅	∅	∅	input, output, raise, throw, rollback, abort, update ^p
rollback	lookup ^p , update ^p , rollback, ∅	∅	∅	input, output, raise, throw, abort
abort	lookup ^p , update ^p , ∅	∅	∅	input, output, raise, throw, rollback, abort
lookup ^{p₀}	input, output, raise, throw, abort, rollback, abort, lookup ^p , update ^{p ≠ p₀} , ∅	∅	update ^{p₀}	∅
update ^{p₀}	input, output, rollback, abort, lookup ^{p ≠ p₀} , update ^{p ≠ p₀} , ∅	lookup ^{p₀}	∅	raise, throw, update ^{p₀}
∅	raise, throw, rollback, abort, lookup ^p , update ^p , ∅	∅	input, output	∅

Figure 13.1: swap sets

Proof

We omit the 48 calculations that show the operations in the middle three columns of the table do indeed satisfy the corresponding swap law in \mathcal{L} , and hence in \mathcal{L}_ε . Most of these calculations follow from the tensor equations. All of these calculations are straightforward. Theorem 12.8 completes the proof. ■

Note that despite our brute force examination of about 160 pairs of effect operations, we are still exponentially better off than trying to exhaust the space of 2^{16} possible pairs $\varepsilon_1, \varepsilon_2$. In Section 13.4 we will generate these optimisation tables mechanically.

We now turn to non-operation-wise valid optimisations, and begin with Copy.

Proposition 13.3. *Let $\varepsilon \subseteq \Pi$ be any effect set. If $\text{input}, \text{update}, \vee \notin \varepsilon$ and, for all ρ in $\mathbf{Reg}_{\mathbf{RW}}$, $\{\text{lookup}^\rho, \text{update}^\rho\} \not\subseteq \varepsilon$, then \mathcal{L}_ε validates the Copy optimisation.*

Proof

The premise of the proposition guarantees that \mathcal{L}_ε has a sub-signature of the signature for

$$\mathcal{L}' := \mathcal{L}_{\{\text{raise}, \text{throw}\}} + (\mathcal{L}_{\{\text{rollback}, \text{abort}\}} \otimes \bigotimes \mathcal{L}_{\text{Env}} \otimes \bigotimes \mathcal{L}_{\text{OW}})$$

Moreover, if \mathcal{L}'_ε is the conservative restriction of \mathcal{L}' to ε , then \mathcal{L}_ε satisfies all \mathcal{L}'_ε -equations, and possibly more.

Each combined theory validates the Copy optimisation. The first two theories contain only constants, hence Theorem 12.10 is applicable. As \mathcal{L}_{OW} is unary, Theorem 12.11 is applicable for its combination. As \mathcal{L}_{Env} also validates Discard, Theorem 12.13 is applicable for its combination. Thus, \mathcal{L}' satisfies the idempotency law, hence \mathcal{L}'_ε also satisfies it. Consequently, \mathcal{L}_ε satisfies this law too, and therefore validates Copy. ■

We treat the Weak Copy optimisation similarly:

Proposition 13.4. *Let $\varepsilon \subseteq \Pi$ be any effect set. If $\text{input}, \text{output} \notin \varepsilon$ and for all ρ in $\mathbf{Reg}_{\mathbf{RW}}$, $\{\text{lookup}^\rho, \text{update}^\rho\} \not\subseteq \varepsilon$, then \mathcal{L}_ε validates the Weak Copy optimisation.*

Proof

The difference from the previous proof is that we also need to consider the theory for non-determinism \mathcal{L}_{ND} , which satisfies both Weak Copy and Discard. Therefore, we may apply Theorem 12.13. The rest of the proof is identical. ■

Finally, we treat the Unique optimisation:

Proposition 13.5. *Let $\varepsilon \subseteq \Pi$ be any effect set. If $\varepsilon \cap \{\text{raise}, \text{rollback}\} = \{\text{op}\}$ and $\varepsilon \subseteq \zeta^{\text{Swap}}(\text{op})$, or if the intersection is empty, then \mathcal{L}_ε validates the Unique optimisation.*

Proof

If the intersection is empty, then vacuously all nullary operations commute with every operation. If $\varepsilon \cap \{\text{raise}, \text{rollback}\} = \{\text{op}\}$, then op is the only nullary constant in ε . The condition $\varepsilon \subseteq \zeta^{\text{Swap}}(\text{op})$ guarantees all ε -operations commute with op . Either way, Theorem 12.16 ensures \mathcal{L}_ε validates the Unique optimisation. ■

13.3 Completeness

In the previous section we gave sufficient conditions in terms of effect sets for the validity of the global algebraic optimisations. We now consider whether these conditions are necessary. We will do so by showing that whenever the conditions in the previous section do not hold, we can construct a counter-example to the validity of the optimisation. The counter-example is an algebraic term that does not satisfy the corresponding algebraic condition.

This process is conceptually straightforward for the operation-wise valid optimisations. All we need is to show that every $\text{op} \notin \zeta^o$, when substituted in the corresponding algebraic characterisation, violates it. For example, $\text{raise} \notin \zeta^{\text{Discard}}$, it suffices to show that the $\mathcal{L}_\varepsilon \not\vdash \text{raise} = \mathbf{x}$ for every ε containing raise . For the other optimisations, we need to exhibit ad-hoc ε -terms that violate the algebraic characterisation in \mathcal{L}_ε . For example, if ε contains both lookup^{ρ_0} and update^{ρ_0} for some ρ_0 , it suffices to show that for

$$t := \text{lookup}^{\rho_0}(\text{update}_{\sim 0}^{\rho_0}(\mathbf{x}), \dots, \text{update}_{\sim (2^{64}-1)}^{\rho_0}(\mathbf{x}))$$

where \sim is the bit-wise NOT function,

$$\mathcal{L}_\varepsilon \not\vdash t(t(\mathbf{x})) = t(\mathbf{x})$$

(cf. Counter-example 12-8).

Unfortunately, as the explicit description of \mathcal{L}_ε is not readily available, establishing non-provability in \mathcal{L}_ε is difficult. We outline two methods to get around this obstacle. The first method, which we pursue in the next section, relies on the defining property

of \mathcal{L}_ε as the *conservative restriction* of \mathcal{L} :

$$\mathcal{L}_\varepsilon \vdash t = s \quad \iff \quad \mathcal{L} \vdash t = s$$

It suffices, therefore, to analyse provability in \mathcal{L} . Recall that the provability relation is captured completely via validity in the free models given by the monad T corresponding to \mathcal{L} . Therefore, if we know the precise description of T , we can calculate the interpretations $\llbracket t \rrbracket, \llbracket s \rrbracket$. By showing they have different interpretations, we deduce the invalidity of the optimisation in \mathcal{L}_ε . Recall that the theory \mathcal{L} is given by:

$$\begin{aligned} \mathcal{L}_{\{\text{raise,throw}\}} + & \left(\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{Env}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\mathbb{V})} \right. \\ & \left. \begin{array}{ccc} \rho \in \mathbf{Reg}_{\text{RO}} & \rho \in \mathbf{Reg}_{\text{WO}} & \rho \in \mathbf{Reg}_{\text{RW}} \end{array} \right) \\ & \otimes (\mathcal{L}_{\{\text{rollback,abort}\}} + \mathcal{L}_{\{\text{input}\}} + \mathcal{L}_{\{\text{output}\}} + \mathcal{L}_{\text{ND}}) \end{aligned}$$

Using Examples 12-2–12-6, we deduce that T is given by

$$\left(T_{\text{ND+IO}(\text{Chor})} \left(\text{Str} + \mathbb{1} + \left((\mathbb{1} + \mathbb{V})^{\mathbf{Reg}_{\text{WO}} \times \mathbb{L}} \times \mathbb{V}^{\mathbf{Reg}_{\text{RW}} \times \mathbb{L}} \times (\text{Str} + \mathbb{1} + -) \right) \right) \right)^{\mathbb{V}^{\mathbf{Reg}_{\text{R}} \times \mathbb{L}}}$$

Our example language is still simple enough to be manageable with paper-and-pencil calculations with this monad. However, it is clear that with more complicated languages such calculations will become much more error-prone and complicated. In the next section we will demonstrate how to deal with this complexity using mechanised assistance.

The second method for establishing non-provability in \mathcal{L}_ε is to give an explicit description of the conservative restriction \mathcal{L}_ε by describing its corresponding monad. We obtain such descriptions by taking the explicit descriptions of the conservative restrictions of the component theories, such as $\mathcal{L}_{\text{GS}(\mathbb{V})}$, and combining them to obtain explicit descriptions of their sum and tensor. We pursue this method in the remainder of this section.

Lemma 13.6. *For every pair of injective translations $\mathcal{L}_1 \xrightarrow{\mathfrak{T}_1} \mathcal{L}'_1$, $\mathcal{L}_2 \xrightarrow{\mathfrak{T}_2} \mathcal{L}'_2$, their sum is also injective:*

$$\mathcal{L}_1 + \mathcal{L}_2 \xrightarrow{\mathfrak{T}_1 + \mathfrak{T}_2} \mathcal{L}'_1 + \mathcal{L}'_2$$

Proof

We use the more general results of Adámek et al. [AMBL12], who proved this lemma when $\mathcal{L}'_1 + \mathcal{L}'_2$ is consistent. We prove the lemma for the additional simple case when $\mathcal{L}'_1 + \mathcal{L}'_2$ is inconsistent.

Assume $\mathcal{L}'_1 + \mathcal{L}'_2$ is inconsistent. Therefore, for some $i \in \{1, 2\}$, \mathcal{L}'_i is inconsistent [AMBL12, Corollary VI.6]. As \mathfrak{T}_i is injective:

$$\mathcal{L}'_i \vdash \mathfrak{T}_i(\mathbf{x}) = \mathbf{x} = \mathbf{y} = \mathfrak{T}_i(\mathbf{y}) \quad \Longrightarrow \quad \mathcal{L}_i \vdash \mathbf{x} = \mathbf{y}$$

and therefore \mathcal{L}_i is inconsistent, hence $\mathcal{L}_1 + \mathcal{L}_2$ is inconsistent too. But any translation from an inconsistent theory is injective, and we are done. ■

We can now deduce an explicit description of the conservative restriction of the sum of two theories from the restrictions of its components.

Theorem 13.7. *Let $\mathcal{L}^1 = \text{Th}_{\langle \sigma_1, E_1 \rangle}$, $\mathcal{L}^2 = \text{Th}_{\langle \sigma_2, E_2 \rangle}$ be two theories, and let \mathcal{L} be their sum $\mathcal{L}^1 + \mathcal{L}^2$. Let $\varepsilon_1 \subseteq \sigma_1$, $\varepsilon_2 \subseteq \sigma_2$ be two effect-sets. Then the conservative restrictions satisfy:*

$$\mathcal{L}_{\varepsilon_1 + \varepsilon_2} \cong \mathcal{L}_{\varepsilon_1}^1 + \mathcal{L}_{\varepsilon_2}^2$$

Proof

Let $\mathcal{L}_{\varepsilon_i}^i \xrightarrow{\mathfrak{T}_i} \mathcal{L}^i$ be the translation mapping each ε_i -term t to itself. By the definition of the conservative restriction, this translation is injective. Therefore, by Lemma 13.6, $\mathcal{L}_{\varepsilon_1}^1 + \mathcal{L}_{\varepsilon_2}^2 \xrightarrow{\mathfrak{T}_1 + \mathfrak{T}_2} \mathcal{L}^1 + \mathcal{L}^2 = \mathcal{L}$ is injective. This translation maps every $\varepsilon_1 + \varepsilon_2$ -term to itself, and therefore, for every pair of $\varepsilon_1 + \varepsilon_2$ -terms t, s satisfying $\mathcal{L} \vdash t = s$, we have:

$$\mathcal{L} \vdash (\mathfrak{T}_1 + \mathfrak{T}_2)(t) = t = s = (\mathfrak{T}_1 + \mathfrak{T}_2)(s) \quad \Longrightarrow \quad \mathcal{L}_{\varepsilon_1}^1 + \mathcal{L}_{\varepsilon_2}^2 \vdash t = s$$

hence $\mathcal{L}_{\varepsilon_1}^1 + \mathcal{L}_{\varepsilon_2}^2$ is the conservative restriction of $\mathcal{L}_{\varepsilon_1 + \varepsilon_2}$. ■

The situation for the tensor of two theories is more delicate. In particular, the analogous result for tensor does not hold. More precisely, given two conservative restrictions $\mathcal{L}_1 \mapsto \mathcal{L}'_1$ and $\mathcal{L}_2 \mapsto \mathcal{L}'_2$, their tensor $\mathcal{L}_1 \otimes \mathcal{L}_2 \rightarrow \mathcal{L}'_1 \otimes \mathcal{L}'_2$ may not form a conservative restriction.

Example 13-1. Let \mathcal{L}_{Mon} be the theory of monoids, given by two operations, $\cdot : 2$ and $1 : 0$, subject to the three equations:

$$1 \cdot \mathbf{x} = \mathbf{x} = \mathbf{x} \cdot 1 \quad \mathbf{x} \cdot (\mathbf{y} \cdot \mathbf{z}) = (\mathbf{x} \cdot \mathbf{y}) \cdot \mathbf{z}$$

The Eckmann-Hilton argument [EH62] shows that $\mathcal{L}_{\text{Mon}} \otimes \mathcal{L}_{\text{Mon}}$ is the theory of *commutative* monoids $\mathcal{L}_{\text{ComMon}}$, i.e., with the additional equation

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$$

We recount this argument.

First, construct a translation $\mathfrak{T} : \mathcal{L}_{\text{Mon}} \otimes \mathcal{L}_{\text{Mon}} \rightarrow \mathcal{L}_{\text{ComMon}}$, identifying the two monoid operations and the two monoid units. The four tensor equations:

$$(\mathbf{x} \cdot_2 \mathbf{y}) \cdot_1 (\mathbf{z} \cdot_2 \mathbf{w}) = (\mathbf{x} \cdot_1 \mathbf{z}) \cdot_2 (\mathbf{y} \cdot_1 \mathbf{w}) \quad 1_1 = 1_2 \quad 1_1 \cdot_2 1_1 = 1_1 \quad 1_2 \cdot_1 1_2 = 1_2$$

translates into

$$(\mathbf{x} \cdot \mathbf{y}) \cdot (\mathbf{z} \cdot \mathbf{w}) = (\mathbf{x} \cdot \mathbf{z}) \cdot (\mathbf{y} \cdot \mathbf{w}) \quad 1 = 1 \quad 1 \cdot 1 = 1 \quad 1 \cdot 1 = 1$$

All these equations hold in the theory of commutative monoids.

Conversely, $\mathcal{L}_{\text{Mon}} \otimes \mathcal{L}_{\text{Mon}}$ proves $1_1 = 1_2$. Substituting these mutual units into both \mathbf{y} and \mathbf{z} in the first tensor equation proves:

$$\mathbf{x} \cdot_1 \mathbf{w} = (\mathbf{x} \cdot_2 1_2) \cdot_1 (1_2 \cdot_2 \mathbf{w}) = (\mathbf{x} \cdot_1 1_1) \cdot_2 (1_1 \cdot_1 \mathbf{w}) = \mathbf{x} \cdot_2 \mathbf{w}$$

Thus, the two monoid products coincide. Substituting the units into \mathbf{x} and \mathbf{w} in the tensor equation proves:

$$\mathbf{y} \cdot_1 \mathbf{z} = (1_2 \cdot_2 \mathbf{y}) \cdot_1 (\mathbf{z} \cdot_2 1_2) = (1_1 \cdot_1 \mathbf{z}) \cdot_2 (\mathbf{y} \cdot_1 1_1) = \mathbf{z} \cdot_2 \mathbf{y}$$

Therefore, the mutual product is commutative. Thus, we may define a translation $\mathfrak{T}_2 : \mathcal{L}_{\text{ComMon}} \rightarrow \mathcal{L}_{\text{Mon}} \otimes \mathcal{L}_{\text{Mon}}$ by $\mathfrak{T}_2(\mathbf{x} \cdot \mathbf{y}) := \mathbf{x} \cdot_1 \mathbf{y}$, $\mathfrak{T}_2(1) := 1_1$. The monoid equations hold in \mathcal{L}_{Mon} , and, as we have just shown, the commutativity equation also holds.

Clearly $\mathfrak{T}_1 \circ \mathfrak{T}_2(t) = t$. In the other direction, $\mathfrak{T}_2 \circ \mathfrak{T}_1(t) = t$, by relabeling all monoid multiplication and units in t with 1. Therefore $\mathcal{L}_{\text{Mon}} \otimes \mathcal{L}_{\text{Mon}} \cong \mathcal{L}_{\text{ComMon}}$.

Take \mathcal{L}^1 , \mathcal{L}^2 , ε_1 and ε_2 to be \mathcal{L}_{Mon} , \mathcal{L}_{Mon} , σ_{Mon} and \emptyset , respectively. Then $\mathcal{L}_{\varepsilon_1}^1$ is \mathcal{L}_{Mon} ; as the theory of monoids is consistent, $\mathcal{L}_{\varepsilon_2}^2$ is $\mathbf{Pres}_{\lambda}^{\text{op}}\mathbf{Set}$; and $\mathcal{L} := \mathcal{L}^1 \otimes \mathcal{L}^2$ is $\mathcal{L}_{\text{ComMon}}$. Consequently,

$$\mathcal{L}_{\varepsilon_1}^1 \otimes \mathcal{L}_{\varepsilon_2}^2 = \mathcal{L}_{\text{Mon}} \otimes \mathbf{Pres}_{\lambda}^{\text{op}}\mathbf{Set} \cong \mathcal{L}_{\text{Mon}} \not\cong \mathcal{L}_{\text{ComMon}} = \mathcal{L}_{\varepsilon_1 + \varepsilon_2}$$

Therefore, the analogous conservativity result for tensor does not hold. \square

The merit of the previous example lies in the familiarity of the Eckmann-Hilton argument. However, it has no evident computational interpretation, as the tensor is not usually used for combining the theory for monoids, equivalently, the list monad, with other theories, let alone for combining it with itself. We therefore present an example with a computational interpretation.

Example 13-2. Let \mathbb{V} , $|\mathbb{V}| \geq 2$ be a finite set denoting storable values. We show that $\mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \mathcal{L}_{\text{ND}} \rightarrow \mathcal{L}_{\text{GS}(\mathbb{V})} \otimes \mathcal{L}_{\text{ND}}$ is not conservative, even though $\mathcal{L}_{\text{OW}(\mathbb{V})} \rightsquigarrow \mathcal{L}_{\text{GS}(\mathbb{V})}$ and $\mathcal{L}_{\text{ND}} \rightsquigarrow \mathcal{L}_{\text{ND}}$ are conservative.

Consider the following $\sigma_{\text{OW}(\mathbb{V})} + \sigma_{\text{ND}}$ -equation:

$$\mathbf{x} \vee \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} = \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} \quad (13.1)$$

Our notation is well-defined as \mathcal{L}_{ND} proves \vee to be associative and commutative, and as \mathbb{V} is non-empty and finite. This equation holds in $\mathcal{L}_{\text{GS}(\mathbb{V})} \otimes \mathcal{L}_{\text{ND}}$:

$$\begin{aligned} \mathbf{x} \vee \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} &\stackrel{\mathcal{L}_{\text{GS}(\mathbb{V})}}{\downarrow} = \text{lookup} \left\langle \bigvee_{v' \in \mathbb{V}} \boxed{\text{update}_{v'} \left(\mathbf{x} \vee \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} \right)} \right\rangle \\ &\stackrel{\otimes\text{-equations}}{\downarrow} = \text{lookup} \left\langle \bigvee_{v' \in \mathbb{V}} \left(\text{update}_{v'} \mathbf{x} \right) \vee \bigvee_{v \in \mathbb{V}} \boxed{\text{update}_{v'} (\text{update}_v \mathbf{x})} \right\rangle \\ &\stackrel{\mathcal{L}_{\text{GS}(\mathbb{V})}}{\downarrow} = \text{lookup} \left\langle \bigvee_{v' \in \mathbb{V}} \boxed{\left(\text{update}_{v'} \mathbf{x} \right) \vee \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x}} \right\rangle \\ &\stackrel{\vee \text{ commutativity and absorption}}{\downarrow} = \text{lookup} \left\langle \bigvee_{v' \in \mathbb{V}} \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} \right\rangle \\ &\stackrel{\text{lookup absorption}}{\downarrow} = \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} \end{aligned}$$

Equation (13.1) does *not* hold in $\mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \mathcal{L}_{\text{ND}}$. Indeed, consider the free model over $\mathbb{1}$, $\mathcal{P}_+^{\mathbb{N}^0}((\mathbb{1} + \mathbb{V}) \times \mathbb{1}) \cong \mathcal{P}_+^{\mathbb{N}^0}(\mathbb{1} + \mathbb{V})$. The interpretations of both sides of Equation (13.1) are

$$\left[\left[\mathbf{x} \vee \bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} \right] \right] = \{\iota_1 \star\} \cup \iota_2[\mathbb{V}] \neq \iota_2[\mathbb{V}] = \left[\left[\bigvee_{v \in \mathbb{V}} \text{update}_v \mathbf{x} \right] \right]$$

Thus $\mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \mathcal{L}_{\text{ND}} \rightarrow \mathcal{L}_{\text{GS}(\mathbb{V})} \otimes \mathcal{L}_{\text{ND}}$ is not conservative. \square

Equation (13.1) has a natural computational interpretation: the action of a computation that writes *all* possible states and also does not change the state is subsumed by a computation that simply writes out all possible states. If the memory cell we model has additional states that are not captured by \mathbb{V} , such as reserved bits only accessible

via the built-in constants, then we should carefully consider the validity of this equation, as a pure computation may leave these additional states unchanged, but the other values may affect them.

We do not have a general condition on an arbitrary pair of theories guaranteeing

$$\mathcal{L}_{\varepsilon_1 + \varepsilon_2} \cong \mathcal{L}_{\varepsilon_1}^1 \otimes \mathcal{L}_{\varepsilon_2}^2$$

However, in our example language we only tensor with the three theories for interacting with a global memory cell. Therefore, as an intermediate remedy, we show that they possess the required property, apart from the restriction $\mathcal{L}_{\text{OW}(\mathbb{V})} \mapsto \mathcal{L}_{\text{GS}(\mathbb{V})}$.

Lemma 13.8. *Let \mathbb{V} , $|\mathbb{V}| \geq 2$ be a finite set. Let \mathcal{L}'_1 be one of $\mathcal{L}_{\text{GS}(\mathbb{V})}$, $\mathcal{L}_{\text{Env}(\mathbb{V})}$, and $\mathcal{L}_{\text{OW}(\mathbb{V})}$.*

- *For every injective translation $\mathfrak{T}_2 : \mathcal{L}_2 \mapsto \mathcal{L}'_2$, the tensored translation is also injective, i.e.,*

$$\mathcal{L}'_1 \otimes \mathfrak{T}_2 : \mathcal{L}'_1 \otimes \mathcal{L}_2 \mapsto \mathcal{L}'_1 \otimes \mathcal{L}'_2$$

- *Let \mathcal{L}_1 be a conservative restriction of \mathcal{L}'_1 to a sub-signature*

$$\varepsilon \subseteq \{\text{lookup} : \mathbb{V}, \text{update} : \mathbb{1} \langle \mathbb{V} \rangle\}, \quad \varepsilon \neq \{\text{update}\}$$

Let $\mathfrak{T}_1 : \mathcal{L}_1 \mapsto \mathcal{L}'_1$ be the induced injective translation given by $\mathfrak{T}_1(t) = t$. For every theory \mathcal{L}'_2 , the tensored translation is also injective, i.e.,

$$\mathfrak{T}_1 \otimes \mathfrak{T}_2 : \mathcal{L}_1 \otimes \mathcal{L}'_2 \mapsto \mathcal{L}'_1 \otimes \mathcal{L}'_2$$

Proof

Let $\mathfrak{T}_2 : \mathcal{L}_2 \mapsto \mathcal{L}'_2$ be any injective translation, and let $m_2 : T_2 \rightarrow T'_2$ be its corresponding monad morphism.

Recall from Examples 12-4–12-6 the three monad morphisms corresponding to tensoring with \mathcal{L}'_1 :

$$\begin{aligned} T_{\text{GS}(\mathbb{V})} \otimes m_2 & : (T_2(\mathbb{V} \times -))^{\mathbb{V}} \xrightarrow{m_2^{\mathbb{V}}} (T'_2(\mathbb{V} \times -))^{\mathbb{V}} \\ T_{\text{Env}(\mathbb{V})} \otimes m_2 & : (T_2(\quad -))^{\mathbb{V}} \xrightarrow{m_2^{\mathbb{V}}} (T'_2(\quad -))^{\mathbb{V}} \\ T_{\text{OW}(\mathbb{V})} \otimes m_2 & : T_2((\mathbb{1} + \mathbb{V}) \times -) \xrightarrow{m_2} T'_2((\mathbb{1} + \mathbb{V}) \times -) \end{aligned}$$

The first two morphisms are injective as exponentials of injective functions are also injective.

Next, consider any theory \mathcal{L}'_2 whose corresponding monad is T'_2 . If \mathcal{L}'_2 is inconsistent, the statement follows trivially. Therefore, assume \mathcal{L}'_2 consistent, and therefore η is injective. Below, we will make use of the fact that all monads over **Set** preserve injections [AMBL12, Lemma IV.1].

Because $\mathcal{L}_{\text{Env}(\mathbb{V})}$ and $\mathcal{L}_{\text{OW}(\mathbb{V})}$ are conservative restrictions of $\mathcal{L}_{\text{GS}(\mathbb{V})}$, and because the tensor \otimes is bifunctorial, it suffices to prove the statement for the following three monad morphisms:

$$\begin{array}{ccc}
 & & T_{\text{GS}(\mathbb{V})} \\
 & \nearrow^{m_{\text{Env}(\mathbb{V})}} & \\
 T_{\text{Env}(\mathbb{V})} & & \\
 & \nwarrow_{\eta_{\text{Env}(\mathbb{V})}} & \\
 & & \text{id} \\
 & \nearrow_{\eta_{\text{OW}(\mathbb{V})}} & \\
 & & T_{\text{OW}(\mathbb{V})}
 \end{array}$$

The tensors with the bottom two morphisms are given by the injective tensor maps \otimes \mathbf{l}_2 :

$$\begin{aligned}
 \eta_{\text{Env}(\mathbb{V})} \otimes \mathcal{L}'_2 : T'_2 &\xrightarrow{\eta^{\mathbb{V}}} (T'_2 -)^{\mathbb{V}} \\
 \eta_{\text{OW}(\mathbb{V})} \otimes \mathcal{L}'_2 : T'_2 &\xrightarrow{\eta} T'_2((\mathbb{1} + \mathbb{V}) \times -)
 \end{aligned}$$

Recall from Example 12-7 that the tensor with the top morphism is:

$$\begin{aligned}
 m_{\text{Env}(\mathbb{V})} \otimes T'_2 : (T'_2 X)^{\mathbb{V}} &\rightarrow (T'_2(\mathbb{V} \times X))^{\mathbb{V}} \\
 \kappa &\mapsto \lambda v. (T'_2(\lambda x. \langle v, x \rangle)) (\kappa(v))
 \end{aligned}$$

Note that for every $v \in \mathbb{V}$, $\lambda x. \langle v, x \rangle$ is injective, and therefore $T'_2(\lambda x. \langle v, x \rangle)$ is injective. As $|\mathbb{V}| \geq 1$, we deduce that $m_{\text{Env}(\mathbb{V})} \otimes T'_2$ is injective. \blacksquare

Note how the argument we applied to $m_{\text{Env}(\mathbb{V})}$ fails for $m_{\text{OW}(\mathbb{V})}$. Indeed, by Example 12-7 the tensor with $m_{\text{OW}(\mathbb{V})}$ is given by:

$$\begin{aligned}
 m_{\text{OW}(\mathbb{V})} \otimes T'_2 : T'_2((\mathbb{1} + \mathbb{V}) \times X) &\rightarrow (T'_2(\mathbb{V} \times X))^{\mathbb{V}} \\
 k &\mapsto \lambda v. \left(T'_2 \left(\begin{array}{l} \langle \mathbf{l}_1 \star, x \rangle \mapsto \langle v, x \rangle \\ \langle \mathbf{l}_2 v_0, x \rangle \mapsto \langle v_0, x \rangle \end{array} \right) (k) \right)
 \end{aligned}$$

As the map

$$\begin{aligned}
 \langle \mathbf{l}_1 \star, x \rangle &\mapsto \langle v, x \rangle \\
 \langle \mathbf{l}_2 v_0, x \rangle &\mapsto \langle v_0, x \rangle
 \end{aligned}$$

is *not* injective, we may not proceed as with $m_{\text{Env}(\mathbb{V})}$. Indeed, as Example 13-2 shows, the conservativity result fails for this morphism.

Using Theorem 13.7 and Lemma 13.8, we establish the necessity of conditions for the validity of the optimisations. We first deal with the operation-wise valid optimisations:

Lemma 13.9. *For every op in Π , the conservative restriction $\mathcal{L}_{\{\text{op}\}}$ of \mathcal{L} is given by the conservative restriction of the component involving op . Explicitly,*

- $\mathcal{L}_{\{\text{raise}\}} \cong \mathcal{L}_{\{\text{rollback}\}} \cong \text{Th}_{\langle \{\text{raise}:0\}, \emptyset \rangle}$;
- $\mathcal{L}_{\{\text{throw}\}} \cong \mathcal{L}_{\{\text{abort}\}} \cong \text{Th}_{\langle \{\text{abort}, s:0 \mid s \in \text{Str}\}, \emptyset \rangle}$;
- $\mathcal{L}_{\{\text{lookup}^\rho\}} \cong \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{Env}(\mathbb{V})}$;
- $\mathcal{L}_{\{\text{update}^\rho\}} \cong \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{OW}(\mathbb{V})}$; and
- $\mathcal{L}_{\{\vee\}} \cong \mathcal{L}_{\text{ND}}$.

Similarly, for every pair op, op' of operations in Π , $\text{op}, \text{op}' \neq \text{update}$, the conservative restriction $\mathcal{L}_{\{\text{op}, \text{op}'\}}$ of \mathcal{L} is given by combining the conservative restrictions of the components involving op and op' .

Proof

Recall \mathcal{L} 's definition:

$$\mathcal{L}_{\{\text{raise}, \text{throw}\}} + \left(\bigotimes_{\substack{\ell \in \mathbb{L} \\ \rho \in \text{Reg}_{\text{RO}}}} \mathcal{L}_{\text{Env}(\mathbb{V})} \otimes \bigotimes_{\substack{\ell \in \mathbb{L} \\ \rho \in \text{Reg}_{\text{WO}}}} \mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \bigotimes_{\substack{\ell \in \mathbb{L} \\ \rho \in \text{Reg}_{\text{RW}}}} \mathcal{L}_{\text{GS}(\mathbb{V})} \right. \\ \left. \otimes (\mathcal{L}_{\{\text{rollback}, \text{abort}\}} + \mathcal{L}_{\{\text{input}\}} + \mathcal{L}_{\{\text{output}\}} + \mathcal{L}_{\text{ND}}) \right)$$

Let op be any operation in Π , and let \mathcal{L}_0 be the component in the combination that involves op . We combine the trivial restrictions $\mathbf{Pres}_{\mathbb{X}_0}^{\text{op}} \mapsto \mathcal{L}'$ for each component \mathcal{L}' not involving op . For these trivial restrictions, we may use Theorem 13.7 and Lemma 13.8 to deduce that $\mathcal{L}_0 \mapsto \mathcal{L}$ is conservative.

Similarly, let op, op' be a pair of different operations in Π , $\text{op}, \text{op}' \neq \text{update}$. As $\text{update} \notin \{\text{op}, \text{op}'\}$, we can apply Theorem 13.7 and Lemma 13.8 and deduce that $\mathcal{L}_{\{\text{op}, \text{op}'\}} \mapsto \mathcal{L}$ is conservative. ■

Proposition 13.10. *The validity conditions for the operation-wise valid optimisations (Proposition 13.1 and in Proposition 13.2) are necessary.*

Proof

Consider any $o \in \{\text{Discard, Pure Hoist, Hoist}\}$ and $\varepsilon \not\subseteq \zeta^o$. Then there is some $\text{op} \in \varepsilon$, satisfying $\text{op} \notin \zeta^o$. By Lemma 13.9, we know the monadic description of $T_{\{\text{op}\}}$, and a case-by-case calculation shows that if $\text{op} \notin \zeta^o$, then $T_{\{\text{op}\}}$ violates the corresponding law.

Similarly, let o be one of the Swap optimisations and consider any triple $\varepsilon_1, \varepsilon_2 \subseteq \varepsilon$ such that $\varepsilon_2 \not\subseteq \bigcap_{\text{op} \in \varepsilon_1} \zeta^o(\text{op})$. Therefore, there exist $\text{op}_2 \in \varepsilon_2$ and $\text{op}_1 \in \varepsilon_1$, such that $\text{op}_2 \notin \zeta^o(\text{op}_1)$. We need to show that $\mathcal{L}_{\{\text{op}_1, \text{op}_2\}}$ invalidates the corresponding Swap condition for op_1, op_2 . For $\text{op}_1, \text{op}_2 \neq \text{update}$, we may use Lemma 13.9 to obtain the monadic description of $T_{\{\text{op}_1, \text{op}_2\}}$, and a (lengthy!) case-by-case calculation shows that if $\text{op}_2 \notin \zeta^o(\text{op}_1)$, the Swap condition does not hold.

It remains to check the cases in which one of the optimisations is $\text{update}^{\text{p}0}$, and the other is one of $\text{update}^{\text{p}0}$, $\text{lookup}^{\text{p}0}$, raise , or throw . Note that in all these cases we may apply Theorem 13.7 and Lemma 13.8 to obtain the explicit description of $\mathcal{L}_{\{\text{op}_1, \text{op}_2\}}$, and case-by-case calculations complete the proof. ■

We proceed to deal with the non-operation-wise valid optimisations, beginning with the two Copy optimisations.

Proposition 13.11. *The conditions for the two Copy optimisations (Proposition 13.3 and Proposition 13.4) are necessary.*

Proof

We begin with the Copy optimisation. We need to show that if at least one of input, output, or \vee is in ε , or if $\{\text{lookup}^{\text{p}}, \text{update}^{\text{p}}\} \subseteq \varepsilon$ for some p , then Copy does not hold.

Note that in each of the first three cases, we may use Lemma 13.9 to calculate $\mathcal{L}_{\{\text{input}\}}$, $\mathcal{L}_{\{\text{output}\}}$, or $\mathcal{L}_{\{\vee\}}$, accordingly. As these three theories violate the algebraic characterisation (i.e., idempotency), so does every conservative extension of them. Thus, in this case, \mathcal{L}_ε violates the Copy optimisation.

Assume, therefore, that for some p , $\{\text{lookup}^{\text{p}}, \text{update}^{\text{p}}\} \subseteq \varepsilon$. Therefore, we may use Theorem 13.7 and Lemma 13.8 to deduce the injectivity of $\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\vee)} \hookrightarrow \mathcal{L}$. Therefore, $\mathcal{L}_{\{\text{lookup}^{\text{p}}, \text{update}^{\text{p}}\}} = \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\vee)}$. This theory violates the Copy optimisation.

We treat Weak Copy analogously, excluding the case $\vee \in \varepsilon$. ■

Proposition 13.12. *The condition for the Unique optimisations (Proposition 13.5) is necessary.*

Proof

Consider any ε that does not satisfy the condition in Proposition 13.5, i.e., either $\{\text{raise, rollback}\} \subseteq \varepsilon$ or $\varepsilon \cap \{\text{raise, rollback}\} = \{\text{op}\}$ but $\varepsilon \not\subseteq \zeta^{\text{Swap}}(\text{op})$.

Assume $\{\text{raise, rollback}\} \subseteq \varepsilon$. We apply Lemma 13.9 and deduce that

$$\mathcal{L}_{\{\text{raise, rollback}\}} = \mathcal{L}_{\{\text{raise}\}} + \mathcal{L}_{\{\text{rollback}\}} = \text{Th}_{\langle \{\text{raise:0, rollback:0}\}, \emptyset \rangle}$$

Thus $\mathcal{L}_{\{\text{raise, rollback}\}}$ has two distinct constants, and therefore \mathcal{L}_ε too. Therefore, in this case, Unique optimisation is not valid.

Assume $\varepsilon \cap \{\text{raise, rollback}\} = \{\text{op}\}$ but $\varepsilon \not\subseteq \zeta^{\text{Swap}}(\text{op})$. By Proposition 13.10, the Swap optimisation does not hold for ε , $\{\text{op}\} \subseteq \varepsilon$, and therefore there exist some $\text{op}' \in \varepsilon$ such that \mathcal{L}_ε does not prove the equation:

$$\text{op}'(\text{op}, \dots, \text{op}) = \text{op}$$

Thus, the two sides of this equation constitute two different constant terms, and \mathcal{L}_ε violates the Unique optimisation. ■

13.4 Mechanised analysis

As we saw in the previous section, calculating a precise description of the conservative restriction \mathcal{L}_ε is subtle. We therefore investigate the use of mechanised assistance in analysing the combined theory \mathcal{L} directly. We construct a HASKELL data structure representing the free models and use HASKELL to analyse it.

Our approach suffers from several limitations. First, HASKELL does not have a complete formal semantics, and the implementation used is not certified to produce correct code against any formalisation of the language. Moreover, we make no attempt to neither specify nor certify our code. Therefore, there is a gap between our analysis and a formal proof of the various completeness propositions.

In addition, we make some simplifying assumptions on our model. For example, instead of studying arbitrarily many regions $\rho \in \mathbf{Reg}$, arbitrarily many locations $\ell \in \mathbb{L}$, and an arbitrary set of storable values \mathbb{V} , we use two different regions, two different locations, and the concrete $\mathbb{V} = \text{Bool}$ to represent our storable values. We leave the rigorous justification of such simplifying assumptions to further work. We state these simplifying assumptions explicitly as we encounter them.

HASKELL [HHPJW07] has an expressive type system, importantly, it has *type classes* [WB89], which help us manage the complexity of constructing the model and its associated operations. In addition, HASKELL has library support and syntactic constructs for manipulating data structures representing monads which also ease our model construction.

We present the code in its entirety in *literate programming* style [Knu84]. This code was developed with the Glasgow Haskell Compiler (v. 7.4.2) with the following extensions enabled:

- *TypeSynonymInstances*;
- *FlexibleInstances*;
- *FlexibleContexts*;
- *OverlappingInstances* ;
- *RankNTypes*; and
- *ImpredicativeTypes*

We use following standard libraries:

```
import Data.Char
import Data.Maybe
import Data.Monoid
import Data.List
import Text.Printf
```

We also use the *monad transformer library* [Jon95] (mtl v. 2.1.1) and the library for free monads (free v. 3.4.1):

```
import Control.Monad.Error
import Control.Monad.Identity
import Control.Monad.State
import Control.Monad.Trans.Free
import Control.Monad.Trans.Reader
import Control.Monad.Trans.Writer
```

Represting $T_{\mathcal{L}}$

We begin by defining a datatype representation of the monad corresponding to the theory

$$\mathcal{L}_{\{\text{raise,throw}\}} + \left(\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{Env}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\mathbb{V})} \right. \\ \left. \begin{array}{ccc} \rho \in \mathbf{Reg}_{\text{RO}} & \rho \in \mathbf{Reg}_{\text{WO}} & \rho \in \mathbf{Reg}_{\text{RW}} \end{array} \right) \\ \otimes (\mathcal{L}_{\{\text{rollback,abort}\}} + \mathcal{L}_{\{\text{input}\}} + \mathcal{L}_{\{\text{output}\}} + \mathcal{L}_{\text{ND}})$$

Our model needs to be a monad in which we can test equality.

```
class (Monad m) ⇒ EqMonad m where
  eqLift :: Eq a ⇒ m a → m a → Bool
instance (Eq a, EqMonad m) ⇒ Eq (m a) where
  (≡) = eqLift
```

We will also need to iterate over certain types, crucially over all arity types.

```
class WithRange a where
  range :: [a]
instance WithRange () where
  range = [()]
instance WithRange Bool where
  range = [False, True]
instance (WithRange a, WithRange b) ⇒ WithRange (a, b) where
  range = [(x, y) | x ← range, y ← range]
```

We also model algebraic operations and generic effects.

```
type AlgOp m a p = Monad m ⇒ ∀c. (a → m c) → (p → m c)
type GenOp m a p = Monad m ⇒ p → m a
```

Note that HASKELL's type system is not strong enough to ensure all instances of $\text{AlgOp } m a p$ are in fact algebraic operations of type $a \langle p \rangle$ for m .

We will use the bijection between algebraic operations and generic effects (Theorem 2.4):

```
op :: Monad m ⇒ GenOp m a p → AlgOp m a p
op gen k p = gen p >>= k
gen :: Monad m ⇒ AlgOp m a p → GenOp m a p
gen op = op (return)
```


We will also use monad morphisms to map the various operations between monads (Theorem 2.8).

```

type MonadMorphism m m' = (Monad m, Monad m') => ∀a.(m a → m' a)
mapGen :: (Monad m, Monad m') =>
  MonadMorphism m m' → GenOp m a p → GenOp m' a p
mapGen mor gen = mor ∘ gen
mapOp :: (Monad m, Monad m') =>
  MonadMorphism m m' → AlgOp m a p → AlgOp m' a p
mapOp mor mop = op (mapGen mor (gen mop))

```

We now implement the various monads and monad morphisms involved in our construction.

The identity monad

Even when the monad is completely defined by the standard libraries, we need to show it allows equality testing.

```

instance EqMonad Identity where
  eqLift a b = (runIdentity a) ≡ (runIdentity b)

```

Exceptions

We use the exception monad transformer *ErrorT* to sum the theories for exceptions with other theories (Example 12-3).

```

instance (EqMonad m, Eq e, Error e) => EqMonad (ErrorT e m) where
  eqLift a b = (runErrorT a) ≡ (runErrorT b)

```

We will use the following two monads (and their transformers):

```

instance Error () where
  noMsg = ()

instance Error Str where
  noMsg = "Default exception." -- Required by the Error type class.

type T{raise} = Either ()
type T{throw} = Either Str

```

To implement the algebraic operations for these monads, we need an empty type.

```
data Empty
instance Eq Empty where
    (≡) a b = True

instance WithRange Empty where
    range = []
```

We will also need to eliminate the empty type. Unfortunately, this version of HASKELL does not support empty pattern matches, therefore, we use HASKELL's built-in exceptions.

```
whatever :: Empty → a
whatever = (λz →
    error$ "GHC bug #2431" ++
    "http://hackage.haskell.org/trac/ghc/ticket/2431")
```

We make use of the injections $T_{\{\text{raise}\}} \rightarrow T_{\{\text{raise}\}} + T \leftarrow T$, and similarly for throw. The injection ι_2 is the *lift* function given by the *ErrorT* monad transformer. For ι_1 , we define:

```
 $\iota_1^{\text{raise}} :: \text{Monad } m \Rightarrow \text{MonadMorphism } T_{\{\text{raise}\}} (\text{ErrorT } () m)$ 
 $\iota_1^{\text{raise}} = \text{ErrorT} \circ \text{return}$ 
 $\iota_1^{\text{throw}} :: \text{Monad } m \Rightarrow \text{MonadMorphism } T_{\{\text{throw}\}} (\text{ErrorT } \text{String } m)$ 
 $\iota_1^{\text{throw}} = \text{ErrorT} \circ \text{return}$ 
```

Finally, we define the operations:

```
errorRaise :: GenOp T_{raise} Empty ()
errorRaise () = Left ()

errorThrow :: GenOp T_{throw} Empty Str
errorThrow s = Left s

errorRaiseOp = op errorRaise
errorThrowOp = op errorThrow
```

Global state

It is crucial that we model memory locations using a small number of values. By having a store type \mathbb{S} with a small range of values, we avoid a state space explosion in

our model. This is a simplifying assumption on our model that further work needs to justify.

```

type  $\mathbb{V}$  = Bool
type  $\mathbb{L}$  = Bool
type  $\mathbb{S}$  =  $\mathbb{L} \rightarrow \mathbb{V}$ 
instance Eq  $\mathbb{S}$  where
    ( $\equiv$ )  $s\ s' = (s\ True \equiv s'\ True) \wedge (s\ False \equiv s'\ False)$ 

```

We use an instance of the global state monad transformer *StateT* to tensor the theory of global state with other theories (Example 12-4).

```

type  $T_{GS} \otimes = StateT\ \mathbb{S}$ 
instance (EqMonad  $m$ )  $\Rightarrow$  EqMonad ( $T_{GS} \otimes m$ ) where
    eqLift  $a\ b = helper\ (runStateT\ a)\ (runStateT\ b)$ 
where
    helper  $x\ y = all\ (\lambda v_0 \rightarrow$ 
        all ( $\lambda v_1 \rightarrow$ 
            let  $s = (\lambda \ell \rightarrow \mathbf{if}\ \ell\ \mathbf{then}\ v_1\ \mathbf{else}\ v_0)$ 
                in ( $x\ s \equiv y\ s$ ) range) range)
type  $T_{GS} = T_{GS} \otimes Identity$ 

```

We implement the monad morphisms $\mathfrak{t}_1^{\otimes} : T_{GS(\mathbb{V})} \rightarrow T_{GS(\mathbb{V})} \otimes T$.

```

 $\mathfrak{t}_1^{\otimes GS(\mathbb{V})} :: Monad\ m \Rightarrow MonadMorphism\ T_{GS}\ (T_{GS} \otimes m)$ 
 $\mathfrak{t}_1^{\otimes GS(\mathbb{V})}\ k = StateT\ (\lambda s \rightarrow \mathbf{let}\ r = runStateT\ k\ s$ 
    in  $return\ \circ\ runIdentity\ \$\ r)$ 

```

Finally, the operations:

```

ref :: GenOp  $T_{GS}\ \mathbb{V}\ \mathbb{L}$ 
ref  $\ell = StateT\ (\lambda s \rightarrow return\ (s\ (\ell), s))$ 
set :: GenOp  $T_{GS}\ ()\ (\mathbb{L}, \mathbb{V})$ 
set ( $\ell_0, v_0$ ) = StateT ( $\lambda s \rightarrow return\ (((), \lambda \ell \rightarrow \mathbf{if}\ (\ell \equiv \ell_0)$ 
    then  $v_0$ 
    else  $s\ (\ell))$ )
lookupOp = op ref
updateOp = op set

```

Read-only memory

Implementing read-only memory is nearly identical to implementing global state.

```

type TEnv⊗ = ReaderT S
instance EqMonad m ⇒ EqMonad (TEnv ⊗ m) where
  eqLift a b = helper (runReaderT a) (runReaderT b)
  where helper x y = all (λv0 →
    all (λv1 →
      let s = (λℓ → if ℓ then v1 else v0)
      in (x s) ≡ (y s)) range) range

type TEnv = TEnv ⊗ Identity
t1⊗Env :: Monad m ⇒ MonadMorphism TEnv (TEnv ⊗ m)
t1⊗Env k = ReaderT (λs → let r = runReaderT k s
  in return ∘ runIdentity $ r)

refRO :: GenOp TEnv ∨ ⊔
refRO ℓ = ReaderT (λs → return (s (ℓ)))
lookupROOp = op refRO

```

Write-only memory

We use HASKELL's monoid library. First, we represent the overwrite monoid:

```

type MOW = ⊔ → Maybe ∨
instance Eq MOW where
  (≡) a b = all (λℓ → (a ℓ) ≡ (b ℓ)) range
instance Monoid (MOW) where
  mempty = λℓ → Nothing
  mappend δ1 δ2 = λℓ → case δ2 ℓ of
    Nothing → δ1 ℓ
    Just v   → Just v

```

We now define the write-only monad transformer:

```

type TOW⊗ = WriterT MOW
instance EqMonad m ⇒ EqMonad (TOW ⊗ m) where

```

$$eqLift\ a\ b = (runWriterT\ a) \equiv (runWriterT\ b)$$

type $T_{OW} = T_{OW} \otimes Identity$

$\mathfrak{l}_1^{\otimes OW} :: Monad\ m \Rightarrow MonadMorphism\ T_{OW}\ (T_{OW} \otimes m)$

$\mathfrak{l}_1^{\otimes OW} = WriterT \circ return \circ runIdentity \circ runWriterT$

$setWO :: GenOp\ T_{OW}\ ()\ (\mathbb{L}, \mathbb{V})$

$setWO\ (\ell_0, v_0) = WriterT \$ return\ ((), \lambda \ell \rightarrow \mathbf{if}\ (\ell \equiv \ell_0)$

then *Just* v_0

else *Nothing*)

$updateWOp = op\ setWO$

I/O

We use HASKELL's representation of characters and strings. To make computations tractable, we only range over a small number of characters. This is a simplifying modelling assumption.

type $Char = Char$

instance *WithRange* $Char$ **where**

$range = ['a' .. 'z']$

type $Str = String$

We will need to range over strings. We assume it is enough to check two different strings. We leave to further work justifying this modelling assumption.

instance *WithRange* $String$ **where**

$range = ["DivideByZero", "RuntimeException"]$

We use Haskell's implementation of free monads. We define the following signature functor:

data $\Sigma_{IO}\ a = Input\ (Char \rightarrow a)$

| $Output\ (Char, a)$

instance *Functor* Σ_{IO} **where**

$fmap\ f\ (Input\ g) = Input\ (f \circ g)$

$fmap\ f\ (Output\ (c, a)) = Output\ (c, f(a))$

instance $Eq\ a \Rightarrow Eq\ (\Sigma_{IO}\ a)$

We lift equality testing to the carrier of the free monad corresponding to Σ_{IO} :

```
instance (Eq a, Eq b) => Eq (FreeF  $\Sigma_{\text{IO}}$  a b) where
  ( $\equiv$ ) (Pure a)           (Pure a')           = a  $\equiv$  a'
  ( $\equiv$ ) (Free (Input g))  (Free (Input h))  = all ( $\lambda$ a  $\rightarrow$ 
                                                    (g a)  $\equiv$  (h a)) range
  ( $\equiv$ ) (Free (Output (c, g))) (Free (Output (c', g'))) = (c  $\equiv$  c')  $\wedge$  (g  $\equiv$  g')
  ( $\equiv$ ) _                 _                 = False
```

With everything in place, we can sum a representation of a monad with the I/O monad by using the appropriate free monad transformer:

```
type TI/O+ = FreeT  $\Sigma_{\text{IO}}$ 
instance (EqMonad m) => EqMonad (TI/O+ m) where
  eqLift a b = (runFreeT a)  $\equiv$  (runFreeT b)

type TI/O = TI/O+ Identity
 $\iota_1^{\text{IO}} :: \text{Monad } m \Rightarrow \text{MonadMorphism } T_{\text{I/O}} (T_{\text{I/O}}+ m)$ 
 $\iota_1^{\text{IO}} k = \text{helper } (\text{runIdentity } (\text{runFreeT } k))$ 

where
  helper (Pure a)           = FreeT (return (Pure a))
  helper (Free (Input g))  = FreeT (return (Free
                                          (Input ( $\iota_1^{\text{IO}} \circ g$ ))))
  helper (Free (Output (c, g))) = FreeT (return (Free (Output (c,  $\iota_1^{\text{IO}}$  g))))

ioInputOp :: AlgOp TI/O Char ()
ioInputOp =  $\lambda$ k _  $\rightarrow$  FreeT (return (Free (Input k)))

ioOutputOp :: AlgOp TI/O () Char
ioOutputOp =  $\lambda$ k c  $\rightarrow$  FreeT (return (Free (Output (c, k ())))))
```

Non-determinism

We represent the finite powerset monad using HASKELL lists.

```
type  $\mathcal{P}_+^{\text{K}_0} = []$ 
instance EqMonad ( $\mathcal{P}_+^{\text{K}_0}$ ) where
  eqLift a b = (all ( $\lambda$ x  $\rightarrow$  elem x b) a)  $\wedge$ 
              (all ( $\lambda$ y  $\rightarrow$  elem y a) b)
```

$$\begin{aligned}
ndToss &:: GenOp \mathcal{P}_+^{\mathbb{N}^0} Bool () \\
ndToss _ &= [True, False] \\
ndChoiceOp &:: AlgOp \mathcal{P}_+^{\mathbb{N}^0} Bool () \\
ndChoiceOp &= op ndToss
\end{aligned}$$

The model

Everything is in place to represent our theory

$$\begin{aligned}
\mathcal{L}_{\{\text{raise,throw}\}} + & \left(\bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{Env}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{OW}(\mathbb{V})} \otimes \bigotimes_{\ell \in \mathbb{L}} \mathcal{L}_{\text{GS}(\mathbb{V})} \right. \\
& \left. \begin{array}{ccc} \rho \in \mathbf{Reg}_{\text{RO}} & \rho \in \mathbf{Reg}_{\text{WO}} & \rho \in \mathbf{Reg}_{\text{RW}} \end{array} \right) \\
& \otimes (\mathcal{L}_{\{\text{rollback,abort}\}} + \mathcal{L}_{\{\text{input}\}} + \mathcal{L}_{\{\text{output}\}} + \mathcal{L}_{\text{ND}})
\end{aligned}$$

type Model

$$\begin{aligned}
&= ErrorT () (ErrorT \text{Str} \\
&\quad (T_{\text{Env}} \otimes (T_{\text{Env}} \otimes (T_{\text{OW}} \otimes (T_{\text{OW}} \otimes (T_{\text{GS}} \otimes (T_{\text{GS}} \otimes \\
&\quad\quad (ErrorT () (ErrorT \text{Str} (T_{\text{I/O}} + \mathcal{P}_+^{\mathbb{N}^0}))))))))))
\end{aligned}$$

Note that to avoid a state space explosion, we only consider two regions of each kind of memory. We leave the justification of this modelling assumption to further work.

We lift the various operations along the monad morphisms. The following \uparrow notation will make the process succinct.

$$\begin{aligned}
\uparrow &:: (Monad\ m, MonadTrans\ t) \Rightarrow AlgOp\ m\ a\ p \rightarrow AlgOp\ (t\ m)\ a\ p \\
\uparrow a &= mapOp\ lift\ a
\end{aligned}$$

$$\begin{aligned}
raiseOp &:: AlgOp\ Model\ Empty\ () \\
raiseOp &= mapOp\ \iota_1^{\text{raise}}\ errorRaiseOp \\
throwOp &:: AlgOp\ Model\ Empty\ \text{Str} \\
throwOp &= \uparrow \$ \quad mapOp\ \iota_1^{\text{throw}}\ errorThrowOp \\
lookupROOp1 &:: AlgOp\ Model\ \mathbb{V}\ \mathbb{L} \\
lookupROOp1 &= \uparrow \$ \uparrow \$ \quad mapOp\ \iota_1^{\otimes \text{Env}}\ lookupROOp \\
lookupROOp2 &:: AlgOp\ Model\ \mathbb{V}\ \mathbb{L}
\end{aligned}$$


```

checkWeakIdempotency:: Eq a ⇒ AlgOp Model a () → Bool
checkWeakIdempotency mop =
  let t1 = mop (λi → mop (λj → return (i)) ()) () in
  let t2 = mop (λk → return (k)) () in
  t1 ≡ t2

```

We now supply this function with the three terms from the proof of Proposition 13.11. The crucial difference is that this time we calculate both sides of the idempotency law in the full monad $T_{\mathcal{L}}$, rather than its conservative restrictions.

```

> checkWeakIdempotency inputOp
  False
> checkWeakIdempotency (λx () → outputOp x 'a')
  False
> let ℓ0 = True
  in checkWeakIdempotency (λx () →
    lookupOp1 (λi →
      updateOp1 (\_ → x ())
        (ℓ0, ¬ i))
    (ℓ0))
  False

```

Thus the condition for Weak Copy is necessary.

To show the necessity of the criterion for Copy, we can use the previous three calculations, together with an additional calculation for non-deterministic choice:

```

checkIdempotency:: Eq a ⇒ AlgOp Model a () → Bool
checkIdempotency mop =
  let t1 = mop (λi → mop (λj → return (i,j)) ()) () in
  let t2 = mop (λk → return (k,k)) () in
  t1 ≡ t2

```

```

> checkIdempotency choiceOp
  False

```

Analysing Unique

Consider the necessity proof of the Unique condition (Proposition 13.12). The only part of the proof that relies on conservative restriction is the case in which we consider an ε containing *both* raise and rollback. The other case reduces to the necessity of the Swap condition.

Therefore, we only need to show that raise and rollback have different interpretations in our model:

```
> let t1 :: Model Empty
    t1 = raiseOp whatever () in
let t2 = rollbackOp whatever () in
    t1 ≡ t2

False
```

Analysing operation-wise valid optimisations

We turn to establish the completeness of our optimisation tables (Proposition 13.10). Here we can automate our analysis even further by systematically substituting all effect operations into each algebraic law.

However, the effect operations have different types. In particular, we cannot easily store the operations in data structures. We therefore introduce an enumeration of the various effects:

```
data OpName = In      | Out      | Raise   | Throw
             | Abort   | Rollback | Lookup1 | Lookup2
             | Update1 | Update2 | RLookup1 | RLookup2
             | WUpdate1 | WUpdate2 | Choice

deriving (Show, Enum, Eq, Bounded)
allOps = [minBound..maxBound]
```

To manipulate the various algebraic laws we introduce *term continuations*:

```
type TermCont b = ∀a.(Eq a, WithRange a) ⇒ (AlgOp Model a () → b)
```

Note that we require the parameter to be the unit type $()$.

The following helper function converts a parameterised operation into a family of terms.

$$\begin{aligned} \text{deparameterise} &:: (\text{Eq } a, \text{WithRange } p, \text{WithRange } a) \Rightarrow \\ &\quad \text{TermCont } b \rightarrow (\text{AlgOp Model } a \ p) \rightarrow [b] \\ \text{deparameterise law mop} &= \text{map } (\lambda p \rightarrow \text{law } (\lambda x _ \rightarrow \text{mop } x \ p)) \ \text{range} \end{aligned}$$

We call the act of invoking a continuation with a given effect operation *dispatching*:

$$\begin{aligned} \text{dispatch} &:: \text{TermCont } b \rightarrow \text{OpName} \rightarrow [b] \\ \text{dispatch law In} &= \text{deparameterise law inputOp} \\ \text{dispatch law Out} &= \text{deparameterise law outputOp} \\ \text{dispatch law Raise} &= \text{deparameterise law raiseOp} \\ \text{dispatch law Throw} &= \text{deparameterise law throwOp} \\ \text{dispatch law Rollback} &= \text{deparameterise law rollbackOp} \\ \text{dispatch law Abort} &= \text{deparameterise law abortOp} \\ \text{dispatch law Lookup1} &= \text{deparameterise law lookupOp1} \\ \text{dispatch law Lookup2} &= \text{deparameterise law lookupOp2} \\ \text{dispatch law RLookup1} &= \text{deparameterise law lookupROOp1} \\ \text{dispatch law RLookup2} &= \text{deparameterise law lookupROOp2} \\ \text{dispatch law Update1} &= \text{deparameterise law updateOp1} \\ \text{dispatch law Update2} &= \text{deparameterise law updateOp2} \\ \text{dispatch law WUpdate1} &= \text{deparameterise law updateWOOOp1} \\ \text{dispatch law WUpdate2} &= \text{deparameterise law updateWOOOp2} \\ \text{dispatch law Choice} &= \text{deparameterise law choiceOp} \end{aligned}$$

Note we make another crucial simplifying assumption. When we deparameterise Str , we consider only two different exceptions. As we will be iterating many times over this list, it is crucial it is finite and short.

Analysing Discard, Pure Hoist and Hoist

We capture the algebraic laws characterising Discard, Pure Hoist, and Hoist by the following type of term continuation:

$$\begin{aligned} \text{type SimpleLaw} &= \text{TermCont Bool} \\ \text{absorption} &:: \text{SimpleLaw} \\ \text{absorption mop} &= (\text{mop } (\lambda i \rightarrow \text{return } ()) \ ()) \equiv \text{return } () \\ \text{pureHoist} &:: \text{SimpleLaw} \\ \text{pureHoist mop} &= \text{any } (\lambda j \rightarrow \text{mop } (\lambda i \rightarrow \text{return } i) \ ()) \equiv \text{return } j \ \text{range} \end{aligned}$$

```

hoist :: SimpleLaw
hoist mop = (pureHoist mop) ∨ (mop (λi → return (True,i)) ())
           ≡
           mop (λi → return (False,i)) ()

```

Given any such a simple law, we generate its optimisation table:

```

optTable :: SimpleLaw → [OpName]
optTable law = filter (λname →
                       all id ( -- typechecker needs help instantiating
                               (dispatch :: SimpleLaw → OpName → [Bool])
                               law name))
                       allOps

```

The following interaction validates Proposition 13.10 for Discard, Pure Hoist and Hoist:

```

> optTable absorption
  [Lookup1,Lookup2,RLookup1,RLookup2,Choice]
> optTable pureHoist
  []
> optTable hoist
  [Raise,Throw,Abort,Rollback]

```

Note that the lists we computed correspond precisely to the optimisation tables in Proposition 13.1. Therefore, any operation *not* present in these tables invalidates the corresponding algebraic law in our model, hence the condition given by the optimisation tables is necessary.

Analysing the Swap optimisations

Finally, we consider the three swap optimisations. The corresponding algebraic laws are parameterised by two terms:

```

type SwapLaw = TermCont SimpleLaw

swap      :: SwapLaw
swap mop mop' = (mop (λi → mop' (λj → return (i,j)) ()) ())
                ≡
                (mop' (λj → mop (λi → return (i,j)) ()) ())

weakSwap :: SwapLaw

```


allOps \\ isw)))

allsets

The following interaction validates Proposition 13.10 for the Swap optimisations. By computing the swap sets mechanically we found our previously published swap sets [KP12] were incorrect. The correct swap sets state that output and input do not satisfy Isolated Swap with themselves, and non-deterministic choice does in fact completely commute with both abort and throw.

We use the following ad-hoc pretty printer to display the swapsets

```

printSwapSets :: IO ()
printSwapSets =
  do {
    mapM (λ(op, (sw, wsw, isw, none)) →
      do {
        printf "%-8s: swap = " (show op);
        printOpList "          " sw;
        putStr "      weakSwap = ";
        printOpList "          " wsw;
        putStr "      isoSwap = ";
        printOpList "          " isw;
        putStr "      none = ";
        printOpList "          " none })
    swapsets :: IO [()];
    return ()}
  where n = 4
        printOpList :: String → [OpName] → IO ()
        printOpList prefix lst = if length lst ≤ n
          then putStrLn ((show lst))
          else do {printf "%s++\n%s" (show (take n lst)) prefix;
            printOpList prefix (drop n lst)}
> printSwapSets

In      : swap      = [Lookup1, Lookup2, Update1, Update2] ++
                [RLookup1, RLookup2, WUpdate1, WUpdate2]
weakSwap = [Choice]

```

```

    isoSwap    = [In]
    none       = [Out, Raise, Throw, Abort] ++
               [Rollback]
Out   : swap  = [Lookup1, Lookup2, Update1, Update2] ++
               [RLookup1, RLookup2, WUpdate1, WUpdate2]
weakSwap    = [Choice]
isoSwap     = []
none        = [In, Out, Raise, Throw] ++
               [Abort, Rollback]
Raise  : swap = [Raise, Lookup1, Lookup2, RLookup1] ++
               [RLookup2, Choice]
weakSwap   = []
isoSwap    = []
none       = [In, Out, Throw, Abort] ++
               [Rollback, Update1, Update2, WUpdate1] ++
               [WUpdate2]
Throw  : swap = [Lookup1, Lookup2, RLookup1, RLookup2] ++
               [Choice]
weakSwap   = []
isoSwap    = []
none       = [In, Out, Raise, Throw] ++
               [Abort, Rollback, Update1, Update2] ++
               [WUpdate1, WUpdate2]
Abort  : swap = [Lookup1, Lookup2, Update1, Update2] ++
               [RLookup1, RLookup2, WUpdate1, WUpdate2] ++
               [Choice]
weakSwap   = []
isoSwap    = []
none       = [In, Out, Raise, Throw] ++
               [Abort, Rollback]
Rollback : swap = [Rollback, Lookup1, Lookup2, Update1] ++
                  [Update2, RLookup1, RLookup2, WUpdate1] ++
                  [WUpdate2, Choice]
weakSwap   = []
isoSwap    = []

```

```

    none      = [In, Out, Raise, Throw] ++
               [Abort]
Lookup1 : swap = [In, Out, Raise, Throw] ++
                 [Abort, Rollback, Lookup1, Lookup2] ++
                 [Update2, RLookup1, RLookup2, WUpdate1] ++
                 [WUpdate2, Choice]
    weakSwap  = []
    isoSwap   = [Update1]
    none      = []
Lookup2 : swap = [In, Out, Raise, Throw] ++
                 [Abort, Rollback, Lookup1, Lookup2] ++
                 [Update1, RLookup1, RLookup2, WUpdate1] ++
                 [WUpdate2, Choice]
    weakSwap  = []
    isoSwap   = [Update2]
    none      = []
Update1 : swap = [In, Out, Abort, Rollback] ++
                 [Lookup2, Update2, RLookup1, RLookup2] ++
                 [WUpdate1, WUpdate2, Choice]
    weakSwap  = [Lookup1]
    isoSwap   = []
    none      = [Raise, Throw, Update1]
Update2 : swap = [In, Out, Abort, Rollback] ++
                 [Lookup1, Update1, RLookup1, RLookup2] ++
                 [WUpdate1, WUpdate2, Choice]
    weakSwap  = [Lookup2]
    isoSwap   = []
    none      = [Raise, Throw, Update2]
RLookup1 : swap = [In, Out, Raise, Throw] ++
                  [Abort, Rollback, Lookup1, Lookup2] ++
                  [Update1, Update2, RLookup1, RLookup2] ++
                  [WUpdate1, WUpdate2, Choice]
    weakSwap  = []
    isoSwap   = []
    none      = []

```



```

RLookup2 : swap = [In, Out, Raise, Throw] ++
                  [Abort, Rollback, Lookup1, Lookup2] ++
                  [Update1, Update2, RLookup1, RLookup2] ++
                  [WUpdate1, WUpdate2, Choice]

weakSwap      = []
isoSwap       = []
none          = []

WUpdate1 : swap = [In, Out, Abort, Rollback] ++
                  [Lookup1, Lookup2, Update1, Update2] ++
                  [RLookup1, RLookup2, WUpdate2, Choice]

weakSwap      = []
isoSwap       = []
none          = [Raise, Throw, WUpdate1]

WUpdate2 : swap = [In, Out, Abort, Rollback] ++
                  [Lookup1, Lookup2, Update1, Update2] ++
                  [RLookup1, RLookup2, WUpdate1, Choice]

weakSwap      = []
isoSwap       = []
none          = [Raise, Throw, WUpdate2]

Choice : swap = [Raise, Throw, Abort, Rollback] ++
                 [Lookup1, Lookup2, Update1, Update2] ++
                 [RLookup1, RLookup2, WUpdate1, WUpdate2] ++
                 [Choice]

weakSwap      = []
isoSwap       = [In, Out]
none          = []

```

Thus our model confirms the necessity of our characterising conditions, up to the simplifying assumptions we described. HASKELL's native support for monads and monad transformers, and its type class mechanism were pivotal for our model construction. In particular, during the development of the model, the type system prevented numerous errors that a less expressive type system, such as ML's, would not be able to track. However, HASKELL's type system was not expressive enough for our treatment of the operation-wise valid optimisations and their algebraic laws. Although we worked around the problem by using boilerplate code such as the *dispatch* func-

tion, we should investigate in the future whether a dependently-typed implementation, using e.g. Agda [Nor07] or the Coq [Cdt12] proof assistant, is more suitable for this purpose.

However, we also desire library support for lists and strings. We therefore use `HASKELL`, despite its limited type-system, and comment on the parts of the code that would benefit from better type support. Brady's Idris [Bra11] may be a suitable dependently-typed alternative to `HASKELL`.

To summarise, we analysed the global algebraic optimisations in a simple functional-imperative language. However, even our simple language involved more than a thousand effect sets. Such complexity cannot be addressed intuitively without error. Our theory allows a rigorous and high level treatment of these optimisations.

Chapter 14

Conclusion

But before you come to any conclusions

Try walking in my shoes

—Depeche Mode



We set out to establish the existence of a general applicable theory of Gifford-style type-and-effect systems. In the first part of the thesis, we presented a spectrum of type-and-effect models accounting for set-theoretic models, domain theoretic models, algebraic models, and logical relations model. Our general account culminated in the categorical conservative restriction construction, establishing that semantics for type-and-effect systems arise as a *property* of an algebraic model, rather than a separately specified structure. In the second part of the thesis, we presented our theory's account of the various aspects of effect-dependent optimisations, ranging from the syntax of type-and-effect systems, through its semantics, the soundness and completeness of the optimisation process, modular treatment of optimisation validity, and a use case for synthesising sound and complete validity decision procedures. We report that, once the semantic constructions were in place, each of these areas required little effort to fit within the theory, and we expect other aspects of type-and-effect systems to follow suit.

The algebraic approach provides a valuable and general point of view, resulting in: the connection between effect operations and effect sets; the conservative restriction construction; the free lifting construction; optimisation classification and discovery of new optimisations; criteria for the validity of abstract optimisations; and methods to derive the validity of optimisations for combinations of effects modularly.

The use of CBPV highlights the interplay between functional constructs and effects. We advocate its use as a fundamental lambda-calculus involving computational

effects. The categorical language greatly helped the organisation of this work, by allowing much reuse of concepts and proofs. It was also crucial in seeing the connection with Führmann’s work, and unifying it with Benton et al.’s.

Further work abounds:

Generality. Our claim for generality will be much strengthened by generalising the logical relations argument of Corollary 9.12 from its current (set-theoretic) presentation models to the (categorical) algebraic models. Doing so will tie the conservative restriction construction to the original non-hierarchical semantics. We conjecture that by imposing sufficiently strong conditions on the factorisation system $\langle \mathcal{E}, \mathcal{M} \rangle$ of the enriching category, the \mathcal{M} -morphisms behave as predicates and allow us to generalise the proof.

Going further, we would then like to generalise the constructions in Part II from the set-theoretic case to the categorical case. However, such generalisation will require more syntactic views of enriched Lawvere theories [Plo06].

Applicability. We would like to incorporate more facets of type-and-effect analysis into our theory. An immediate first step is to incorporate Plotkin and Pretnar’s *effect handlers* [PP09a, PP09b] so our theory can account for exception handlers. We foresee no problem in doing so in light of our work equipping a calculus of effect handlers with a type-and-effect system [KLO13] and Bauer and Pretnar’s type-and-effect system for the Eff programming language [BP13].

Effect reconstruction is of immediate importance. It should be possible to derive general algorithms for type-and-effect annotation. Our semantics can then be used to give semantics to such programs. Levy’s translations of call-by-value and call-by-name into CBPV could then be used to generalise call-by-value type-and-effect systems and discover novel call-by-name type-and-effect systems. Another closely related direction is *region inference* [BT01]. Bauer and Pretnar’s work on effect handlers [BP13] suggests region inference may generalise from memory accesses to arbitrary algebraic effects.

Additional effects. Another direction is to better fit additional computational effects into the algebraic theory of effects in the first step, and then into our theory of type-and-effect systems. Notions of locality, particularly local state, are very important. It may be possible to make use of work on the algebraic treatment of locality, e.g., Plotkin and Power [PP02], Melliès [Mel10], and Staton [Sta09,

Sta13b] to obtain a more general optimisation theory. This should enable incorporating the work of Benton et al. on dynamic allocation [BKBH07]. Interestingly, it should also incorporate Staton’s account for logic programming [Sta13a]. Incorporating higher-order store [BKBH09] would require solving recursive domain equations [AJ94, Lev02]. We are also very interested in accounting for parallelism, as in Gifford’s work [LG88].

We outline further work arising from each chapter:

Algebraic operations. In Chapter 2 we considered only Eilenberg-Moore CBPV models. It would be interesting to generalise our account to arbitrary CBPV models.

Models. While we used categorical language to formulate our models, we did not consider any *category of models*. It would be interesting to identify the appropriate notions of morphisms for each of our model classes. We could then formulate the relationship between our different categories of models using categorical notions like isomorphism and equivalence of categories. More speculatively, categories of models would also include categorical constructions such as limits and colimits, and would perhaps give an abstract account of our conservative restriction construction.

Lawvere theories. As we noted on page 196, there is a mismatch between presentations and Lawvere theories: presentations allow us to have arbitrarily infinite parameter types, whereas λ -Lawvere theories are restricted by the cardinal λ . It is possible that there are other equivalent descriptions of algebraic theories that avoid this mismatch. Melliès¹ suggests there might be a connection with monads with arities [BMW12].

Another shortcoming of our account is the large amount of intimidating commutative diagrams. We obtained these proofs by translating proofs devised in our own variant of string diagrams [BS11] for symmetric monoidal closed categories. As we are not aware of any standard notation for string diagrams of (symmetric) monoidal closed categories, we decided to use the widely familiar commutative diagrams. (See Selinger [Sel09] for a survey of such notations.) We leave to further work to recast our proofs in a familiar string diagrammatic form.

¹Paul-André Melliès, private communication, 2013.

In this context, Fiore² suggested to us to use *monoidal actions* instead of enrichment. Fiore conjectures that monoidal actions produce a more pleasant theory of generalised Lawvere theories that does not involve as complicated commutative diagrams. We leave the investigation of his suggestion to further work.

Atkey used parameterised monads [Atk09b] to account for type-and-effect systems, and more generally, capabilities [Atk09a]. It would be interesting to investigate whether parameterised monads have a parameterised notion of Lawvere theories, and whether these can be used to extend our theory to the parameterised case.

Algebraic models. Our construction of a factorisation system of enriched Lawvere theories from a given factorisation in the enriching category is unsatisfactory, as we do not have an explicit characterisation of the left orthogonality class of \mathcal{M}^{law} . Our free lifting construction suggests this class may have to coincide with its subclass \mathcal{E}^{law} of all morphisms whose A, \llbracket components are \mathcal{E} -morphisms, as is certainly the case in **Set**. Thus we are interested to find out whether \mathcal{E}^{law} may be a proper subclass of the left orthogonality class and under what conditions they coincide.

Presentation models. We would like to give a syntactic generalisation of presentation models to complement enriched Lawvere theories. However, the appropriate generalisation of equational logic has not yet been developed [Plo06]. It is possible that Staton’s notion of parameterised theories [Sta13b] would serve this purpose.

Predicate models. Our notion of predicate models is general enough to include Benton et al.’s relational models. It is important to recast their models in terms of predicate models and compare them with the conservative restriction construction.

Our free lifting construction is a form of inductive lifting. It would be interesting to compare it to other lifting constructions, most notably Katsumata’s categorical- $\top\top$ -lifting [Kat05, Kat13] and Larrecq et al.’s use of scoping [GLLN05, GLLN02].

Intermediate language. Despite having the domain-theoretic machinery in place (see Chapter 4), we did not incorporate recursion into our source language in order to

²Marcelo P. Fiore, private communication, 2013.

focus on the shortest account towards effect-dependent optimisation. However, not accounting for recursion detracts from our applicability. Thus, an immediate next step is to give semantics to a MAIL variant with recursion [KP12] using our recursion models. To achieve establish the connection with the conservative restriction model, we use a continuous variant of our free lifting construction.

Another limitation of our intermediate language is caused by restricting arities to be of ground type. Generalising arities to non-ground types may involve recursive domain equations, as in the hypothesised treatment of higher-order store.

Optimisations. The logic we used for our optimisations is a simple equational logic — we have only considered equations between terms. It seems straightforward to devise a richer effect-dependent counterpart to Plotkin and Pretnars logic [PP08, Pre09]. Another direction is an effect-dependent account of refinement types, starting with Denney’s thesis [Den99].

We did not fully treat the local algebraic optimisations, those optimisations that originate from particular equations in the Lawvere theory. A full treatment would present a systematic translation from equations in the Lawvere theory into equations between program phrases, as Plotkin and Pretnar’s logic [PP08].

We did not supply any of the 36 proofs justifying the equivalent conditions for the global algebraic optimisations in Figure 11.2. We conjecture that there is a more basic description of these global optimisations using quantification over terms in the presentation \mathcal{L}_ε . This description may lead to a meta-theory of these optimisations. By exposing this syntax, we could generate both pristine and utilitarian forms mechanically from the more basic description. Their equivalence to the algebraic characterisation (and to each other using structural rules only) could then prove wholesale 27 of the proof obligations. The remaining 9 obligations are the equivalence of the monadic condition and either of the other characterisations. This basic description of the global optimisations would also allow us to uniformly define the notion of *operation-wise validity*, rather than give an ad-hoc definition for each of the optimisations.

Combining effects. We concentrated on the more common sum and tensor combinators. Hyland and Power’s distributive combination of theories [HP06] (used for combining ordinary and probabilistic computation) should be investigated too.

The combination of write-only state and non-determinism should be re-examined

in light of Example 13-2.

Use case We are interested in a tensor analogue of the sum conservativity theorem Theorem 13.7. However, in light of Example 13-2 where we combined write-only state and non-determinism, we cannot hope to a sufficient condition that encompasses all practical cases.

Our HASKELL model for validating the completeness of our decision procedures is limited by the HASKELL type system. We are therefore interested in a similar model in a dependently-typed programming language, such as Idris [Bra11], Agda [Nor07] or the Coq [Cdt12] proof assistance, which seem more suitable for this purpose.

We aspire to change the effect systems discussion. Rather than proceeding from case to case by analogy, we hope that the generality and applicability of our approach will provide a first step on the way to obtaining a scientifically-based engineering methodology to type-and-effect systems.

Appendix A

Conventions

*Here's a one of a kind
Convention of the mind
—Red Hot Chili Peppers*



Sections indicated with a “beware cats” sign assume deeper, but standard, knowledge of category theory. The assumed knowledge includes: (co)limits, adjunctions, (co)continuity, and monads, as covered by Mac Lane [ML98], and symmetric monoidal closed categories, and enrichment, as covered by Kelly [Kel82a].



The “cat-free” parts assume only basic categorical notions, previously used in the semantics and functional programming communities. We assume familiarity with functors, monads, natural transformations, and monad morphisms. These parts may be read sequentially, skipping over “beware cats” sections. While the statements are formulated in these more accessible terms, their proofs may rely on categorically involved accounts. The scope of these two modifiers extends to the end of the literary unit they are introduced in, such as Chapter, Section, or Proof.

We revisit definitions, examples, and theorems during the development. **Shading** indicates the difference from the original unit. An asterisk, e.g., Definition 3.2*, indicates the revisited unit is recast in terms of generic effects. An ω , e.g., Definition 3.1 ω , indicates the revisited unit is a domain-theoretic specialisation. A (revisited) mark, e.g., Definition 2.10 (revisited), indicates the revisited unit is a set-theoretic reformulation of a categorically involved unit. A plus, e.g. Definition 3.2⁺, indicates the revisited unit is an algebraic reformulation in terms of theories and/or presentations. These modifiers combine, e.g., Definition 3.2* (revisited)⁺, meaning an algebraic reformulation of the set-theoretic instance of the generic effects alternative to Definition 3.2. Electronic versions of this document use hyper-references to ease cross-references.

Bibliography

Look at me; I'm in tune

References around my room

—Blondie

- [AHS90] Jiří Adámek, Horst Herrlich, and George E. Strecker, *Abstract and concrete categories: the joy of cats*, Pure and applied mathematics, Wiley, 1990.
- [AJ94] Samson Abramsky and Achim Jung, *Domain theory*, Handbook of logic in computer science (vol. 3) (Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, eds.), Oxford University Press, Oxford, UK, 1994, pp. 1–168.
- [ALSU06] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, techniques, and tools (2nd edition)*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
- [AMBL12] Jiří Adámek, Stefan Milius, Nathan Bowler, and Paul B. Levy, *Coproducts of monads on set*, Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on, 2012, pp. 45–54.
- [AR94] Jiří Adámek and Jiří Rosický, *Locally presentable and accessible categories*, London Mathematical Society Lecture Note Series, no. 189, Cambridge University Press, 1994.
- [Atk09a] Robert Atkey, *Algebras for parameterised monads*, Algebra and Coalgebra in Computer Science (Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, eds.), Lecture Notes in Computer Science, vol. 5728, Springer Berlin Heidelberg, 2009, pp. 3–17.

- [Atk09b] ———, *Parameterised notions of computation*, *Journal of Functional Programming* **19** (2009), no. 3-4, 335–376.
- [BB07] Nick Benton and Peter Buchlovsky, *Semantics of an effect analysis for exceptions*, *Proceedings of the 2007 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation (New York, NY, USA), TLDI '07, ACM, 2007*, pp. 15–26.
- [Ben96] Nick Benton, *On the relationship between formal semantics and static analysis*, *ACM Computing Surveys* **28** (1996), no. 2, 321–323.
- [BK99] Nick Benton and Andrew Kennedy, *Monads, effects and transformations*, vol. 26, 1999, HOOTS '99, *Higher Order Operational Techniques in Semantics*, pp. 3 – 20.
- [BKBH07] Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann, *Relational semantics for effect-based program transformations with dynamic allocation*, *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (New York, NY, USA), PPDP '07, ACM, 2007*, pp. 87–96.
- [BKBH09] ———, *Relational semantics for effect-based program transformations: Higher-order store*, *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (New York, NY, USA), PPDP '09, ACM, 2009*, pp. 301–312.
- [BKHB06] Nick Benton, Andrew Kennedy, Martin Hofmann, and Lennart Beringer, *Reading, writing and relations*, *Programming Languages and Systems (Naoki Kobayashi, ed.), Lecture Notes in Computer Science, vol. 4279, Springer Berlin Heidelberg, 2006*, pp. 114–130.
- [BKR98] Nick Benton, Andrew Kennedy, and George Russell, *Compiling Standard ML to Java Bytecodes*, *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '98, ACM, 1998*, pp. 129–140.
- [BMW12] Clemens Berger, Paul-André Melliès, and Mark Weber, *Monads with arities and their associated theories*, *Journal of Pure and Applied Algebra*

- 216** (2012), no. 89, 2029 – 2048, Special Issue devoted to the International Conference in Category Theory ‘CT2010’.
- [Bou77] Aldridge K. Bousfield, *Constructions of factorization systems in categories*, Journal of Pure and Applied Algebra **9** (1977), no. 2, 207–220.
- [BP13] Andrej. Bauer and Matija. Pretnar, *An effect system for algebraic effects and handlers*, ArXiv e-prints (2013).
- [Bra11] Edwin C. Brady, *Idris — systems programming meets full dependent types*, Proceedings of the 5th ACM workshop on Programming languages meets program verification (New York, NY, USA), PLPV ’11, ACM, 2011, pp. 43–54.
- [Bri12] Philip Bridge, *Essentially algebraic theories and localizations in toposes and abelian categories*, Ph.D. thesis, The University of Manchester, 2012.
- [BS81] Stanley Burris and Hanamantagouda P. Sankappanavar, *A course in universal algebra*, Graduate Texts in Mathematics, no. 78, Springer-Verlag, 1981.
- [BS11] John Baez and Michael Stay, *Physics, topology, logic and computation: A rosetta stone*, New Structures for Physics (Bob Coecke, ed.), Lecture Notes in Physics, vol. 813, Springer Berlin / Heidelberg, 2011, pp. 95–172.
- [BT01] Lars Birkedal and Mads Tofte, *A constraint-based region inference algorithm*, Theoretical Computer Science **258** (2001), no. 1, 299–392.
- [BW95] Michael Barr and Charles Wells, *Category theory for computing science (2nd ed.)*, Prentice Hall international series in computer science, Prentice Hall, 1995.
- [Cdt12] The Coq development team, *The Coq proof assistant reference manual*, 2012, Version 8.4.
- [Den99] Ewen Denney, *A theory of program refinement*, Ph.D. thesis, University of Edinburgh, 1999.

- [dMS95] André Luís de Medeiros Santos, *Compilation by transformation in non-strict functional languages*, Ph.D. thesis, University of Glasgow, 1995.
- [EH62] Beno Eckmann and Peter J. Hilton, *Group-like structures in general categories I: multiplications and comultiplications*, *Mathematische Annalen* **145** (1962), no. 3, 227–255 (English).
- [FF86] Matthias Felleisen and Daniel P. Friedman, *Control operators, the SECD-machine, and the λ -calculus*, Proceedings of the IFIP TC 2/WG2.2 Working Conference on Formal Description of Programming Concepts Part III (Ebberup, Denmark), Indiana University, Computer Science Department, August 1986, pp. 193–219.
- [Fil07] Andrzej Filinski, *On the relations between monadic semantics*, *Theoretical Computer Science* **375** (2007), no. 1-3, 41 – 75, Festschrift for John C. Reynolds’s 70th birthday.
- [Füh00] Carsten Führmann, *The structure of call-by-value*, Ph.D. thesis, School of Informatics, University of Edinburgh, 2000.
- [Füh02] ———, *Varieties of effects*, *Foundations of Software Science and Computation Structures* (Mogens Nielsen and Uffe Engberg, eds.), *Lecture Notes in Computer Science*, vol. 2303, Springer Berlin Heidelberg, 2002, pp. 144–159 (English).
- [GLLN02] Jean Goubault-Larrecq, Slawomir Lasota, and David Nowak, *Logical relations for monadic types*, *Computer Science Logic* (Julian Bradfield, ed.), *Lecture Notes in Computer Science*, vol. 2471, Springer Berlin Heidelberg, 2002, pp. 553–568 (English).
- [GLLN05] ———, *Logical Relations for Monadic Types*, eprint arXiv:cs/0511006 (2005).
- [Gur91] Douglas J. Gurr, *Semantic frameworks for complexity*, Ph.D. thesis, University of Edinburgh, 1991.
- [HHPJW07] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler, *A history of Haskell: being lazy with class*, Proceedings of the third ACM SIGPLAN conference on History of programming languages (New York, NY, USA), HOPL III, ACM, 2007, pp. 12–1–12–55.

- [HLPP07] Martin Hyland, Paul B. Levy, Gordon D. Plotkin, and John Power, *Combining algebraic effects with continuations*, Theoretical Computer Science **375** (2007), no. 13, 20 – 40, Festschrift for John C. Reynolds’s 70th birthday.
- [HMN05] Fritz Henglein, Henning Makholm, and Henning Niss, *Effect types and region-based memory management*, Advanced Topics in Types and Programming Languages (Benjamin C. Pierce, ed.), MIT Press, 2005.
- [HP06] Martin Hyland and John Power, *Discrete Lawvere theories and computational effects*, Theoretical Computer Science **366** (2006), no. 12, 144 – 162, Algebra and Coalgebra in Computer Science.
- [HP07] ———, *The category theoretic understanding of universal algebra: Lawvere theories and monads*, Electronic Notes in Theoretical Computer Science **172** (2007), 437–458.
- [HPP06] Martin Hyland, Gordon D. Plotkin, and John Power, *Combining effects: Sum and tensor*, Theoretical Computer Science **357** (2006), no. 1-3, 70–99.
- [Jac94] Bart Jacobs, *Semantics of weakening and contraction*, Annals of Pure and Applied Logic **69** (1994), no. 1, 73 – 106.
- [Jac01] ———, *Categorical logic and type theory*, Studies in Logic and the Foundations of Mathematics, Vol 141, Elsevier Science Limited, 2001.
- [Jon95] Mark P. Jones, *Functional programming with overloading and higher-order polymorphism*, Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text (London, UK, UK), Springer-Verlag, 1995, pp. 97–136.
- [Kat05] Shin-ya Katsumata, *A semantic formulation of $\top\top$ -lifting and logical predicates for computational metalanguage*, Computer Science Logic (Luke Ong, ed.), Lecture Notes in Computer Science, vol. 3634, Springer Berlin Heidelberg, 2005, pp. 87–102.
- [Kat13] ———, *Relating computational effects by $\top\top$ -lifting*, Information and Computation **222** (2013), 228 – 246, 38th International Colloquium on Automata, Languages and Programming (ICALP 2011).

- [Kei10] Klaus Keimel, *Bicontinuous domains and some old problems in domain theory*, *Electronic Notes in Theoretical Computer Science* **257** (2010), 35–54.
- [Kel80] Gregory M. Kelly, *A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on*, *Bulletin of the Australian Mathematical Society* **22** (1980), 1–83.
- [Kel82a] ———, *Basic concepts of enriched category theory*, *Theory and Applications of Categories*, 1982, Reprinted in 2005.
- [Kel82b] ———, *Structures defined by finite limits in the enriched context, I*, *Cahiers de Topologie et Géométrie Différentielle Catégoriques* **23** (1982), no. 1, 3–42 (English).
- [Kie98] Richard B. Kieburtz, *Taming effects with monadic typing*, *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '98*, ACM, 1998, pp. 51–62.
- [KLO13] Ohad Kammar, Sam Lindley, and Nicolas Oury, *Handlers in action*, *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '13*, ACM, 2013, pp. 145–158.
- [Knu84] Donald E. Knuth, *Literate programming*, *The Computer Journal* **27** (1984), no. 2, 97–111.
- [Koc71] Anders Kock, *Bilinearity and cartesian closed monads*, *Mathematica Scandinavica* **29** (1971), 161–174.
- [Koc72] ———, *Strong functors and monoidal monads*, *Archiv der Mathematik* **23** (1972), 113–120.
- [KP12] Ohad Kammar and Gordon D. Plotkin, *Algebraic foundations for effect-dependent optimisations*, *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (New York, NY, USA), POPL '12*, ACM, 2012, pp. 349–360.
- [KW93] David J. King and Philip Wadler, *Combining monads*, *Functional Programming, Glasgow 1992* (John Launchbury and Patrick Sansom, eds.),

- Workshops in Computing, Springer London, 1993, pp. 134–143 (English).
- [Law63] Bill Lawvere, *Functorial semantics of algebraic theories*, Proceedings of the National Academy of Sciences of the United States of America **50** (1963), no. 1, 869–872.
- [Leh80] Daniel J. Lehmann, *On the algebra of order*, Journal of Computer and System Sciences **21** (1980), no. 1, 1–23.
- [Lev01] Paul B. Levy, *Call-by-push-value*, Ph.D. thesis, Queen Mary, University of London, 2001.
- [Lev02] ———, *Possible world semantics for general storage in call-by-value*, Computer Science Logic (Julian Bradfield, ed.), Lecture Notes in Computer Science, vol. 2471, Springer Berlin Heidelberg, 2002, pp. 232–246 (English).
- [Lev04] ———, *Call-by-push-value: A functional/imperative synthesis*, Semantics Structures in Computation, vol. 2, Springer, 2004.
- [LG88] John M. Lucassen and David K. Gifford, *Polymorphic effect systems*, Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (New York, NY, USA), POPL '88, ACM, 1988, pp. 47–57.
- [LHJ95] Sheng Liang, Paul Hudak, and Mark Jones, *Monad transformers and modular interpreters*, Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), POPL '95, ACM, 1995, pp. 333–343.
- [Lin69] Fred E. J. Linton, *Relative functorial semantics: Adjointness results*, Category Theory, Homology Theory and Their Applications III, Lecture Notes in Mathematics, vol. 99, Springer Berlin Heidelberg, 1969, pp. 384–418.
- [LP82] Daniel Lehmann and Ana Pasztor, *Epis need not be dense*, Theoretical Computer Science **17** (1982), no. 2, 151 – 161.

- [LP09] Stephen Lack and John Power, *Gabriel-Ulmer duality and Lawvere theories enriched over a general base*, *Journal of Functional Programming* **19** (2009), 265–286.
- [Mel10] Paul-André Melliès, *Segal condition meets computational effects*, *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science (Washington, DC, USA), LICS '10, IEEE Computer Society, 2010*, pp. 150–159.
- [Mes80] José Meseguer, *Varieties of chain-complete algebras*, *Journal of Pure and Applied Algebra* **19** (1980), no. 0, 347 – 383.
- [Mes81] ———, *Completions, factorizations and colimits for ω -posets*, *Mathematical Logic in Computer Science, Salgotarjan, 1978, Colloquia Mathematica Societatis Janos Bolyai, vol. 26, North Holland, 1981*, pp. 509–545.
- [Mit90] John C. Mitchell, *Handbook of theoretical computer science (vol. b)*, MIT Press, Cambridge, MA, USA, 1990, pp. 365–458.
- [ML98] Saunders Mac Lane, *Categories for the working mathematician (graduate texts in mathematics)*, 2nd ed., Springer, September 1998.
- [Mog89] Eugenio Moggi, *Computational lambda-calculus and monads*, *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (Piscataway, NJ, USA), IEEE Press, 1989*, pp. 14–23.
- [Mog90] ———, *An abstract view of programming languages*, Tech. Report ECS-LFCS-90-113, Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh, Edinburgh, Scotland, 1990.
- [Mog91] ———, *Notions of computation and monads*, *Information and Computation* **93** (1991), no. 1, 55–92.
- [Nor07] Ulf Norell, *Towards a practical programming language based on dependent type theory*, Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, September 2007.

- [Plo06] Gordon D. Plotkin, *Some varieties of equational logic*, Algebra, Meaning, and Computation (Kokichi Futatsugi, Jean-Pierre Jouannaud, and Jos Meseguer, eds.), Lecture Notes in Computer Science, vol. 4060, Springer Berlin Heidelberg, 2006, pp. 150–156.
- [Pow00] John Power, *Enriched Lawvere theories*, Theory and Applications of Categories **6** (2000), 83–93.
- [PP01] Gordon D. Plotkin and John Power, *Semantics for algebraic operations*, Electr. Notes Theor. Comput. Sci. **45** (2001), 332 – 345.
- [PP02] ———, *Notions of computation determine monads*, Foundations of Software Science and Computation Structures (Mogens Nielsen and Uffe Engberg, eds.), Lecture Notes in Computer Science, vol. 2303, Springer Berlin Heidelberg, 2002, pp. 342–356 (English).
- [PP03] ———, *Algebraic operations and generic effects*, Applied Categorical Structures **11** (2003), 2003.
- [PP08] Gordon D. Plotkin and Matija Pretnar, *A logic for algebraic effects*, Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science (Washington, DC, USA), LICS '08, IEEE Computer Society, 2008, pp. 118–129.
- [PP09a] ———, *Handlers of algebraic effects*, Proceedings of the 18th European Symposium on Programming Languages and Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009 (Berlin, Heidelberg), ESOP '09, Springer-Verlag, 2009, pp. 80–94.
- [PP09b] ———, *Handlers of algebraic effects*, Proceedings of the 18th European Symposium on Programming Languages and Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, ESOP '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 80–94.
- [Pre09] Matija Pretnar, *The logic and handling of algebraic effects*, Ph.D. thesis, University of Edinburgh, 2009.

- [PS82] Gordon D. Plotkin and Michael B. Smyth, *The category-theoretic solution of recursive domain equations*, SIAM J. Comput. **11** (1982), no. 4, 761–783.
- [Rey74] John C. Reynolds, *On the relation between direct and continuation semantics*, Proceedings of the 2Nd Colloquium on Automata, Languages and Programming (London, UK), Springer-Verlag, 1974, pp. 141–156.
- [Sco72] Dana S. Scott, *Continuous lattices*, Toposes, algebraic geometry and logic (1971 Dalhousie University Conference) (F. W. Lawvere, ed.), Lecture Notes in Mathematics, vol. 74, Springer-Verlag, New York, 1972, pp. 97–136.
- [Sel09] P. Selinger, *A survey of graphical languages for monoidal categories*, ArXiv e-prints (2009).
- [SGLH11] Nikhil Swamy, Nataliya Guts, Daan Leijen, and Michael Hicks, *Lightweight monadic programming in ml*, Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '11, ACM, 2011, pp. 15–27.
- [SO11] Tom Schrijvers and Bruno C.d.S. Oliveira, *Monads, zippers and views: Virtualizing the monad stack*, Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '11, ACM, 2011, pp. 32–44.
- [Sta09] Sam Staton, *Two cotensors in one: Presentations of algebraic theories for local state and fresh names*, Electronic Notes in Theoretical Computer Science **249** (2009), 471–490.
- [Sta13a] ———, *An algebraic presentation of predicate logic*, Foundations of Software Science and Computation Structures (Frank Pfenning, ed.), Lecture Notes in Computer Science, vol. 7794, Springer Berlin Heidelberg, 2013, pp. 401–417.
- [Sta13b] ———, *Instances of computational effects: An algebraic perspective*, Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium on, June 2013, pp. 519–519.

- [Tar38] Alfred Tarski, *Ein beitrag zur axiomatik der abelschen gruppen*, *Fundamenta Mathematicae* **30** (1938), no. 1, 253–256 (German).
- [TB98] Mads Tofte and Lars Birkedal, *A region inference algorithm*, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **20** (1998), no. 4, 724–767.
- [TB11] Jacob Thamsborg and Lars Birkedal, *A kripke logical relation for effect-based program transformations*, *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '11*, ACM, 2011, pp. 445–456.
- [TCGS91] Val Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov, *Inheritance as implicit coercion*, *Information and Computation* **93** (1991).
- [Tol98] Andrew P. Tolmach, *Optimizing ML using a hierarchy of monadic types*, *Types in Compilation*, 1998, pp. 97–115.
- [Wad90] Philip Wadler, *Comprehending monads*, *Proceedings of the 1990 ACM conference on LISP and functional programming (New York, NY, USA), LFP '90*, ACM, 1990, pp. 61–78.
- [Wad92] ———, *The essence of functional programming*, *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), POPL '92*, ACM, 1992, pp. 1–14.
- [Wad98] ———, *The marriage of effects and monads*, *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '98*, ACM, 1998, pp. 63–74.
- [WB89] Philip Wadler and Stephen Blott, *How to make ad-hoc polymorphism less ad hoc*, *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), POPL '89*, ACM, 1989, pp. 60–76.
- [WT03] Philip Wadler and Peter Thiemann, *The marriage of effects and monads*, *ACM Transactions on Computational Logic* **4** (2003), no. 1, 1–32.