Categorical models of second-order abstract syntax

Dmitrij Szamozvancev
Downing College

May 2025

This thesis is submitted for the degree of Doctor of Philosophy

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or, is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the Degree Committee of the Faculty of Computer Science and Technology.

Published work The content of Chapter 12 (*Computer formalisation*) is largely derived from the following paper presented at *Principles of Programming Languages 2022* and published in the corresponding proceedings:

Marcelo Fiore and Dmitrij Szamozvancev (2022)
Formal Metatheory of Second-Order Abstract Syntax
In: *Proceedings of the ACM on Programming Languages* 6 (POPL), 53:1–53:29.
DOI: 10.1145/3498715

Abstract

Mathematics and computer science increasingly rely on proof assistants to verify reasoning and ensure program correctness. Yet a persistent obstacle in the formalisation of programming languages and calculi is the treatment of variables and the associated operations of α-renaming and capture-avoiding substitution. Despite many proposed approaches, none match the flexibility and clarity of informal reasoning on paper. As a result, formalising languages in proof assistants often demands navigating a cumbersome layer of syntactic metatheory before any real benefits of mechanisation can be realised.

In parallel, mathematical frameworks offer powerful, reusable tools for working with syntax – such as type-preserving simultaneous substitution and compositional semantics via initial algebras. However, their practical impact on formal verification has been limited, due to their categorical sophistication and the challenges of encoding them in dependently-typed settings.

This thesis bridges this gap by formally relating the method of *intrinsically typed encodings* to the *presheaf approach* for languages with binding. We introduce the *familial model* of second-order abstract syntax as a categorical foundation for intrinsic typing, and show its equivalence to the presheaf model both syntactically and semantically. This places many ad hoc practices on solid mathematical footing, opening up principled paths for abstraction and extension. Along the way, we develop general tools for weakened monoidal structures and functorial models of syntax, framed by adjoint modalities – laying the groundwork for scalable and mechanised reasoning about syntax.

Acknowledgements

This work has been a long time in the making, and I am deeply grateful to my supervisor, Marcelo Fiore, for his patient, dedicated guidance, his generosity with ideas and research direction, and his unwavering support over many years. Marcelo has been with me from my very first Discrete Maths lecture, through illuminating seminars on denotational semantics and category theory, to long and enriching meetings that sparked years of joyful exploration. I have learned immensely from his teaching and writing; his emphasis on rigour and abstraction has profoundly shaped how I think and approach problems within and beyond mathematics.

This thesis would simply not have been possible without Nathanael Arkor, my other mentor in category theory. Nathanael was my sounding board, personal encyclopaedia, and safe space for asking all manner of questions – no matter how basic. Always generous with his time, he never dismissed or discouraged, and his depth of knowledge was both inspiring and invaluable. I must also mention the indispensable tool he created, `q.uiver.app`, without which this thesis would have taken even longer and contained far less category theory.

My PhD journey was bookended by research with Neel Krishnaswami and Jeremy Yallop, whose mentorship helped contextualise my work and gave me essential skills in type theory and programming languages. Neel introduced me to Agda and the joys and challenges of language formalisation, and my early struggles with substitution during my Master's were a key motivator for this thesis. Jeremy has been incredibly patient as I balanced writing up with the Modular Macros project. Our meetings, full of encouragement and insight – even when he was explaining macros for the mth time – made me feel I was doing truly open-ended research. His kindness and reassurance have meant a great deal, and I'm excited to see where our collaboration leads.

I also owe a great debt to Robert Harle for believing in me from the very beginning – fishing me from the 2014 Winter Pool, supporting me as Director of Studies, and eventually becoming someone I am honoured to call a colleague. He gave me countless opportunities to grow, and his generosity has more than made up for the times he teased me about being a theoretician in front of our students.

Downing College and the Department of Computer Science have been my academic homes for the past 11 years. I'm immensely thankful to the many friends, students, lecturers, Fellows, and staff who made this time so rewarding. Special thanks to Guy Williams for supporting my path to a Fellowship, and to Lise Gough, Joy Rook, Marketa Green, Karen Basser, Helen Neal, Dinah Pounds, and all the administrative

and teaching staff who helped me take part in supervisions, open days, and admissions. I felt truly supported throughout, and am grateful to all who made studying and working here such a privilege.

Many people have shaped the warm and welcoming Programming, Logic and Semantics Group. I'm especially thankful to Michael Gale, Philip Saville, Hugo Paquet, and Ohad Kammar for their academic guidance; to the pre-2020 PhD cohort – Ian Orton, Dylan McDermott, Matthew Daggitt, Sam Ainsworth, Sam Steenkamp, Vikraman Choudhury, Adam Ó Conghaile, Andrej Ivašković, Chelsea Edmonds, and Derek Sorensen – for many lively lunches and seminars; and to the current FS corridor/Victoria Road collective – Jake Bennett-Woolf, Michael Lee, Theo Wang, Yulong Huang, and Alistair O'Brien – for keeping me grounded and social (and still inviting me to lunch every week). I've also enjoyed working with many talented Master's students, especially Gregor Feierabend.

I have made lifelong friends in Downing and CUGCR, and I'm so impressed to see all the cool things they're up to in The Real World – Will, Ross, Greg C, Kirstie, Robin, Alfie, Gloria, Greg T, Nat, Katie, Dan, Stephen, Tom, Callum, Ella, Prannoy, Emma, Poppy, Alex F, Alex VL, and many others made time in university unforgettably fun and fulfilling. Robert has been an inspiration and very important friend since first grade, and I owe him nothing less than igniting my spark of curiosity for knowledge and learning, and guiding me in the first steps of my academic journey. I also want to thank all the students I taught the beauty of computer science – meeting and learning from so many bright, friendly, and deeply passionate people is truly one of the great joys and privileges of Cambridge teaching.

My parents always told me to keep learning while I can, and although 23 years in formal education may not be what they had in mind, I cannot be grateful enough for their support and love. I wouldn't be where I am without their encouragement and trust to reach further and try things, and they instilled in me values that I will carry around forever. I am also lucky to have the greatest and most inspiring big sister and brother in law, and the most wonderful niece and nephew – I will definitely explain monads to them one day.

Last but most definitely not least, my wife has been the most loving, caring, supportive, and patient partner I could have hoped for. She champions everything I do and helped me be excited for who I am and who I can be. I cannot wait to see what life brings, especially once it finally gallops past this slow, moving, 300-page milestone. Thank you for everything.

Table of contents

List of Definitions

CHAPTER 1

Introduction

> *A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail.*
>
> — Vladimir Voevodsky (2014)

The increasing complexity of modern mathematical and theoretical work has exposed the limitations of informal reasoning. Influential results have occasionally been undermined by subtle but critical errors; technical arguments often span many pages, pushing the bounds of human comprehension; and detailed proofs frequently go unverified by the broader community. In many cases, confidence is placed in the reputation of the author rather than the rigour of the argument itself. Identifying a mistake may cast doubt over an entire body of work, leaving it unclear whether the fault lies in the proof, the theorem, or the counterexample. As a discipline that progresses by building on the results of the past, mathematics must grapple with the uncomfortable reality that even widely accepted results can rest on fragile foundations.

Computer verification offers a promising solution. Increasingly, mathematics embraces proof assistants to formalise both historical work and new research (Paulson, 2023; Tao, 2025). With their reliance on small, trusted cores, flexible syntax, and well-understood logical foundations, proof assistants have opened new avenues for exploration and collaboration. The benefits are clear: formally verified proofs guarantee correctness; light automation handles "tedious but straightforward" details that would otherwise fill pages with mechanical symbol-pushing; and the open-source nature of formalised mathematics promotes collaboration, compartmentalisation, and peer review.

That said, there are good reasons a mathematician might hesitate to use a proof assistant at the start of a project: the steep learning curve, limited expressivity compared to informal reasoning, and the strict requirement for rigour introduce a kind of viscosity that familiar pen-and-paper methods do not. Nevertheless, proponents argue that it is precisely this stubborn demand for precision that gives proof assistants their greatest strength: no longer can we gloss over trivial-looking steps, forget obscure assumptions, or leave key arguments to the imagination of the "interested reader."

Research in theoretical computer science poses a slightly different challenge, both for humans and computers. Due to the inductive nature of most aspects of programming languages

– grammar of expressions, typing relation, evaluation, etc. – recursive definitions and inductive proofs are the primary way of interfacing with the metatheory of a syntax. Induction over the structure of a grammar or a typing derivation is notoriously tedious, often involving dozens of similar but subtly different cases, complex induction hypotheses, finicky constraints, and delicate bookkeeping. Using a proof assistant to streamline such work quickly reveals an additional difficulty: syntax involving variable-binding operators is far more intricate than pen-and-paper proofs tend to acknowledge. Subtle issues easily glossed over by human intuition become major obstacles during formalisation.

The primary culprits are notions of scope, variables, binding, and capture-avoiding substitution, all silently interacting with the following fundamental principle:

Variable names do not matter, as long as binding relationship match.

Humans naturally apply this principle and consciously avoid situations where variable names introduce confusion. Good practice dictates stating and following naming conventions, being explicit about changes of variables, and avoiding the reuse or overloading of names where ambiguity might arise. Proof assistants, however, must deal with the fluidity of variable names, resulting in challenges such as:

- $f(x) = x^2$ and $f(y) = y^2$ define the same function, as does $g(z) = z^2$ but with a different name which may or may not matter depending on the context;
- $f(x, y) = x^2 + y$ and $f(y, x) = y^2 + x$ are the same, $f(x, x) = x^2 + x$ is not a valid definition, but it is a valid predicate over x;
- $f(x) = x^2 + a$ and $f(y) = y^2 + b$ may be different or the same depending on the context:
  - they can be made equal by replacing a and b with the same term t: e.g. $f(x) = x^2 + (3 + c)$ the same as $f(y) = y^2 + (3 + c)$ for any c;
  - except if t has occurrences of x or y: e.g. $f(x) = x^2 + (3 + y)$ differs from $f(y) = y^2 + (3 + y)$;
  - except if they're bound occurrences: e.g. $f(x) = x^2 + ((\lambda y.\ 3 + y)\ c)$ is the same function as $f(y) = y^2 + ((\lambda y.\ 3 + y)\ c)$.

A researcher in programming languages may initially turn to proof assistants to expedite the long and tedious inductive proofs of properties like progress and type preservation, only to find themselves trapped instead in the minutiae of syntax representation, structural lemmas, and the definition and well-typedness of substitution – technicalities that informal proofs often gloss over with a few words or a footnote. Worse, because syntactic metatheory is often tailored to each specific language, this painstaking infrastructure must be rebuilt for every new formalisation project, with little opportunity for reuse.

1.1 The challenge

A comprehensive study of a language often involves analysing its computational behaviour and semantic meaning, expecting a close alignment between the two. One describes the former using *operational semantics*, and the latter via *denotational semantics*; their correspondence is proved using soundness, adequacy, and abstraction results. *Operational metatheory* for a language could involve:

Reduction A reduction relation $s \rightsquigarrow t$ defining how expressions are computed.

Equivalence Two terms are contextually equivalent if they reduce to the same value in the same evaluation context.

Determinacy The reduction relation is deterministic: if $s \rightsquigarrow t_1$ and $s \rightsquigarrow t_2$, then $t_1 = t_2$.

Progress Every closed well-typed term is either a value, or takes a reduction step.

Preservation Reduction preserves the typing and variable context of a term.

Safety Well-typed programs don't get stuck.

Closure If $s \rightsquigarrow t$, then s reduces to a value iff t reduces to a value.

Normalisation Every well-typed closed term reduces to a value.

On the other hand, *denotational metatheory* addresses both the mathematical meaning of programs and its relationship to operational behaviour:

Semantics Every term $\Gamma \vdash t : \alpha$ determines a morphism $[\![t]\!] : [\![\Gamma]\!] \to [\![\alpha]\!]$.

Compositionality For all evaluation contexts $\mathcal{C}[-]$, $[\![s]\!] = [\![t]\!]$ iff $[\![\mathcal{C}[s]]\!] = [\![\mathcal{C}[t]]\!]$.

Soundness If $\Gamma \vdash s, t : \alpha$ and $s \rightsquigarrow t$ then $[\![s]\!] = [\![t]\!] : [\![\Gamma]\!] \to [\![\alpha]\!]$.

Adequacy If two terms are denotationally equivalent, they are contextually equivalent.

Full abstraction If two terms are contextually equivalent, they are denotationally equivalent.

These informal lists illustrate the kinds of deep and often challenging properties one might wish to prove for a given language. Since many of these results are established via structural induction on syntax, proof assistants would appear to be a natural fit: they can automate routine arguments and manage the complex web of assumptions, variables, and hypotheses involved in more intricate proofs.

In practice, however, the path to such results is frequently obstructed by a mountain of syntactic overhead. Before one can reach the *interesting* theorems, one must first tackle an array of *uninteresting* but necessary properties about the syntax and its associated operations. Some of these are familiar and straightforward, while others arise unexpectedly mid-proof, interrupting progress with yet another tedious induction. Worse still, these properties often have little to do with the semantics or computational content of the language under study – they stem instead from the choice of syntactic representation and its inherent complexity: variable binding, scoping, and other structural constraints.

Regardless of the formalisation strategy adopted, syntactic concerns are all but guaranteed to surface at inopportune moments. Unlike informal proofs – where handwaving and intuition can often fill in the gaps – a proof assistant demands full rigour. The resulting *metasyntactic metatheory* involves the following kinds of foundational properties:

α-renaming Changing the name of binding and bound occurrences in parallel.

Substitution $[s/x]t$ replaces every free occurrence of x in t, avoiding variable capture.

Well-scoping If $\Gamma \vdash t : \alpha$, then $fv(t) \subseteq \mathrm{dom}(\Gamma)$.

Weakening If $\Gamma \vdash t : \beta$, then $\Gamma, x : \alpha \vdash t : \beta$.

Exchange If $\Gamma, x : \alpha, y : \beta \vdash t : \gamma$, then $\Gamma, y : \beta, x : \alpha \vdash t : \gamma$.

Substitution typing If $\Gamma, x : \alpha \vdash t : \beta$ and $\Gamma \vdash s : \alpha$, then $\Gamma \vdash [s/x]t : \beta$.

Substitution invariance If $x \notin fv(t)$, then $[s/x]t = t$.

Syntactic substitution lemma For all terms r, t, s, $[r/y]([s/x]t) = [[r/y]s/x]([r/y]t)$.

Semantic substitution lemma For all terms t and s, $[\![[s/x]t]\!] = [\![t]\!] \circ \langle \mathrm{id}, [\![s]\!] \rangle$.

These properties can be surprisingly intricate to prove in a formal setting – often more so than the high-level results they support. Naïvely following the structure of informal proofs may fail due to quirks of the syntax representation, or be finicky without the crutch of intuition and syntactic conventions. A successful formalisation typically requires abstracting and generalising beyond the original statement, introducing auxiliary lemmas, or rethinking the encoding of syntax entirely. In many cases, significant effort is required merely to reproduce what once seemed a triviality on paper.

1.2 Our solution

A number of techniques and frameworks exist to formalise languages with variable binding in proof assistants. One of the most critical design decisions concerns the representation of variables. Below, we outline some approaches, leaving a more detailed survey to Section 2.3.

Textual variables Variables are represented as strings (or, more generally, atoms with decidable equality), mirroring conventional notation. This offers human-readability, but does not provide any static enforcement of binding structure. Capture-avoiding substitution requires explicit string comparison and either limits substitution to closed terms or necessitates a separate mechanism for generating fresh variable names.

de Bruijn indices Variables are encoded as natural numbers indicating their binding depth. This nameless representation avoids the need for α-conversion or quotienting, but terms become less readable, and metatheoretic manipulations of indices can be subtle and error-prone.

Intrinsic syntax Terms are defined in such a way that only well-scoped and well-typed expressions can be represented. This integrates typing directly into the term structure, unifying term definitions with well-formedness properties. For instance, defining substitution simultaneously proves that substitution preserves typing. While this approach offers strong invariants, it also requires a nontrivial foundation of general-purpose syntactic operations and lemmas that must be derived from first principles.

Among these options, we adopt an intrinsic, nameless representation of syntax, owing to the strong static guarantees it provides in dependently-typed settings. This choice ensures that syntactic ill-formedness is excluded by construction, making subsequent semantic development more robust. However, we also recognise the significant up-front effort that intrinsic representations demand: much of the groundwork must be formalised before any high-level results can be established. Accordingly, our research aims to place this groundwork on a solid theoretical foundation, clarifying the structure of syntax, its mathematical underpinnings, and its role in categorical semantics.

Our starting point is the presheaf model of abstract syntax, developed by Fiore, Plotkin, and Turi 1999. In this framework, terms with variable binding are elements of presheaves – functors from a category of contexts and renamings to the category of sets. A term in a presheaf indexed by a context Γ is, by construction, well-scoped with respect to Γ. By building additional structure on the presheaf category, one can axiomatise substitution, variables, and binding structure in categorical terms.

While the presheaf model provides a conceptually clean and general account of syntax, it is not directly amenable to formalisation in dependently-typed proof assistants, particularly due to its reliance on quotient constructions and other mathematical tools. To address this, we develop a new, formalisable model that retains the key strengths of the presheaf approach while being suitable for mechanised formalisation.

Our approach is based on *indexed families of sets*, without assuming renaming as a primitive. By examining the presheaf model from the ground up, we isolate the minimal use of renaming and reconstruct the theory around a simpler foundation. The resulting model is both expressive and amenable to formalisation in Agda, enabling the automatic derivation of rich syntactic metatheory for a wide class of binding-aware languages. Concretely, we make the following contributions:

- A new categorical model of abstract syntax based on *indexed families of sets*, rather than presheaves over cocartesian categories. In this model, renaming is treated as additional, rather than intrinsic, structure. This results in a simplified formulation of substitution and binding, while retaining compatibility with existing theory.
  - We present a systematic account of categorical structures associated with syntax, including skew-monoidal closed categories, actions, modules, strength, and linearity, clarifying the relationships between multiple definitions in the literature.
  - We introduce the notion of *synthetic monoidal categories*, applicable in contexts where skew-monoidal structure is lost when objects are equipped with additional object-level structure (e.g. algebraic or semantic).
  - To bridge the gap between indexed families and presheaves, we prove general results for lifting categorical structures and universal constructions along forgetful functors.
- A formalisation framework in Agda built on the familial model. This framework provides a near end-to-end solution for defining syntactic structures and deriving their metatheory from a high-level specification. Key features include:
  - A generic metatheoretic development over arbitrary signatures, using initial algebra semantics rather than recursion on a fixed syntax.
  - An intrinsically-typed, structurally recursive syntax implementation, overcoming common issues of positivity and termination that affect other generic syntactic frameworks.

Our decision to pursue an abstract, category-theoretic foundation is motivated by a desire to uncover the deeper mathematical structure of syntax and to guide the design of formalisation tools in a principled way. In doing so, we aim not only to support the formalisation of simply-typed languages, but to lay the groundwork for modular, adaptable techniques applicable to

polymorphic, linear, or dependently-typed languages. A more ad hoc, problem-specific development would have likely obscured these general patterns and limited the reusability of results. By contrast, our approach aspires to connect the rich body of categorical semantics with practical mechanisation strategies, and to offer a framework extensible to future developments in the semantics of programming languages.

The formalisation framework is furthermore equipped with a lightweight code generation pipeline, enabling users to transition from a concise and human-readable syntax specification to a complete Agda formalisation via a single command. Users provide a high-level textual description of a typed language with variable binding and an optional equational theory. For instance, the following specification defines the simply-typed lambda calculus (STLC) extended with natural numbers (type N) and a function type constructor ($\to$):

```
term
  app  :       α → β   α  → β
  lam  :             α.β  → α → β
  ze   :                  → N
  su   :                N → N
  nrec : N   α   (α,N).α  → α

theory
  (λβ)          b : α.β  a : α ⊳       app(lam(x. b[x]), a) = b[a]
  (λη)             f : α → β ⊳          lam(x. app(f, x)) = f
  (zβ)       z : α  s : (α,N).α ⊳ nrec(ze,    z, rm. s[r,m]) = z
  (sβ) n : N z : α  s : (α,N).α ⊳ nrec(su(n), z, rm. s[r,m]) =
                                  s[nrec (n, z, rm. s[r,m]), n]
```

Our system generates Agda code for:

- a grammar of types and an intrinsically-typed data type of terms;
- operations of weakening and substitution together with their correctness properties;
- a record that, when instantiated with a mathematical model, induces a semantic interpretation of the syntax in the model that preserves substitution;
- a term constructor for parametrised metavariables and their associated operation of capture-permitting metasubstitution; and
- an equational/rewriting theory that can be instantiated with the axioms of the syntax to obtain a library for second-order equational/rewriting reasoning. For example, the equational proof that 1 + 2 = 3 looks as follows:

```
1+2 : ■ ▷ ∅ ⊢ plus $ su ze $ su (su ze) ≈ su (su (su ze))
1+2 = begin
  plus $ su ze $ su (su ze)          ≈⟨ thm λβ² with⟪ nrec x₁ x₀ (su x₀) ◃ su ze ◃ su (su ze) ⟫ ⟩
  nrec (su ze) (su (su ze)) (su x₀)  ≈⟨ ax sβ with⟪ ze ◃ su (su ze) ◃ su x₀ ⟫ ⟩
  su (nrec ze (su (su ze)) (su x₀))  ≈⟨ cong⟦ ax zeβ with⟪ su (su ze) ◃ su x₀ ⟫ ⟧inside su ○ᵃ ⟩
  su (su (su ze))                    ■
```

where plus is the primitive recursive encoding of addition, $\lambda\beta^2$ is a derived equation for two-argument β-reduction, and thm, ax and cong are helper operations for applying axioms and proved theorems inside subexpressions to construct explicit equational proofs.

1.3 Outline

Following this introductory chapter and a discussion of theoretical background and related work in Chapter 2, the thesis is organised into four main parts.

Part I introduces the mathematical tools used in the main development.

- Chapter 3 establishes the equivalence between distributive laws and liftings, a recurring tool used to extend functors to categories with added structure. In particular, Section 3.3 proves one of the central theorems of the thesis: that the family of syntactic terms generated by a signature gives rise canonically to a presheaf of terms, i.e. the renaming structure is determined by recursion on syntax.
- Chapter 4 develops the theory of biclosed modular categories over monoidal categories, including strong functors and the clone (double-dualisation) monad construction. This structure underpins the metasubstitution operation, which is shown to be a strong monad morphism between the term monad and the clone monad, with the relevant axioms derived from this general theory.

Part II explores the substitution structure of indexed families in a setting where monoidal coherence laws are relaxed, leading to skew-monoidal structure.

- Chapter 5 develops a systematic theory of skew-monoidal closed structures in categories and modular categories, including the formulation of monoid and module objects, and properties of parametrised morphisms.
- Chapter 6 considers categories that embed into skew-monoidal categories while possessing internal analogues of the unit and tensor. These *synthetic monoidal categories* support a theory of synthetic monoidal and module functors, as well as monoid and module objects, and are essential to formalising constructions over families that do not carry over directly from presheaves, such as pointed strength.
- Chapter 7 revisits the notion of skew-monoidal *warpings*, which allow the generation of new monoidal structures from existing ones. *Skew-closed* warpings are introduced, along with sufficient conditions for lifting categorical structure along warpings. The key result is an equivalence between algebraic monoids and lifted structures, which ultimately yields an equivalence between syntactic models in the presheaf and familial settings.

Part III introduces the *familial model*, relates it to the presheaf model, and demonstrates its use in establishing initiality and freeness theorems for second-order abstract syntax.

- Chapter 8 surveys the theory of presheaves and related constructions such as co/ends, Kan extensions, Day convolution, nerves, and realisations.
- Chapter 9 synthesises substitution structure for presheaves and families in three ways: via universal properties, from first principles, and through an adjoint warping. This last formulation yields the canonical skew substitution tensor on families, providing a formal bridge to the presheaf model.

- Chapter 10 presents the familial model in full detail, emphasising differences from the presheaf approach. Notably, it employs Day convolution for variable binding and metasubstitution, module structure for renaming, skew-monoidal structure for substitution, and synthetic monoidal structure to capture strength and algebraic behaviour.
- Chapter 11 applies this theory to reproduce and generalise the initiality and freeness results of the presheaf model in the familial setting. The new proofs are more efficient, leveraging internal homs to better accommodate the skew structure. The chapter also derives a second-order metasubstitution theory and sound equational reasoning.

Part IV explores the practical advantages of the familial model in formalisation, highlighting its computational tractability in contrast with the more abstract presheaf approach.

- Chapter 12 describes the Agda formalisation framework developed from the familial model. Families are naturally represented as functions into Set, offering a simpler and more usable foundation than fully functorial encodings. With modest adaptations, the framework faithfully implements the core model and syntactic metatheory without quotient types.
- Chapter 13 presents example applications of the framework, including generic syntactic constructions and concrete encodings of various syntaxes and equational systems.

Finally, Chapter 14 summarises the main contributions and outlines future directions for extending this work in both theory and practice.

Practicalities

Most of the relevant categorical notation is introduced as part of the text. Equational and diagrammatic proofs are annotated with references implying an equality step or the commutativity of a certain cell. We collect the main conventions for diagram labelling below – they are intended to be more indicative than fully rigorous, and in some cases the same label is used for a class of laws, with identification of the exact property left to the reader.

- Equalities appear as double lines without arrowheads, and no label.
- Isomorphisms appear without arrows at either end, and often without a label when the transformation is clear. The direction is also omitted if it is inferable or unimportant.
- A cell is not labelled if the edges are identical composites, or it commutes by functoriality.
- A cell is labelled $o^{\triangleq}$ if it commutes by definition of the operator o.
- A cell is labelled $\boxed{\varphi}$ if it commutes by naturality of φ. If a natural transformation is multi-ary, $\boxed{\varphi}_k$ indicates naturality in the k^{th} component.
- A natural transformation φ preserving an operation o is generally notated as $\varphi\lfloor o \rceil$.
- An F-co/algebra homomorphism axiom for f is labelled $f\lfloor \vec{F} \rceil$.
- Monad laws (unit or associativity) for T will simply be labelled by the monad name itself, and laws of a monad algebra $a\colon TA \to A$ by $\vec{T}$.
- Adjunction zig-zag identities will be denoted $\dashv$.

When working with several adjunctions, we will avoid using ε and η (and labelled/renamed variants) for the co/units in favour of explicitly referencing the natural isomorphism of hom-sets $\tau\colon \mathcal{D}(FA, B) \cong \mathcal{C}(A, GB)$ associated with the particular adjunction. We will write $\tau\colon F \dashv G\colon \mathcal{C} \to \mathcal{D}$ for such an adjunction. Then the unit $\tau(\mathrm{id}_{FA})\colon A \to GFA$ and counit $\overline{\tau}(\mathrm{id}_{GB})\colon FGB \to B$ will be denoted as τ_A and $\overline{\tau}_B$, respectively; this will not lead to ambiguity as we will not be referring to specific components of τ as a natural isomorphism.

CHAPTER 2

Background

We begin by motivating and contextualising the technical developments of this thesis through a detailed exposition of the two main strands of research it aims to unify: intrinsically-typed language formalisation and the presheaf model of abstract syntax. Section 2.1 introduces the intrinsic encoding of syntax, highlighting both its intuitive appeal and the often counterintuitive, piecemeal nature of developing its metatheory, which is prone to ad hoc constructions and subtle pitfalls. In addressing these challenges, we identify opportunities for abstraction that recent work has touched upon but not yet grounded in a coherent theoretical framework. Section 2.2 then develops the classical (cartesian) presheaf model of Fiore et al. (1999), extending it to encompass second-order features. In particular, we provide a more comprehensive treatment of second-order features that of Fiore (2008), offering a principled account of metasubstitution and meta-interpretation. Finally, Section 2.3 presents an in-depth survey of existing literature on the formal foundations of syntax and its mechanised implementation.

2.1 Intrinsic syntax

Intrinsic syntax, also called *type- and scope-safe encoding*, embraces the "illegal phrases are unrepresentable" principle in unifying the term grammar with the typing judgment. Every encoded term is well-typed and well-scoped by construction: types and contexts are assigned "at construction", and respected by all operations, such as substitution and reduction, with correctness statically enforced by the type checker. This stands in contrast with the usual extrinsic style, where a raw term grammar is defined independently of the typing judgment, and operations are defined independently of their type-preservation proofs. Since most metatheoretic properties – progress, safety, normalisation, etc. – require a well-typedness assumption, intrinsic syntax avoids redundancy and administrative overhead inherent in duplicating the term grammar and type system.

2.1.1 Advantages

Intrinsic encodings operate over sets of the form $\{t \mid \Gamma \vdash t : \alpha\}$, containing only well-typed, well-scoped terms. Defining and manipulating such families requires dependently-typed inductive families (Dybjer, 1994), which proof assistants like Agda and Rocq support natively. For example, an intrinsic encoding of the simply typed lambda calculus (STLC) in Agda defines four components: sorts, contexts, variables, and terms:

```
data S : Set where
  B    : S
  _→_  : S → S → S

data Ctx : Set where
  ∅    : Ctx
  _·_  : S → Ctx → Ctx
```

```
data V : S → Ctx → Set where
  new :          V α (α · Γ)
  old  : V β Γ → V β (α · Γ)

data Λ : S → Ctx → Set where
  var  : V α Γ → Λ α Γ
  app  : Λ (α → β) Γ → Λ α Γ → Λ β Γ
  lam  : Λ β (α · Γ) → Λ (α → β) Γ
```

Contexts are lists of sorts, and variables are type- and scope-safe de Bruijn indices. Each term constructor enforces its typing rule statically: variables select a context entry, applications combine terms of matching function and argument types, and λ-abstractions extend the context and bind a new variable. As examples, consider the SKI combinator terms:

```
I : Λ (α → α) Γ
I = lam (var new)
```

```
K : Λ (α → β → α) Γ
K = lam (lam (var (old new)))
```

```
S : Λ ((α → β → γ) → ((α → β) → α → γ)) Γ
S = lam (lam (lam (app (app (var (old (old new))) (var new))
                       (app (var (old new)) (var new))))))
```

Although verbose due to explicit de Bruijn manipulations, these terms encode type- and scope-safe syntax trees: any change to the expressions will result in a type error. Agda can even synthesize parts of these definitions automatically, owing to the rigid type and context dependencies enforced by the constructors. Such static guarantees would not be enforced if one used untyped terms, numeric de Bruijn indices, or textual variables. Operations like weakening, substitution, and reduction are naturally typed to preserve well-typedness:

```
wkn  : Λ α Γ → Λ α (β · Γ)
sub  : Λ α Γ → Λ β (α · Γ) → Λ β Γ
_⇝_ : Λ α Γ → Λ α Γ → Set
```

For instance, weakening explicitly shifts variables in the syntax tree, with the type system ensuring correctness. Substitution encapsulates the single-variable substitution lemma; reduction expresses type-preserving evaluation steps. In each case, type preservation is not merely a theorem but is embedded directly into the recursive definition of the operations.

2.1.2 Challenges

Intrinsic typing has some very compelling properties, but fully committing to it also means accepting its idiosyncrasies. Namely, it forces us to prove more things upfront than other approaches, and will forbid us from continuing with the development until Agda's type checker is happy – prototyping, experimentation and testing are all blocked by the "boring bits". In this section we give a flavour the familiar frustrations of defining the innocuous top-variable substitution operation $\mathsf{sub} : \Lambda\ \alpha\ \Gamma \to \Lambda\ \beta\ (\alpha \cdot \Gamma) \to \Lambda\ \beta\ \Gamma$ that forms the basis of the operational semantics of the STLC. In an intrinsically-typed setting, the definition should parallel the usual proof of the well-typedness of substitution, which rarely causes difficulties on paper: it proceeds by induction on t, pushing the term s under constructors until a free occurrence of the last variable in the context is reached. Agda requires the utmost rigour however, so deriving single-variable substitution will take us through some meandering workarounds.

First attempt: top substitution We define $\mathsf{sub}\ s\ t$ by induction on $t : \Lambda\ \beta\ (\alpha \cdot \Gamma)$. The key cases are variables and abstractions, where contexts change nontrivially. In the variable case, if $t = \mathsf{var}\ x$, pattern-matching on x distinguishes two subcases:

- If $x = \mathsf{new} : \mathsf{V}\ \alpha\ (\alpha \cdot \Gamma)$, corresponding to the top variable, we return s.
- If $x = \mathsf{old}\ (y : \mathsf{V}\ \beta\ \Gamma)$, a lower variable, we leave it unchanged as $\mathsf{var}\ y : \Lambda\ \beta\ \Gamma$.

In application nodes, substitution recurses structurally:

$$
\begin{aligned}
&\mathsf{sub} : \Lambda\ \alpha\ \Gamma \to \Lambda\ \beta\ (\alpha \cdot \Gamma) \to \Lambda\ \beta\ \Gamma \\
&\mathsf{sub}\ s\ (\mathsf{var}\ \mathsf{new}) \quad\ = s \\
&\mathsf{sub}\ s\ (\mathsf{var}\ (\mathsf{old}\ y)) = \mathsf{var}\ y \\
&\mathsf{sub}\ s\ (\mathsf{app}\ f\ a) \quad\ = \mathsf{app}\ (\mathsf{sub}\ s\ f)\ (\mathsf{sub}\ s\ f)
\end{aligned}
$$

The abstraction case, however, reveals a difficulty. If $t = \mathsf{lam}\ b : \Lambda\ (\beta \to \gamma)\ (\alpha \cdot \Gamma)$, the body b has type $\Lambda\ \gamma\ (\beta \cdot \alpha \cdot \Gamma)$. Informally, the substitution $[s/x](\lambda y : \beta.\ b)$ recurses into the body of the binding, avoiding variable shadowing and capture by renaming the bound variable y if needed. This is not immediately possible in Agda, since freshness of the new variable in b is explicitly represented by context extension, and its type β "covers" the free variable α that we are substituting for. The recursive call to sub is therefore not general enough, since it only substitutes for the last variable in the context.

$$\mathsf{sub}\ s\ (\mathsf{lam}\ b) = \mathsf{lam}\ \big(\mathsf{sub}\ (\square : \Lambda\ \alpha\ (\beta \cdot \Gamma))\ (\square : \Lambda\ \gamma\ (\alpha \cdot \beta \cdot \Gamma)) : \Lambda\ \gamma\ (\beta \cdot \Gamma)\big)$$

In a pen-and-paper proof of the syntactic substitution lemma, we would use the structural rules of exchange to swap α and β in the context of b, and weakening to weaken $s : \Lambda\ \alpha\ \Gamma$ to $\Lambda\ \alpha\ (\beta \cdot \Gamma)$ before applying the induction hypothesis – again, very standard properties that are rarely dwelt upon. Sadly, to define $\mathsf{wkn} : \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ (\tau \cdot \Gamma)$ leads to the same problem as before: the recursive call is impossible since we would need to add the τ variable under the newly bound one, weakening $b : \Lambda\ \beta\ (\alpha \cdot \Gamma)$ to $\Lambda\ \beta\ (\alpha \cdot \tau \cdot \Gamma)$ without disturbing α.

$$\mathsf{wkn}\ (\mathsf{lam}\ (b : \Lambda\ \beta\ (\alpha \cdot \Gamma))) = \mathsf{lam}\ \big(\mathsf{wkn}\ (\square : \Lambda\ \beta\ (\alpha \cdot \Gamma)) : \Lambda\ \beta\ (\alpha \cdot \tau \cdot \Gamma)\big)$$

Perhaps all will be solved if we define $\mathsf{exch} : \Lambda\ \alpha\ (\beta \cdot \gamma \cdot \Gamma) \to \Lambda\ \alpha\ (\gamma \cdot \beta \cdot \Gamma)$? No luck – once again, the recursive call only accounts for the top of the context which gets extended in binding terms. We evidently need to strengthen our induction hypothesis.

Second attempt: middle substitution An easy way to generalise the substitution function is to allow substituting for an arbitrary variable, surrounded by any two contexts:

$$\mathsf{sub} : (\Gamma : \mathsf{Ctx}) \to \Lambda\ \alpha\ (\Gamma + \Delta) \to \Lambda\ \beta\ (\Gamma + (\alpha \cdot \Delta)) \to \Lambda\ \beta\ (\Gamma + \Delta)$$

This is strong enough to make the recursive call in the lam, instantiating Γ with $(\beta \cdot \Gamma)$. However, lam (sub s b) doesn't typecheck yet since s has to be weakened from $\Lambda\ \alpha\ (\Gamma + \Delta)$ to $\Lambda\ \alpha\ (\beta \cdot (\Gamma + \Delta))$. We can do this using a generalised middle-weakening map

$$\mathsf{wkn} : \Lambda\ \alpha\ (\Gamma + \Delta) \to \Lambda\ \alpha\ (\Gamma + (\tau \cdot \Delta))$$

which is now definable, assuming we also establish weakening for variables:

```
wkn-var : (Γ : Ctx) → V α (Γ + Δ) → V α (Γ + (τ · Δ))
wkn-var ∅       x       = old x
wkn-var (α · Γ) new     = new
wkn-var (α · Γ) (old x) = old (wkn-var Γ x)

wkn : (Γ : Ctx) → Λ α (Γ + Δ) → Λ α (Γ + (τ · Δ))
wkn Γ (var x)      = var (wkn-var Γ x)
wkn Γ (app f a)    = app (wkn Γ f) (wkn Γ a)
wkn Γ (lam {γ} b) = lam (wkn (γ · Γ) b)
```

Though the abstraction case of sub is solved, we are now unable to complete the variable case:

```
sub : (Γ : Ctx) → Λ α (Γ + Δ) → Λ β (Γ + (α · Δ)) → Λ β (Γ + Δ)
sub ∅       s (var new)     = s
sub ∅       s (var (old x)) = var x
sub (α · Γ) s (var x)       = var new
sub (α · Γ) s (var (old x)) = var (old ( □ : V β (Γ + Δ)))
sub Γ       s (app f a)     = app (sub Γ s f) (sub Γ s a)
sub Γ       s (lam {γ} b)   = lam (sub (γ · Γ) (wkn ∅ s) b)
```

The hole expects a variable $\mathsf{V}\ \beta\ (\Gamma + \Delta)$, but x has type $\mathsf{V}\ \beta\ (\Gamma + (\alpha \cdot \Delta))$ and can't generally be "strengthened" to the required form. Despite the promising initial steps, no amount of pattern-matching will let us close the hole and complete the definition of sub.

Third attempt: simultaneous substitution We need to step back and consider the most general form of the substitution operation: simultaneous substitution. Instead of substituting a single term s of type α for a single free variable of type α in t, we replace every free variable in t with different terms of appropriate types, all of which must be in the same context. For example, applying the simultaneous substitution $\sigma = [x_1 \mapsto (y : \beta \vdash s_1 : \alpha_1), x_2 \mapsto (y : \beta \vdash s_2 : \alpha_2)]$ to the term $x_1 : \alpha_1, x_2 : \alpha_2 \vdash t : \tau$ results in the term $y : \beta \vdash [\sigma]t : \tau$. There are two ways to represent a simultaneous substitution rule σ in Agda: as an inductive data type containing a sequence of terms, or as a dependent function space that maps variables in one context to terms in another. For the purposes of formalising substitution laws the function space representation is more convenient.

```
Sub : Ctx → Ctx → Set
Sub Γ Δ = {τ : S} → V τ Γ → Λ τ Δ
```

The simultaneous substitution operation now has the type `sub : Sub Γ Δ → Λ α Γ → Λ α Δ`, easily specialised to single-variable substitution using `Sub (β · Δ) Δ` that maps the existing variables to themselves and the new variable to a term `Λ β Δ`. In the definition of `sub`, variables are looked up using the substitution rule without any case analysis, and application recurses into subterms. More work is needed for abstraction, since the body binds a new variable which must be left undisturbed by the substitution. We need to "lift" a substitution `Sub Γ Δ` over a binder, resulting in a map `Sub (τ · Γ) (τ · Δ)`; intuitively, it maps the new variable to itself, and old variables to the original terms, weakened by `τ`:

```
lift : Sub Γ Δ → Sub (τ · Γ) (τ · Δ)
lift σ new     = var new
lift σ (old x) = wkn (σ x)
```

The operation `wkn : Λ α Γ → Λ α (τ · Γ)` is derived from the centre-weakening of the previous attempt. With this, we can finally implement the remaining cases of substitution, and derive single-variable substitution by mapping the last variable of the context to `s`:

```
sub : Sub Γ Δ → Λ α Γ → Λ α Δ
sub σ (var x)   = σ x
sub σ (app f a) = app (sub σ f) (sub σ a)
sub σ (lam b)   = lam (sub (lift σ) b)

[_/] : Λ α Γ → Λ β (α · Γ) → Λ β Γ
[ s /] = sub λ{ new ↦ s ; old x ↦ var x }
```

This approach works well, and it scales to terms that bind any number of variables – just need to apply `lift` several times. Nevertheless, it is worth pondering if there are further opportunities for abstraction that may clean up the definitions, especially of `wkn-var` and `wkn`. Indeed there are: structural lemmas – weakening, exchange, etc. – can all be expressed as renaming the variables of a term according to some renaming rule between contexts. A renaming rule `Ren Γ Δ` is a mapping from variables in `Γ` to variables in `Δ`, and the renaming operation applies the rule `Ren Γ Δ` to a term `Λ α Γ` to obtain a term `Λ α Δ`.

```
Ren : Ctx → Ctx → Set
Ren Γ Δ = {τ : S} → V τ Γ → V τ Δ

ren : Ren Γ Δ → Λ α Γ → Λ α Δ
ren ρ (var x)   = var (ρ x)
ren ρ (app f a) = app (ren ρ f) (ren ρ a)
ren ρ (lam b)   = lam (ren (ext ρ) b)

ext : Ren Γ Δ → Ren (γ · Γ) (γ · Δ)
ext ρ new     = new
ext ρ (old x) = old (ρ x)
```

Here, `ext` – the analogue of `lift` for variables – lifts a renaming over a binder by mapping a newly bound variable to itself, and "weakening" the existing variables using `old`. The weakening map `wkn : Λ α Γ → Λ α (γ · Γ)` used in the definition of `lift` above is then defined as `ren old`.

While we are not making full use of renaming here, the abstraction is valuable as it lifts any context transformation to the level of terms. Instead of one structurally recursive definition for every structural property with inductive correctness laws, we have a single renaming operation, a few general laws involving renaming, and as many instances of the structural lemmas – weakening, exchange, contraction, permutation, etc. – as we want.

With much trial and error, we were able to define the substitution operation for the simply-typed lambda calculus. Examining the approach, we make two main observations: both renaming and substitution are required, but their definitions and the auxiliary functions (weakening, lifting) they depend on are quite similar. Perhaps this can be generalised even further?

Fourth attempt: syntactic traversal Since ren is needed for wkn, lift and subsequently sub, renaming cannot be derived from substitution. But McBride (2005) showed that there is a way to generalise both operations into a single *traversal* function that, with the appropriate instantiations, specialises to renaming and substitution. A traversal recurses into a term and replaces every free variable with "stuff", for a suitable notion of "stuff": if it is a variable, the traversal acts as a renaming, and if it is a term, we get substitution. A type-preserving mapping from variables to a type- and context-indexed family of sets generalises the Sub and Ren constructs defined above:

```
Map : (S → Ctx → Set) → Ctx → Ctx → Set
Map 𝒳 Γ Δ = {τ : S} → V τ Γ → 𝒳 τ Δ
```

A term traversal trav : Map 𝒳 Γ Δ → Λ α Γ → Λ α Δ can be defined generically if we assume some structure on the family 𝒳, axiomatised in a record that McBride calls a Kit. The record supports the translation of variables into stuff, translation of stuff into terms, and weakening of stuff. Every Kit instance supports a lifting operation on traversal rules.

```
record Kit (𝒳 : S → Ctx → Set) : Set where
  field var : V α Γ → 𝒳 α Γ
        trm : 𝒳 α Γ → Λ α Γ
        wkn : 𝒳 α Γ → 𝒳 α (τ · Γ)

lift : Map 𝒳 Γ Δ → Map 𝒳 (τ · Γ) (τ · Δ)
lift σ new     = var new
lift σ (old x) = wkn (σ x)
```

With an instance of Kit 𝒳, the traversal operation is defined as follows:

```
trav : Map 𝒳 Γ Δ → Λ α Γ → Λ α Δ
trav σ (var x)   = trm (σ x)
trav σ (app f a) = app (trav σ f) (trav σ a)
trav σ (lam b)   = lam (trav (lift σ) b)
```

Renaming and substitution can be extracted by defining Kit instances for variables and terms. Crucially, this is a two-step process: the renaming operation derived from Kit V is used to instantiate wkn in Kit Λ.

```
V-Kit : Kit V
V-Kit = { var = id  ; trm = var ; wkn = old }

ren : Map V Γ Δ → Λ α Γ → Λ α Δ
ren = V-Kit.trav

Λ-Kit : Kit Λ
Λ-Kit = { var = var ; trm = id  ; wkn = ren old }

sub : Map Λ Γ Δ → Λ α Γ → Λ α Δ
sub = Λ-Kit.trav
```

Finally, having waded through some messy definitions and ad hoc generalisations, we arrived at a satisfying conclusion: an abstract term traversal operation that encompasses both renaming and substitution, but maintains their conceptual separation through a dependency relationship. This invites yet another opportunity for abstraction.

Fifth attempt: semantic traversals McBride's syntactic traversal approach was extended by Allais et al. (2017) to arbitrary *semantic traversals*. The key insight is that the Kit.tm field can more generally return a family that supports the operations of the λ-calculus, i.e. a *model* of STLC. An Allais-style semantic kit is thus parametrised by two type- and context-indexed families, and requires conversion maps $\mathsf{V} \xrightarrow{\mathsf{var}} \mathcal{X} \xrightarrow{\mathsf{trm}} \mathcal{M}$, weakening on $\mathcal{X}$, and analogues of the application and λ-abstraction (presented in terms of extension) operations on $\mathcal{M}$.

```
record Traversal (𝒳 ℳ : S → Ctx → Set) : Set where
  field var : V α Γ → 𝒳 α Γ
        trm : 𝒳 α Γ → ℳ α Γ
        wkn : 𝒳 α Γ → 𝒳 α (τ · Γ)
        A   : ℳ (α → β) Γ → ℳ α Γ → ℳ β Γ
        L   : ℳ β (α · Γ)→ ℳ (α → β) Γ
```

The traversal function acts as a semantic interpretation of STLC terms in an arbitrary model, additionally incorporating a mapping into a new context.

```
trav : Map 𝒳 Γ Δ → Λ α Γ → ℳ α Δ
trav σ (var x)     = trm (σ x)
trav σ (app f a) = A (trav σ f) (trav σ a)
trav σ (lam b)     = L (trav (lift σ) b)
```

Renaming and substitution are recovered from instances Traversal V Λ and Traversal Λ Λ, but the construction is far more general: Allais et al. (2021) show how to use it for pretty-printing, normalisation-by-evaluation, CPS transformation, desugaring, and many other applications. Considering how systematic the structure of Traversal and trav is, a natural question arises: can this construction be *syntax-generic*, operating over an arbitrary second-order signature?

Sixth attempt: signature-generic traversals Generically encoding the constructors of a second-order syntax can be done in several ways. One, using algebras for signature endofunctors, is the main topic of this thesis. Another, a technique based on generic programming (Weirich and Casinghino, 2012), is to encode the second-order syntax as a universe of codes (Chapman et al., 2010), and generalise the Traversal record to arbitrary syntax encodings. Introduced by Allais et al. (2021), this is a powerful but quite technical solution, and forces the user into a particular term representation. While their approaches served as an important inspiration to our work, the Agda-heavy implementation obscures some fruitful paths of investigation that present themselves quite naturally when looking at the categorical model.

2.1.3 The need for theoretical foundations

The definitions of renaming, substitution, and other traversals are only half the story: we also rely on their correctness properties, beyond simple type preservation. These typically take the form of compatibility laws that express how traversals can be combined, commuted, or simplified. Examples are the substitution associativity law $[r/y]([s/x]t) = [[r/y]s/x]([r/y]t)$, the substitution identity law $[x/x]t = t$, or the semantic substitution lemma $[\![[s/x]t]\!] = [\![t]\!] \circ \langle \mathrm{id}, [\![s]\!] \rangle$. As with substitution itself, formalising these laws in a proof assistant is far more intricate than it first appears, and the challenges are rarely discussed in the literature. Nevertheless, they are unavoidable: such laws regularly surface in proofs of important theorems like soundness and normalisation.

Attempting to prove a special case of a law is typically hopeless: the induction hypotheses will not be strong enough to handle variable or binding cases. Consequently, the laws must be formulated for simultaneous substitution, which in turn depends on corresponding laws for lift, themselves requiring a sequence of auxiliary lemmas about renaming, and so on. Working with intrinsic typing demands considerable patience and determination: once we recognise a required property of substitution, we must be prepared for a cascade of technical lemmas that must first be established. Benton et al. (2012) list twelve such correctness properties – covering all the identity and composition laws for renaming and substitution, together with their associated lifting lemmas – that must be proved sequentially, each one building on previous results, in order to establish substitution associativity. There are four "unit" laws:

$$\begin{aligned}
\mathsf{ext}\ \mathsf{id} &= \mathsf{id} &&: \forall\ \alpha\ (\tau \cdot \Gamma) \to \forall\ \alpha\ (\tau \cdot \Gamma) && \text{(e-id)}\\
\mathsf{ren}\ \mathsf{id} &= \mathsf{id} &&: \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Gamma && \text{(r-id)}\\
\mathsf{lift}\ \mathsf{var} &= \mathsf{var} &&: \forall\ \alpha\ (\tau \cdot \Gamma) \to \Lambda\ \alpha\ (\tau \cdot \Gamma) && \text{(l-v)}\\
\mathsf{sub}\ \mathsf{var} &= \mathsf{id} &&: \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Gamma && \text{(s-v)}
\end{aligned}$$

and eight "fusion" laws, for all renaming rules $\rho : \mathsf{Ren}\ \Gamma\ \Delta$, $\varrho : \mathsf{Ren}\ \Delta\ \Theta$, and substitution rules $\sigma : \mathsf{Map}\ \Lambda\ \Gamma\ \Delta$, $\varsigma : \mathsf{Map}\ \Lambda\ \Delta\ \Theta$:

$$\begin{aligned}
\mathsf{ext}\ (\varrho \circ \rho) &= \mathsf{ext}\ \varrho \circ \mathsf{ext}\ \rho &&: \forall\ \alpha\ (\tau \cdot \Gamma) \to \forall\ \alpha\ (\tau \cdot \Theta) && \text{(e-e)}\\
\mathsf{ren}\ (\varrho \circ \rho) &= \mathsf{ren}\ \varrho \circ \mathsf{ren}\ \rho &&: \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Theta && \text{(r-r)}\\
\mathsf{lift}\ (\varsigma \circ \rho) &= \mathsf{lift}\ \varsigma \circ \mathsf{ext}\ \rho &&: \forall\ \alpha\ (\tau \cdot \Gamma) \to \Lambda\ \alpha\ (\tau \cdot \Theta) && \text{(l-e)}\\
\mathsf{sub}\ (\varsigma \circ \rho) &= \mathsf{sub}\ \varsigma \circ \mathsf{ren}\ \rho &&: \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Theta && \text{(s-r)}\\
\mathsf{lift}\ (\mathsf{ren}\ \varrho \circ \sigma) &= \mathsf{ren}\ (\mathsf{ext}\ \varrho) \circ \mathsf{lift}\ \sigma &&: \forall\ \alpha\ (\tau \cdot \Gamma) \to \Lambda\ \alpha\ (\tau \cdot \Theta) && \text{(re-l)}\\
\mathsf{sub}\ (\mathsf{ren}\ \varrho \circ \sigma) &= \mathsf{ren}\ \varrho \circ \mathsf{sub}\ \sigma &&: \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Theta && \text{(r-s)}\\
\mathsf{lift}\ (\mathsf{sub}\ \varsigma \circ \sigma) &= \mathsf{sub}\ (\mathsf{lift}\ \varsigma) \circ \mathsf{lift}\ \sigma &&: \forall\ \alpha\ (\tau \cdot \Gamma) \to \Lambda\ \alpha\ (\tau \cdot \Theta) && \text{(sl-l)}\\
\mathsf{sub}\ (\mathsf{sub}\ \varsigma \circ \sigma) &= \mathsf{sub}\ \varsigma \circ \mathsf{sub}\ \sigma &&: \Lambda\ \alpha\ \Gamma \to \Lambda\ \alpha\ \Theta && \text{(s-s)}
\end{aligned}$$

Allais et al. (2021) take a more general approach, considering a notion of simulation between traversals, and fusion of two traversals into one; while powerful, their method employs an advanced logical relations framework to prove results that, conceptually, should stem from straightforward (albeit tedious) structural inductions on the grammar.

The central aim of our work, then, is to place the tradition of intrinsic encodings on a formal foundation, where the abstractions outlined above can be properly captured. Our starting point is the *presheaf model* of abstract syntax developed by Fiore et al. (1999), which, despite its parallel evolution alongside much of programming language formalisation, has yet to be adopted directly as a basis for a metatheoretic verification framework. This is likely due both to the model's abstract mathematical nature – which does not lend itself easily to implementation – and a prevailing attitude of "I'm just doing Rocq/Agda; I don't need abstract category theory." Indeed, Allais et al. (2021, Section 10.3) express a somewhat dismissive sentiment about the presheaf model, arguing that by avoiding commitment to an external mathematical semantics, their framework remains more flexible and less encumbered by type-theoretic assumptions.

Our belief is that the presheaf model provides an efficient and robust encapsulation of intrinsic encoding, along with many of its generalisations, extensions, and pragmatic adaptations developed through years of trial and error. The adjustments required to fit the presheaf model into a practical, working implementation are all mathematically justified. Moreover, by committing to the full rigour of the theory, we are able to exploit its full categorical strength.

2.2 The presheaf model

We set the scene for the work by giving a high-level, but concrete summary of the presheaf model of second-order abstract syntax (Fiore et al., 1999; Fiore, 2002, 2008). To give context for the abstract constructions, we use the simply-typed λ-calculus as our running example, notationally mirroring the intrinsically-typed formalisation of the previous section.

Sorts and contexts The grammar of STLC sorts below generates the set S:

$$\alpha, \beta \in S \ ::= \ \mathsf{B} \mid \alpha \to \beta$$

The category of *contexts* is the free cocartesian category generated by S, $\mathbb{F}[S]$. Its objects are lists of sorts, morphisms are renaming rules $\Gamma \to \Delta$ mapping elements of Γ to elements of Δ, and the coproduct is context concatenation $\Gamma + \Delta$, with injections $\iota_2^{\Gamma,\Delta} : \Gamma \to \Gamma + \Delta$ and $\iota_1^{\Gamma,\Delta} : \Delta \to \Gamma + \Delta$, and copairing of renaming rules $\rho : \Gamma \to \Theta$ and $\varrho : \Delta \to \Theta$ to $[\rho, \varrho]_\Theta^{\Gamma,\Delta} : \Gamma + \Delta \to \Theta$.

Category of presheaves A *sorted presheaf* $\mathcal{P} = \{\mathcal{P}_\alpha \Gamma\}_{\alpha \in S, \Gamma \in \mathbb{F}[S]} \in (\mathbf{Set}^{\mathbb{F}[S]})^S$ is a sort- and context-indexed family of sets with an associated renaming structure that maps $\Gamma \to \Delta$ to a function $\mathcal{P}_\alpha \Gamma \to \mathcal{P}_\alpha \Delta$ for all α. Elements $t \in \mathcal{P}_\alpha \Gamma$ of a presheaf are intrinsically typed and scoped, representing a term t of type α in context Γ. Sorted presheaves and natural transformations between them form the category $\mathbf{PSh}_S \triangleq (\mathbf{Set}^{\mathbb{F}[S]})^S$.

Syntactic signature A *binding signature* lists the operators of the syntax, along with their type signatures which explicitly specify when an operator binds a variable in its argument.

$$\mathsf{app} : (\alpha \to \beta) \times \alpha \ \longrightarrow \ \beta \qquad \mathsf{lam} : (\alpha.\beta) \ \longrightarrow \ (\alpha \to \beta)$$

The signature of a second-order syntax gives rise to a *signature endofunctor* $\Sigma : \mathbf{PSh}_S \to \mathbf{PSh}_S$, mapping a presheaf to the coproduct of the operator parameters. The context extension oper-

ator $\delta_\tau(\mathcal{P})_\alpha\Gamma = \mathcal{P}_\alpha(\tau \cdot \Gamma)$ can be used to express binding terms, and equality constraints are used to specialise the output sort. For the STLC, this would be

$$\Sigma_\Lambda \mathcal{P}_\tau \triangleq \left[\sum_{\alpha,\beta\in S} \mathcal{P}_{\alpha\to\beta} \times \mathcal{P}_\beta \times (\tau = \beta)\right] + \left[\sum_{\alpha,\beta\in S} \delta_\alpha \mathcal{P}_\beta \times (\tau = (\alpha \to \beta))\right]$$

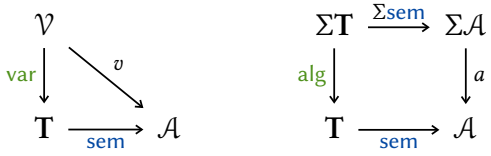
As the definition only involves products, coproducts, and the context extension endofunctor $\delta_\tau : \mathbf{PSh}_S \to \mathbf{PSh}_S$, the output is indeed a presheaf over $\mathbb{F}[S]$.

Variables The presheaf of *variables* $\mathcal{V} \in (\mathbf{Set}^{\mathbb{F}[S]})^S$ is defined as the Yoneda embedding $\mathbb{F}[S]^{\mathrm{op}} \to \mathbf{Set}^{\mathbb{F}[S]}$ precomposed with the singleton list embedding $[-] : S \to \mathbb{F}[S]$:

$$\mathcal{V}_\alpha\Gamma \triangleq \text{Y}[\alpha](\Gamma) \triangleq \mathbb{F}[S]([\alpha], \Gamma)$$

This corresponds to the Var type in Agda, as a morphism in $\mathbb{F}[S]([\alpha], \Gamma)$ selects an occurrence of α in Γ, whose index may be encoded as a well-typed de Bruijn index.

Terms The presheaf of terms is the initial $(\mathcal{V} + \Sigma)$-algebra, whose object part is the inductive definition of terms above. Concretely, it is a family $\mathbf{T}$ with natural transformations $\mathsf{var} : \mathcal{V} \to \mathbf{T}$ (the "$\mathcal{V}$-algebra" part) and $\mathsf{alg} : \Sigma\mathbf{T} \to \mathbf{T}$ (the Σ-algebra part). The first corresponds to the constructor for variables, while the Σ-algebra is a copairing of the term constructors. Initiality of the term presheaf gives rise to definitions by structural recursion: for any $(\mathcal{V} + \Sigma)$-algebra $(\mathcal{A},\ v : \mathcal{V} \to \mathcal{A},\ a : \Sigma\mathcal{A} \to \mathcal{A})$, there exists a unique morphism $\mathsf{sem} : \mathbf{T} \to \mathcal{A}$ satisfying the homomorphism conditions of $(\mathcal{V} + \Sigma)$-algebras:



For the signature of the STLC, we therefore get natural transformations in $\mathbf{PSh} = \mathbf{Set}^{\mathbb{F}[S]}$

$$\mathsf{var} : \mathcal{V}_\alpha \to \mathbf{T}_\alpha \qquad \mathsf{lam} : \delta_\alpha \mathbf{T}_\beta \to \mathbf{T}_{\alpha\to\beta} \qquad \mathsf{app} : \mathbf{T}_{\alpha\to\beta} \times \mathbf{T}_\alpha \to \mathbf{T}_\beta$$

for all $\alpha, \beta \in S$. The initial homomorphism conditions state that given any Σ_Λ-endofunctor $(\mathcal{A}, v, a, l)$, we have an interpretation map $\mathsf{sem} : \mathbf{T} \to \mathcal{A}$ that satisfies the following compositionality laws, which can equivalently be seen as the definition of sem by structural recursion on the syntax encoded in the initial algebra:

$$\begin{aligned}
\mathsf{sem}\,(\mathsf{var}\,x) &= v\,x \\
\mathsf{sem}\,(\mathsf{lam}\,b) &= l\,(\mathsf{sem}\,b) \\
\mathsf{sem}\,(\mathsf{app}\,(f,t)) &= a\,(\mathsf{sem}\,f, \mathsf{sem}\,t)
\end{aligned}$$

Renaming and substitution rules Context maps ${}^\Gamma\mathcal{P}_\Delta$ are products $\mathcal{P}^{\times\Gamma}(\Delta) \triangleq \prod_{\alpha\in\Gamma} \mathcal{P}_\alpha\Delta$, mapping a variable of type α in Γ to a $\mathcal{P}$-term of type α in Δ. They are called *renaming rules*

for $\mathcal{P} = \mathcal{V}$ and denoted simply as $\Gamma \to \Delta$ as they are isomorphic to morphisms in $\mathbb{F}[S]$. For $\mathcal{P} = \mathbf{T}$, these will be denoted $\Gamma \xrightarrow{\mathbf{T}} \Delta$ and called *substitution rules*.

Renaming under a binder Given a renaming rule $\Gamma \to \Delta$, the lifting of ρ is the action of the context extension endofunctor $\delta_\tau \rho \colon (\tau \cdot \Gamma) \to (\tau \cdot \Delta)$ defined as $\mathrm{id}_{[\tau]} + \rho \colon ([\tau] + \Gamma) \to ([\tau] + \Delta)$. In the presheaf model, the context extension endofunctor has an equivalent characterisation as exponentiation by the presheaf of variables: $\delta_\tau P \cong (\mathcal{V}_\tau \supset P)$, and for arbitrary contexts, $\delta_\Theta P \cong \yo\Theta \supset P$. This follows from the Yoneda lemma via the following calculation:

$$(\yo\Theta \supset P)(\Gamma) \cong \mathbf{PSh}\big(\yo\Gamma \times \yo\Theta, P\big) \cong \mathbf{PSh}\big(\yo(\Theta + \Gamma), P\big) \cong P(\Theta + \Gamma) \cong \delta_\Theta P(\Gamma)$$

Simultaneous renaming Since the initial $(\mathcal{V} + \Sigma)$-algebra is a presheaf, its functorial action maps $\Gamma \to \Delta$ to $\mathbf{T}_\alpha(\rho) \colon \mathbf{T}_\alpha\Gamma \to \mathbf{T}_\alpha\Delta$ and corresponds to renaming. The algebra structure map $(\mathcal{V} + \Sigma)\mathbf{T} = \mathcal{V} + \Sigma\mathbf{T} \to \mathbf{T}$ is a natural transformation between presheaves, and its naturality decomposes as the following compatibility conditions between functoriality and constructors:

$$\begin{aligned}
\mathbf{T}(\rho)\ (\mathsf{var}\ x) &= \mathsf{var}\ (\rho\ x) \\
\mathbf{T}(\rho)\ (\mathsf{lam}\ b) &= \mathsf{lam}\ (\mathbf{T}([\alpha] + \rho)\ b) \\
\mathbf{T}(\rho)\ (\mathsf{app}\ (f, t)) &= \mathsf{app}\ (\mathbf{T}(\rho)\ f, \mathbf{T}(\rho)\ t)
\end{aligned}$$

These, of course, correspond to the structural recursive definition one would use to equip the underlying family of the initial algebra with the functorial action.

Weakening The single-variable weakening of a term $\mathsf{wkn} \colon \mathbf{T}_\alpha\Delta \to \mathbf{T}_\alpha(\tau \cdot \Delta)$ is derived as renaming by the injection $\iota_1^{[\tau],\Delta} \colon \Delta \to [\tau] + \Delta$. Of course, this works for any presheaf P, and an arbitrary extending context Θ:

$$\mathsf{wkn} \triangleq P(\iota_1^{\Theta,\Gamma}) \colon P(\Gamma) \to P(\Theta + \Gamma)$$

Substitution under a binder Given a substitution rule $\sigma \colon \Gamma \xrightarrow{\mathbf{T}} \Delta$, we can extend both the domain and codomain of σ to $\mathsf{lift}\ \sigma \colon \Theta + \Gamma \xrightarrow{\mathbf{T}} \Theta + \Delta$ by the copairing

$$\mathbf{T}\big[\mathsf{var} \circ \iota_2^{\Theta,\Gamma},\ \mathsf{wkn} \circ \sigma\big]^{\Theta,\Gamma}_{\Theta+\Delta} \colon \Theta + \Gamma \xrightarrow{\mathbf{T}} \Theta + \Delta$$

The lifting extends into a more general operation constructed in terms of the *substitution tensor product* $\otimes$ of presheaves, defined through the *substitution action* $\oslash$ as

$$(P \oslash \mathcal{Q})(\Delta) \triangleq \int^{\Gamma \in \mathbb{F}[S]} P(\Gamma) \times {}^{\Gamma}\mathcal{Q}_\Delta \qquad (\mathcal{P} \otimes \mathcal{Q})_\alpha \triangleq \mathcal{P}_\alpha \oslash \mathcal{Q}$$

The substitution tensor product contains equivalence classes of tuples $(\Gamma,\ t \in \mathcal{P}_\alpha\Gamma,\ \sigma \colon {}^{\Gamma}\mathcal{Q}_\Delta) \in (\mathcal{P} \otimes \mathcal{Q})_\alpha\Delta$ of a context, a term and a substitution, quotiented by the equivalence relation generated from the following condition: for all $\rho \colon \Gamma \to \Delta$, $\sigma \in {}^{\Delta}\mathcal{Q}_\Theta$, and terms $t \in \mathcal{P}_\alpha\Gamma$, we identify the (componentwise unequal) tuples

$$(\Delta,\ \mathcal{P}(\rho)t,\ \sigma) \sim (\Gamma,\ t,\ \sigma \circ \rho)$$

Being a tensor product, we have natural isomorphisms $\lambda_{\mathcal{Q}} \colon \mathcal{V} \otimes \mathcal{Q} \cong \mathcal{Q}$, $\rho_{\mathcal{P}} \colon \mathcal{P} \otimes \mathcal{V} \cong \mathcal{P}$ and $\alpha_{\mathcal{P},\mathcal{Q},\mathcal{R}} \colon (\mathcal{P} \otimes \mathcal{Q}) \otimes \mathcal{R} \cong \mathcal{P} \otimes (\mathcal{Q} \otimes \mathcal{R})$, which make use of functoriality and the quotienting. The "lifting" is then expressed as a *pointed tensorial strength* transformation for the context extension endofunctor δ_Θ, operating on a presheaf $P \in \mathbf{PSh}$ and *pointed* presheaf $(\mathcal{Q} \in \mathbf{PSh}_S, \eta \colon \mathcal{V} \to \mathcal{Q})$:

$$(\delta_\Theta P) \oslash \mathcal{Q} \to \delta_\Theta (P \oslash \mathcal{Q})$$

This maps equivalence classes of tuples $(\Gamma,\ t \in P(\Theta + \Gamma),\ \sigma \colon {}^{\Gamma}\mathcal{Q}_{\Delta}) \in (\delta_\Theta P \oslash \mathcal{Q})\Delta$ to the equivalence class of

$$(\Theta + \Gamma,\ t,\ \mathsf{lift}\ \sigma) \in \delta_\Theta (P \oslash \mathcal{Q})\Delta$$

where $\mathsf{lift}\ \sigma \colon {}^{\Theta+\Gamma}\mathcal{Q}_{\Theta+\Delta}$ is the generalised lifting for the context map $\sigma \colon {}^{\Gamma}\mathcal{Q}_{\Delta}$

$$\mathcal{Q}\left[\eta \circ \iota_2^{\Theta,\Gamma},\ \mathsf{wkn} \circ \sigma\right]_{\Theta+\Delta}^{\Theta,\Gamma} \colon {}^{\Theta+\Gamma}\mathcal{Q}_{\Theta+\Delta}$$

The strength for δ_Θ forms part of the strength for the signature endofunctor $\Sigma \colon \mathbf{PSh}_S \to \mathbf{PSh}_S$

$$\mathsf{str} \colon \Sigma(\mathcal{P}) \otimes \mathcal{Q} \to \Sigma(\mathcal{P} \otimes \mathcal{Q})$$

which is responsible for "pushing" substitutions into constructor arguments and under binders. While the strength may not exist for arbitrary functors, it can always be derived when Σ is constructed from a second-order signature: since $\Sigma\mathcal{P}$ is a sum-of-products-of-deltas, and we have the strength $\delta_\Theta P \oslash (\mathcal{R}) \to \delta_\Theta (P \oslash \mathcal{R})$, and isomorphisms and transformations

$$(P + Q) \oslash \mathcal{R} \cong (P \oslash \mathcal{R}) + (Q \oslash \mathcal{R}) \qquad (P \times Q) \oslash \mathcal{R} \to (P \oslash \mathcal{R}) \times (Q \oslash \mathcal{R})$$

we use these repeatedly to push $(-) \oslash \mathcal{R}$ deeper into the signature. For the STLC,

$$\begin{aligned} \left(\delta_\alpha \mathcal{P}_\beta + (\mathcal{P}_{\alpha\to\beta} \times \mathcal{P}_\alpha)\right) \oslash \mathcal{R} \cong (\delta_\alpha \mathcal{P}_\beta \oslash \mathcal{R}) \ &+\ \left((\mathcal{P}_{\alpha\to\beta} \times \mathcal{P}_\alpha) \oslash \mathcal{R}\right) \\ \to \delta_\alpha (\mathcal{P}_\beta \oslash \mathcal{R}) \ &+\ \left((\mathcal{P}_{\alpha\to\beta} \oslash \mathcal{R}) \times (\mathcal{P}_\alpha \oslash \mathcal{R})\right) \end{aligned}$$

which, as explicit mappings on constructor components, amounts to

$$\begin{aligned} \left[(\Gamma,\ b \in \delta_\alpha \mathcal{P}_\beta \Gamma,\ \sigma \in {}^{\Gamma}\mathcal{Q}_{\Delta})\right] &\mapsto \left[(\alpha \cdot \Gamma,\ b \in \mathcal{P}_\beta(\alpha \cdot \Gamma),\ \mathsf{lift}\ \sigma \colon {}^{\alpha\cdot\Gamma}\mathcal{Q}_{\alpha\cdot\Delta})\right] \\ \left[(\Gamma,\ (f \in \mathcal{P}_{\alpha\to\beta}\Gamma,\ t \in \mathcal{P}_\alpha \Gamma),\ \sigma \in {}^{\Gamma}\mathcal{Q}_{\Delta})\right] &\mapsto \left([(\Gamma, f, \sigma)], [(\Gamma, t, \sigma)]\right) \end{aligned}$$

The pointed strength also allows us to generalise initiality of the presheaf of terms $\mathbf{T}$ to *parametrised initiality*, using the closed structure: for any pointed presheaf $(\mathcal{P}, v \colon \mathcal{V} \to \mathcal{P})$ – taking the role of the *parameter* – and $(\mathcal{P} + \Sigma)$-algebra $(\mathcal{A}, p \colon \mathcal{P} \to \mathcal{A}, a \colon \Sigma\mathcal{A} \to \mathcal{A})$, there exists a unique $(\Sigma + \mathcal{V})$-algebra homomorphism $\mathsf{trav} \colon \mathbf{T} \otimes \mathcal{P} \to \mathcal{A}$ satisfying

$$\begin{array}{ccc} \mathcal{V} \otimes \mathcal{P} & \xrightarrow{\mathcal{V}\otimes p} & \mathcal{V} \otimes \mathcal{A} \\ {\scriptstyle \mathsf{var}\otimes\mathcal{P}}\downarrow & & \downarrow{\scriptstyle \lambda_{\mathcal{A}}} \\ \mathbf{T} \otimes \mathcal{P} & \xrightarrow[\mathsf{trav}]{} & \mathcal{A} \end{array} \qquad \begin{array}{ccccc} \Sigma\mathbf{T} \otimes \mathcal{P} & \xrightarrow{\mathsf{str}} & \Sigma(\mathbf{T} \otimes \mathcal{P}) & \xrightarrow{\Sigma\mathsf{trav}} & \Sigma\mathcal{A} \\ {\scriptstyle \mathsf{alg}\otimes\mathcal{P}}\downarrow & & & & \downarrow{\scriptstyle a} \\ \mathbf{T} \otimes \mathcal{P} & & \xrightarrow[\mathsf{trav}]{} & & \mathcal{A} \end{array}$$

For the STLC, the axioms for this traversal operation become the recursive specifications

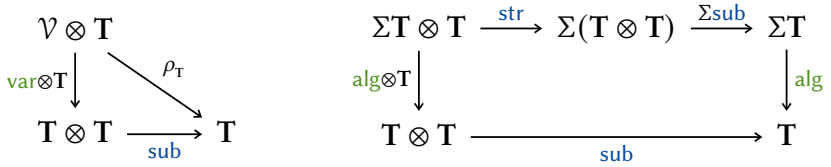
$$\begin{aligned}
\mathsf{trav}\,(\mathsf{var}\,x, \sigma) &= p\,(\sigma\,v) \\
\mathsf{trav}\,(\mathsf{lam}\,b, \sigma) &= l\,(\mathsf{trav}\,(b, \mathsf{lift}\,\sigma)) \\
\mathsf{trav}\,(\mathsf{app}\,(f, t), \sigma) &= a\,(\mathsf{trav}\,f, \sigma, \mathsf{trav}\,t, \sigma)
\end{aligned}$$

The naming is not coincidental – trav generalises the traversal maps of McBride (2005) (see Section 2.1.2) and behaves similarly to the semantics operation of Allais et al. (2021).

Simultaneous substitution The simultaneous substitution operation for presheaves can be expressed as a *monoid multiplication* for the substitution tensor product, turning $\mathbf{T}$ into a monoid object in $\mathbf{PSh}_S$:

$$\mathsf{var} : \mathcal{V} \to \mathbf{T} \qquad \mathsf{sub} : \mathbf{T} \otimes \mathbf{T} \to \mathbf{T}$$

It maps a term $t \in \mathbf{T}_\alpha\Gamma$ and a substitution rule $\sigma : \Gamma \xrightarrow{\mathbf{T}} \Delta$ to a term $\mathsf{sub}\,(\Gamma, t, \sigma) : \mathbf{T}_\alpha\Delta$. The substitution operation $\mathsf{sub} : \mathbf{T} \otimes \mathbf{T} \to \mathbf{T}$ is an instance of a traversal operation induced by parametrised initiality, with the pointed presheaf $\mathcal{P}$ instantiated with $(\mathbf{T}, \mathsf{var})$ and $(\mathcal{P} + \Sigma)$-algebra with $(\mathbf{T}, \mathrm{id} : \mathbf{T} \to \mathbf{T}, \mathsf{alg} : \Sigma\mathbf{T} \to \mathbf{T})$. The axioms of traversal maps then become



expanding for the STLC to the expected recursive specification of substitution:

$$\begin{aligned}
\mathsf{sub}\,(\mathsf{var}\,x, \sigma) &= \sigma\,v \\
\mathsf{sub}\,(\mathsf{lam}\,b, \sigma) &= \mathsf{lam}\,(\mathsf{sub}\,(b, \mathsf{lift}\,\sigma)) \\
\mathsf{sub}\,(\mathsf{app}\,(f, t), \sigma) &= \mathsf{app}\,(\mathsf{sub}\,(f, \sigma), \mathsf{sub}\,(t, \sigma))
\end{aligned}$$

Single-variable substitution The single-variable substitution operation is a natural transformation $[-/] : \mathbf{T}_\alpha \times \delta_\alpha\mathbf{T}_\beta \to \mathbf{T}_\beta$, that performs a substitution with a rule ${}^{\alpha\cdot\Gamma}\mathbf{T}_\Gamma$:

$$[\,s\,/]\,t \triangleq \mathsf{sub}\big(\alpha \cdot \Gamma,\ t,\ \mathrm{T}[t,\ \mathsf{var}]_\Gamma^{[\alpha],\Gamma}\big)$$

An algebraic description of this operation is the subject of Fiore et al. (1999, Section 3) and, more recently, Fiore and Ranchod (2024, Section 4).

Simultaneous substitution laws The categorical presheaf model makes laws an essential part of the structures, rather than an afterthought. We do the hard work upfront – in sufficient generality – from which valuable corollaries are easily derived, rather than having to work backwards to figure out the general properties needed to derive the specific laws one seeks. Of the 12 laws in the previous section, the ones concerning ext and lift follow from the strength laws for the endofunctor δ_Θ:

$$\mathsf{ext}\,\mathrm{id} = \mathrm{id} \qquad \mathsf{lift}\,(x \mapsto (\Gamma, \sigma\,x, \varsigma))\,y = (\Delta, \mathsf{lift}\,\sigma\,y, \mathsf{lift}\,\varsigma)$$

The ones concerning sub and ren are the unit, naturality and associativity laws of the monoid multiplication $\mathsf{sub} : \mathbf{T} \otimes \mathbf{T} \to \mathbf{T}$ – establishing them is part of showing that $\mathbf{T}$ is a monoid.

Any signature endofunctor has a strength transformation which satisfies the required axioms (which are straightforward for products and sums, and reduce to the two axioms above for δ_τ), from which the main theorem is proved: the initial $(\mathcal{V}+\Sigma)$-algebra is a monoid. The monoid multiplication – substitution – is extracted as a traversal as above, and its laws use the *uniqueness* of parametrised-initial interpretations. For example, the unit law $\mathsf{sub}\,(\mathsf{var} \circ \rho)\,t = \mathbf{T}\,\rho\,t$ expresses the equality of the two morphisms

$$\mathbf{T} \otimes \mathcal{V} \xrightarrow{\mathbf{T} \otimes \mathsf{var}} \mathbf{T} \otimes \mathbf{T} \xrightarrow{\mathsf{sub}} \mathbf{T} \qquad \mathbf{T} \otimes \mathcal{V} \xrightarrow{\lambda_{\mathbf{T}}} \mathbf{T}$$

which, if shown to be homomorphisms of $(\mathcal{V}+\Sigma)$-algebras, must be equal to the unique traversal $\mathsf{trav} : \mathbf{T} \otimes \mathcal{V} \to \mathbf{T}$ induced by parametrised initiality on the pointed presheaf $(\mathcal{V}, \mathrm{id})$ and $(\mathcal{V} + \Sigma)$-algebra $(\mathbf{T}, \mathsf{var}, \mathsf{alg})$. Similarly, the associativity law $\mathsf{sub}\,(\mathsf{sub}\,\varsigma \circ \sigma) = \mathsf{sub}\,\varsigma \circ \mathsf{sub}\,\sigma$ is the equality of the composites

$$\mathbf{T} \otimes (\mathbf{T} \otimes \mathbf{T}) \xrightarrow{\alpha_{\mathbf{T},\mathbf{T},\mathbf{T}}} (\mathbf{T} \otimes \mathbf{T}) \otimes \mathbf{T} \xrightarrow{\mathsf{sub} \otimes \mathbf{T}} \mathbf{T} \otimes \mathbf{T} \xrightarrow{\mathsf{sub}} \mathbf{T}$$
$$\mathbf{T} \otimes (\mathbf{T} \otimes \mathbf{T}) \xrightarrow{\mathbf{T} \otimes \mathsf{sub}} \mathbf{T} \otimes \mathbf{T} \xrightarrow{\mathsf{sub}} \mathbf{T}$$

which are both $(\mathcal{V} + \Sigma)$-algebra homomorphisms and therefore equal the traversal map $\mathsf{trav} : \mathbf{T} \otimes (\mathbf{T} \otimes \mathbf{T}) \to \mathbf{T}$ induced by parametrised initiality with pointed presheaf parameter $(\mathbf{T} \otimes \mathbf{T}, \mathcal{V} \cong \mathcal{V} \otimes \mathcal{V} \xrightarrow{\mathsf{var} \otimes \mathsf{var}} \mathbf{T} \otimes \mathbf{T})$ and $((\mathbf{T} \otimes \mathbf{T}) + \Sigma)$-algebra $(\mathbf{T}, \mathsf{sub}, \mathsf{alg})$.

Models and interpretations A model for the syntax is nothing but a $(\mathcal{V} + \Sigma)$-algebra $\mathcal{M}$: a presheaf which supports variables and the operators of the syntax. The initiality theorem characterising $\mathbf{T}$ as the initial such model gives rise to compositional interpretations $\mathsf{int} : \mathbf{T} \to \mathcal{M}$ directly, satisfying the expected recursive equations. The notion can be further strengthened to also incorporate substitution. A Σ-*monoid* is therefore a monoid $(\mathcal{M}, \eta, \mu)$ with a Σ-algebra structure $a : \Sigma\mathcal{M} \to \mathcal{M}$ that is compatible with the monoid multiplication:

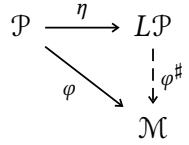
$$\begin{array}{ccccc} \Sigma\mathcal{M} \otimes \mathcal{M} & \xrightarrow{\mathsf{str}} & \Sigma(\mathcal{M} \otimes \mathcal{M}) & \xrightarrow{\Sigma\mu} & \Sigma\mathcal{M} \\ {\scriptstyle a \otimes \mathcal{M}}\downarrow & & & & \downarrow{\scriptstyle a} \\ \mathcal{M} \otimes \mathcal{M} & & \xrightarrow[\mu]{} & & \mathcal{M} \end{array}$$

The traversal axioms for sub exhibit $\mathbf{T}$ as a Σ-monoid. In fact, it is initial in the category of Σ-monoids, meaning that any model $\mathcal{M}$ with a substitution operation that commutes with the constructors in the appropriate way is equipped with a compositional interpretation $\mathsf{int} : \mathbf{T} \to \mathcal{M}$ which *automatically* satisfies the semantic substitution lemma:

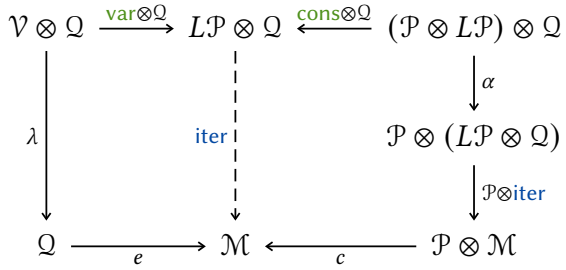
$$\mathsf{int}\,(\mathsf{sub}\,(t, \sigma)) = \mu\,(\mathsf{int}\,t, \mathsf{int} \circ \sigma)$$

Consequently, the soundness of substitution is established as soon as we show that the model is a Σ-monoid, removing the need for proving the inductive compatibility laws between int and renaming, weakening, etc. We will refer to this result as the *initiality theorem*.

Metavariables *Metavariables* are a feature not generally addressed by formalisation, but they appear quite naturally as part of the presheaf model. Above we argued that the initial $(\Sigma + \mathcal{V})$-algebra $\mathbf{T}$ is an initial Σ-monoid, which allowed us to define compositional interpretations in models that support substitution. A natural question to ask is whether there is a free construction that turns any presheaf into a Σ-monoid, just how the list constructor turns any set into a free set-theoretic monoid. A free monoid on $\mathcal{P}$ in the monoidal category of presheaves $(\mathbf{PSh}_S, \mathcal{V}, \otimes)$ would be a presheaf $L\mathcal{P}$ with a natural transformation $\eta : \mathcal{P} \to L\mathcal{P}$ such that for any monoid $\mathcal{M} \in \mathbf{PSh}_S$ and map $\varphi : \mathcal{P} \to \mathcal{M}$, there exists a unique monoid homomorphism $\varphi^\sharp : L\mathcal{P} \to \mathcal{M}$ factorising $\varphi = \varphi^\sharp \circ \eta$:



A *list object* (Cockett, 1990; Fiore and Saville, 2017), if it exists, is a free monoid: on a presheaf $\mathcal{P}$, it consists of morphisms $\mathsf{var} : \mathcal{V} \to L\mathcal{P}$ and $\mathsf{cons} : \mathcal{P} \otimes L\mathcal{P} \to L\mathcal{P}$ such that for all objects $\mathcal{Q}, \mathcal{M}$ with $n : \mathcal{Q} \to \mathcal{M}$ and $c : \mathcal{P} \otimes \mathcal{M} \to \mathcal{M}$, there exists a unique map $\mathsf{iter} : L\mathcal{P} \otimes \mathcal{Q} \to \mathcal{M}$ satisfying the parametrised-initiality condition:



To generalise this to a list object with algebraic structure, we also ask for an algebra constructor $\mathsf{alg} : \Sigma L\mathcal{P} \to L\mathcal{P}$; since this, with the "empty list" constructor $\mathsf{var} : \mathcal{V} \to L\mathcal{P}$, makes $L\mathcal{P}$ into a $(\Sigma + \mathcal{V})$-algebra, we are justified in thinking of the object as the presheaf of terms $\mathbf{T}\mathcal{P}$ with an additional constructor $\mathsf{cons} : \mathcal{P} \otimes \mathbf{T}\mathcal{P} \to \mathbf{T}\mathcal{P}$, which we will write as mvar from now on.

The intuitive interpretation of the constructor $\mathsf{mvar} : \mathcal{P} \otimes \mathbf{T}\mathcal{P} \to \mathbf{T}\mathcal{P}$ was first suggested by Hamana (2004): the opaque, non-syntactic term $\mathfrak{m} \in \mathcal{P}_\tau\Pi$ is a *metavariable*, and the associated substitution rule assigning types in Π to syntactic terms in context Γ is a *metavariable environment*. Parametrised metavariables $\mathsf{mvar}\,(\mathfrak{m}, \langle t_1, \ldots, t_n\rangle)$, also denoted $\mathfrak{m}\langle t_1, \ldots, t_n\rangle$, allow one to construct syntactic terms with holes that stand for arbitrary expressions, with the holes having a predefined number of slots that can be populated with terms of a predefined type. The type and number of the metavariable slots is predetermined by the metavariable context Π. For example, for elements $\mathfrak{m} \in \mathcal{P}_\beta[\alpha]$ and $\mathfrak{n} \in \mathcal{P}_\alpha[]$, standing for metavariables $\mathfrak{m} : [\alpha]\beta$ and $\mathfrak{n} : []\alpha$ with one and zero parameters respectively, we can construct $\mathbf{T}$-terms

$$\mathsf{app}\,\big(\mathsf{lam}\,(\mathsf{mvar}\,(\mathfrak{m}, \langle \mathsf{var\ new}\rangle)), \mathsf{mvar}(\mathfrak{n}, \langle\rangle)\big) \qquad \mathsf{mvar}\,(\mathfrak{m}, \langle \mathsf{mvar}\,(\mathfrak{n}, \langle\rangle)\rangle)$$

that encode the syntax of the informal meta-terms $(\lambda x : \alpha.\, \mathfrak{m}\{x\})(\mathfrak{n})$ and $\mathfrak{m}\{\mathfrak{n}\}$. Metavariables provide a formal way to refer to abstract terms that can be instantiated with concrete syntactic expressions, and *parametrised* metavariables are further associated with an environment of terms that are substituted for the free variables of the instantiating expression. For exam-

ple, instantiating $\mathfrak{m}\{y\}$ above with $f(y)$ for some term $f : \alpha \to \beta$ results in $(\lambda x : \alpha.\ f(x))(\mathfrak{n})$ and $f(\mathfrak{n})$: in the first, the actual parameter x bound to the λ is plugged into the y formal parameter of the metavariable; in the second, we replace y with the term consisting of the metavariable $\mathfrak{n}$. It is worth emphasising that instantiation is not standard substitution, as it is *capture-permitting*: given a second-order term

$$\mathfrak{m} : [\mathbb{N}](\mathbb{N} \to \mathbb{N}) \ \triangleright\ f : \alpha \to \mathbb{N} \ \vdash\ \lambda x : \alpha.\ (\mathfrak{m}\{f\,x\})\,(f\,x) : \mathbb{N}$$

and the instantiation $\mathfrak{m}\{x\} \mapsto \lambda y : \mathbb{N}.\ x + y$, the resulting term is

$$\emptyset \ \triangleright\ f : \alpha \to \mathbb{N} \ \vdash\ \lambda x : \alpha.\ (\lambda y : \mathbb{N}.\ (f\,x) + y)\,(f\,x) : \mathbb{N}$$

and the bound variable x maintains the binding to $f\,x$.

One application of metavariables and instantiation is the specification of second-order equational systems (Fiore and Hur, 2010), which construct a sound and complete equational logic from a set of second-order axioms, namely a pair of terms with metavariables which abstract over all instantiations of the axiom. For example, the β and η-rules of STLC and the first-order universal-conjunction equivalence can be represented as the axioms

$$\begin{aligned}
\mathfrak{m} : [\alpha]\beta, \mathfrak{n} : []\alpha \ \triangleright\ \emptyset \ \vdash\ (\lambda x : \alpha.\ \mathfrak{m}\{x\})(\mathfrak{n}) \ &\equiv\ \mathfrak{m}\{\mathfrak{n}\} && : \beta \\
\mathfrak{f} : [](\alpha \to \beta) \ \triangleright\ \emptyset \ \vdash\ \lambda x : \alpha.\ \mathfrak{f}\,x \ &\equiv\ \mathfrak{f} && : \alpha \to \beta \\
\mathfrak{p} : []\mathbb{B}, \mathfrak{q} : [\mathbb{N}]\mathbb{B} \ \triangleright\ \emptyset \ \vdash\ \mathfrak{p} \wedge (\forall n \in \mathbb{N}.\ \mathfrak{q}\{n\}) \ &\equiv\ \forall n \in \mathbb{N}.\ (\mathfrak{p} \wedge \mathfrak{q}\{n\}) : \mathbb{B}
\end{aligned}$$

Note that parametrised metavariables give us a clear control over what variables are free and bound in each metavariable: for the η-rule, the metavariable $\mathfrak{f}$ has no parameters and therefore does not bind the x, and for the FOL axiom, only $\mathfrak{q}$ refers to the quantified variable as $\mathfrak{p}$ has no parameters it can connect to.

As shown in Corollary 4 of Fiore (2008) and Lemma 5.6 of Fiore and Saville (2017), the initial $(\Sigma + \mathcal{V} + \mathcal{P}\otimes)$-algebra – the syntax of terms that support constructors, variables, and metavariables – is a Σ-list object, which is furthermore the free Σ-monoid: for all Σ-monoids $\mathcal{M}$ with interpretation $\varphi : \mathcal{P} \to \mathcal{M}$, there is a unique Σ-monoid homomorphism $\varphi^\sharp : \mathbf{T}\mathcal{P} \to \mathcal{M}$ such that $\varphi = \varphi^\sharp \circ \mathsf{emb}$, where $\mathsf{emb} : \mathcal{P} \to \mathbf{T}\mathcal{P}$ maps $\mathfrak{m} \in \mathcal{P}_\tau\Pi$ to $\mathsf{mvar}(\mathfrak{m}, \mathsf{var})$. In other words, given a model of the syntax $\mathcal{M}$, and an interpretation for the metavariables $\varphi : \mathcal{P} \to \mathcal{M}$, there is a unique extension $\varphi^\sharp = \mathsf{int}_\varphi : \mathbf{T}\mathcal{P} \to \mathcal{M}$ which satisfies

$$\begin{array}{ccc}
\mathcal{V} & & \\
{\scriptstyle \mathsf{var}}\downarrow & \searrow^{\eta} & \\
\mathbf{T}\mathcal{P} & \xrightarrow[\mathsf{int}_\varphi]{} & \mathcal{M}
\end{array}
\qquad
\begin{array}{ccc}
\Sigma\mathbf{T}\mathcal{P} & \xrightarrow{\Sigma\mathsf{int}_\varphi} & \Sigma\mathcal{M} \\
{\scriptstyle \mathsf{alg}}\downarrow & & \downarrow{\scriptstyle a} \\
\mathbf{T}\mathcal{P} & \xrightarrow[\mathsf{int}_\varphi]{} & \mathcal{M}
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{P} \otimes \mathbf{T}\mathcal{P} & \xrightarrow{\varphi\otimes\mathsf{int}_\varphi} & \mathcal{M} \otimes \mathcal{M} \\
{\scriptstyle \mathsf{mvar}}\downarrow & & \downarrow{\scriptstyle \mu} \\
\mathbf{T}\mathcal{P} & \xrightarrow[\mathsf{int}_\varphi]{} & \mathcal{M}
\end{array}$$

$$\begin{aligned}
\mathsf{int}_\varphi\,(\mathsf{var}\,x) &= \eta\,x \\
\mathsf{int}_\varphi\,(\mathsf{alg}\,t) &= a\,(\Sigma\,\mathsf{int}_\varphi\,b) \\
\mathsf{int}_\varphi\,(\mathsf{mvar}\,(\mathfrak{m}, \varepsilon)) &= \mu\,(\varphi\,\mathfrak{m}, \mathsf{int}_\varphi \circ \varepsilon)
\end{aligned}$$

We will refer to this as the *freeness theorem*; it can be instantiated to the initiality theorem on page 40 with $\mathcal{P} \triangleq \bot$ and the initial map $\bot \to \mathcal{M}$, which extends to $\mathbf{T}\bot \to \mathcal{M}$, the compositional interpretation of terms without any metavariables. When metavariables are present, they are mapped to elements of the model into which the interpretation of the metavariable environment is substituted.

One subtle question is the categorical representation of the metavariable family in the presheaf model. Every presheaf admits renaming, but this works against our intention to rigidly enumerate the metavariables featured in an equation or theory. For example, we may want to collect every metavariable in the examples above into a single family:

$$\mathfrak{m} : \mathcal{P}_\beta[\alpha] \qquad \mathfrak{n} : \mathcal{P}_\alpha[] \qquad \mathfrak{f} : \mathcal{P}_{\alpha\to\beta}[] \qquad \mathfrak{p} : \mathcal{P}_{\mathbb{B}}[] \qquad \mathfrak{q} : \mathcal{P}_{\mathbb{B}}[\mathbb{N}]$$

To fit in the presheaf model, we also need to define the arrow mapping of the presheaf $\mathcal{P}$, but it is not at all clear how to do this for the elements above; and even if we did, the use for such a renaming operation would be uncertain. The mvar operator is defined in terms of $\otimes$ and therefore performs the usual quotienting:

$$(\mathcal{P}\rho\mathfrak{m})\langle\varepsilon\rangle = \mathfrak{m}\{\varepsilon \circ \rho\}$$

but again, the meaning and utility of this is not clear as it introduces a malleability to metavariables that we would rather avoid.

Hamana's (2004) suggestion is to keep metavariables in indexed families of sets $\mathbf{Fam}_S \triangleq (\mathbf{Set}^{\mathbb{F}[S]})^S$, but freely equip this family (henceforth denoted $\mathfrak{A}, \mathfrak{B}$) with a presheaf structure

$$\overline{\mathfrak{A}}_\alpha\Delta \triangleq \sum_{\Gamma\in S^*} \mathbb{F}(\Gamma,\Delta) \times \mathfrak{A}_\alpha\Gamma$$

This associates a metavariable with a renaming rule, and the presheaf action of $\overline{\mathfrak{A}}$ simply alters the renaming rule rather than altering the context of the metavariable. Thus, the family $\mathfrak{A}$ can be embedded into the presheaf framework without a significant change to its role as an enumeration of metavariables. Composing the free presheaf and free Σ-monoid functors gives the free Σ-monoid on sorted families $\mathfrak{A} \mapsto \mathbf{T}\overline{\mathfrak{A}}$ (which we still denote $\mathbf{T} : \mathbf{Fam}_S \to \Sigma\text{-}\mathbf{Mon}$) computed on $\overline{\mathfrak{A}}$ as the initial $(\Sigma + \mathcal{V} + \overline{\mathfrak{A}}\otimes)$-algebra.

Metasubstitution The instantiation of metavariables – the metasubstitution operation – combines a term in $\mathbf{T}\mathcal{P}$ with a mapping of metavariables in $\mathcal{P}$ to terms in $\mathbf{T}\mathcal{Q}$ to obtain a term in $\mathbf{T}\mathcal{Q}$. It operates in a way very similar to object-level substitution, but one level up: the metasubstitution rule $\mathcal{P} \to \mathbf{T}\mathcal{Q}$ is lifted to a mapping $\mathbf{T}\mathcal{P} \to \mathbf{T}\mathcal{Q}$. However, since the category of presheaves is closed, the operation has an external and internal form (introduced by Hamana (2004) and Fiore (2008), respectively), with important practical differences between the two.

The *external metasubstitution rule* is the hom-set $\mathbf{PSh}_S(\mathcal{P}, \mathbf{T}\mathcal{Q})$, and the *external metasubstitution operation* is the Kleisli extension

$$\mathbf{PSh}_S(\mathcal{P}, \mathbf{T}\mathcal{Q}) \to \mathbf{PSh}_S(\mathbf{T}\mathcal{P}, \mathbf{T}\mathcal{Q})$$

Concretely, given a Σ-monoid $\mathcal{M}$, the freeness of the free Σ-monoid functor $\mathbf{T}$ amounts to

a bijection between natural transformations $\varphi \colon \mathcal{P} \Longrightarrow \mathcal{M}$, and Σ-monoid homomorphisms $\mathsf{int}_\varphi \in \Sigma\text{-}\mathbf{Mon}(\mathbf{T}\mathcal{P}, \mathcal{M})$ which, for $\mathcal{M} = \mathbf{T}\mathcal{Q}$, induces the metasubstitution operation. In addition, for $\mathcal{P} = \mathbf{T}\mathcal{Q}$, the extension of the identity $\mathbf{T}\mathcal{Q} \to \mathbf{T}\mathcal{Q}$ is the monad multiplication $\mathsf{join} = \mathsf{int}_{\mathrm{id}} \colon \mathbf{T}(\mathbf{T}\mathcal{Q}) \to \mathbf{T}\mathcal{Q}$, exhibiting $\mathbf{T}$ as a monad, as expected:

$$\begin{aligned} \mathsf{join}\,(\mathsf{var}\,x) &= \mathsf{var}\,x \\ \mathsf{join}\,(\mathsf{alg}\,t) &= \mathsf{alg}\,(\Sigma\,\mathsf{join}\,b) \\ \mathsf{join}\,(\mathsf{mvar}\,(t,\varepsilon)) &= \mathsf{sub}\,(t, \mathsf{join} \circ \varepsilon) \end{aligned}$$

The external metasubstitution operation is too limited for the intended purpose. An external metasubstitution rule $\mathcal{P} \to \mathbf{T}\mathcal{Q}$ assigns every metavariable $\mathbb{m} \in \mathcal{P}_\tau\Pi$ to a term $\mathbf{T}\mathcal{Q}_\tau\Pi$, so the target of the rule may only refer to variables in the parameter context; in particular, metavariables with no parameters may only be instantiated with closed terms. For example, in $\mathbb{m} \colon [\mathbb{N}]\alpha \triangleright x \colon \mathbb{N}, y \colon \mathbb{N} \vdash \mathbb{m}\{y\} \colon \alpha$, we cannot instantiate $\mathbb{m}\{y\}$ with $x+y$ as the target term of the metasubstitution can only refer to the y parameter of $\mathbb{m}$. This is restrictive considering x is in scope throughout the whole term.

The solution is to *internalise* the metasubstitution operation as a natural transformation between presheaf exponentials, rather than a function between homsets – this is the key contribution of the two-level substitution calculus of second-order abstract syntax as developed by Fiore (2008). Temporarily moving to the unsorted setting of the paper, an *internal metasubstitution rule* is the presheaf exponential $P \supset \mathbf{T}Q \in \mathbf{PSh}$, and the *internal metasubstitution operation* is a natural transformation between exponentials

$$(P \supset \mathbf{T}Q) \Longrightarrow (\mathbf{T}P \supset \mathbf{T}Q) \in \mathbf{PSh}$$

also presentable, in uncurried form, as $\mathbf{T}P \times (P \supset \mathbf{T}Q) \Longrightarrow \mathbf{T}Q$. The presheaf exponential $P \supset \mathbf{T}Q$ is defined at all Γ as the hom-set $\mathbf{PSh}\big(\text{ɬ}\Gamma \times P, \mathbf{T}Q\big)$, which can be curried to $\mathbf{PSh}\big(P, \text{ɬ}\Gamma \supset \mathbf{T}Q\big) \cong \mathbf{PSh}(P, \delta_\Gamma \mathbf{T}Q)$. Thus, the type of the metasubstitution operator at component Γ expands as

$$\mathbf{T}P(\Gamma) \times \Big(\prod_{\Pi \in \mathbb{F}} P(\Pi) \to \mathbf{T}Q(\Gamma + \Pi)\Big) \to \mathbf{T}Q(\Gamma)$$

As intended, the ambient context Γ is available in the $\mathbf{T}Q$-terms in the target of the rule, alongside the metavariable parameter contexts Π: in $\mathbb{m} \colon [\mathbb{N}]\alpha \triangleright x : \mathbb{N}, y : \mathbb{N} \vdash \mathbb{m}\{y\} \colon \alpha$, we can now instantiate $\mathbb{m}\{y\}$ with $x : \mathbb{N}, y : \mathbb{N} \vdash x + y \colon \mathbb{N}$. Note that we're only extending the parameter context with the overall context of the term, not the particular variable scope at the point of instantiation: with $\mathbb{m} \colon [\mathbb{N}]\alpha \triangleright y : \mathbb{N} \vdash \lambda x : \mathbb{N}.\ \mathbb{m}\{y\} \colon \alpha$, we still cannot instantiate $\mathbb{m}\{y\}$ with $x + y$, even though x is in scope under the binder. This is the intended behaviour for constraining variables to not be free in metavariables, otherwise nothing would stop us from instantiating the η rule $\lambda x : \alpha.\ (\mathfrak{f})\, x \equiv \mathfrak{f}$ with $\mathfrak{f} \mapsto x$ and deduce that $(\lambda x : \alpha.\ x\, x) \equiv x$.

The reintroduction of sorts requires more care: we cannot simply quantify over sorts across the whole metasubstitution operation, as the sorts of the metavariables are different from the

sort of the host term. The operation becomes, for all $\alpha \in S$ and $\Gamma \in \mathbb{F}[S]$:

$$\mathbf{T}\mathcal{P}_\alpha\Gamma \times \Big(\prod_{\tau\in S,\Pi\in\mathbb{F}[S]} \mathcal{P}_\tau\Pi \to \mathbf{T}\mathcal{Q}_\tau(\Gamma+\Pi)\Big) \to \mathbf{T}\mathcal{Q}_\alpha\Gamma$$

By writing $\langle\mathcal{P},\mathcal{Q}\rangle \in \mathbf{PSh}$ for the product of exponentials $\langle\mathcal{P},\mathcal{Q}\rangle \triangleq \prod_{\tau\in S^*} \mathcal{P}_\alpha \supset \mathcal{Q}_\alpha$ and $(-)*(=)\colon \mathbf{PSh}_S \times \mathbf{PSh} \to \mathbf{PSh}_S$ for the action $(\mathcal{P}*Q)_\alpha \triangleq \mathcal{P}_\alpha \times Q$, the type is concisely expressed as the natural transformation

$$\mathbf{T}\mathcal{P} * \langle\mathcal{P},\mathbf{T}\mathcal{Q}\rangle \Longrightarrow \mathbf{T}\mathcal{Q}$$

Alternatively, we can take $\mathbf{PSh}_S$ as $\mathbf{PSh}$-enriched, with $\langle\mathcal{P},\mathcal{Q}\rangle$ above defining the hom-presheaf of two sorted presheaves. Denoting it as $\mathbf{PSh}_S\langle\mathcal{P},\mathcal{Q}\rangle$, the metasubstitution operation then takes the form of the hom-presheaf morphism (natural transformation)

$$\mathbf{PSh}_S\langle\mathcal{P},\mathbf{T}\mathcal{Q}\rangle \Longrightarrow \mathbf{PSh}_S\langle\mathbf{T}\mathcal{P},\mathbf{T}\mathcal{Q}\rangle \in \mathbf{PSh}$$

which resembles the type of the external metasubstitution, but now in the presheaf-enriched setting. Generalising the target $\mathbf{T}\mathcal{Q}$ to an arbitrary Σ-monoid $\mathcal{M}$, and specialising the presheaf of metavariables $\mathcal{P}$ to the free presheaf $\overline{\mathfrak{A}}$ generated from a metavariable family $\mathfrak{A}$, we have the equivalent specifications of meta-extension as

$$\mathbf{T}\mathfrak{A} * \langle\overline{\mathfrak{A}},\mathcal{M}\rangle \Longrightarrow \mathcal{M} \qquad \mathbf{PSh}_S\langle\overline{\mathfrak{A}},\mathcal{M}\rangle \Longrightarrow \mathbf{PSh}_S\langle\mathbf{T}\mathfrak{A},\mathcal{M}\rangle$$

Of these, the second one (together with the unit $\mathbf{PSh}_S\langle\overline{\mathfrak{A}},\mathbf{T}\mathfrak{A}\rangle$) exhibits $\mathbf{T}\colon \mathbf{Fam}_S \to \Sigma\text{-}\mathbf{Mon}$ as a $(\overline{\cdot})$-*relative* $\mathbf{PSh}$-*enriched monad*, which also informs the laws the operation must satisfy.

How exactly is the internal/enriched metasubstitution defined? We present both the derivation given by Fiore (2008), and an alternative derivation focusing on the enriched setting.

To construct the (uncurried) metasubstitution operation

$$\mathbf{T}\mathcal{P} * \langle\mathcal{P},\mathbf{T}\mathcal{Q}\rangle \to \mathbf{T}\mathcal{Q}$$

we induce the *cartesian strength* for the free Σ-monoid monad $\mathbf{T}$: for all $\mathcal{P} \in \mathbf{PSh}_S$ and $Q \in \mathbf{PSh}$,

$$\mathsf{cstr}\colon \mathbf{T}\mathcal{P} * Q \to \mathbf{T}(\mathcal{P} * Q)$$

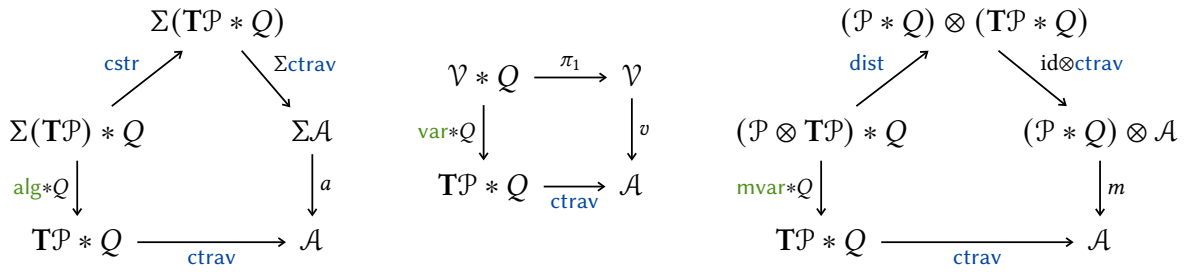
From this, we internalise the functorial action of $\mathbf{T}$ as the *meta-renaming* map

$$\mathsf{mren}\colon \mathbf{T}\mathcal{P} * \langle\mathcal{P},\mathcal{Q}\rangle \xrightarrow{\mathsf{cstr}} \mathbf{T}(\mathcal{P} * \langle\mathcal{P},\mathcal{Q}\rangle) \xrightarrow{\mathbf{T}\mathsf{eval}} \mathbf{T}\mathcal{Q}$$

and from here, derive internal metasubstitution via join:

$$\mathsf{msub}\colon \mathbf{T}\mathcal{P} * \langle\mathcal{P},\mathbf{T}\mathcal{Q}\rangle \xrightarrow{\mathsf{mren}} \mathbf{T}(\mathbf{T}\mathcal{Q}) \xrightarrow{\mathsf{join}} \mathbf{T}\mathcal{Q}$$

A cartesian strength for the signature endofunctor $\Sigma\mathcal{P} * Q \to \Sigma(\mathcal{P} * Q)$ can be extended to a cartesian strength $\mathbf{T}\mathcal{P} * Q \to \mathbf{T}(\mathcal{P} * Q)$. This is induced by *cartesian-parametrised initiality*: given a presheaf Q and a $(\Sigma + \mathcal{V} + (\mathcal{P} * Q)\otimes)$-algebra $(\mathcal{A}, [a, v, m])$, there exists a unique *cartesian traversal* operation $\mathsf{ctrav}\colon \mathbf{T}\mathcal{P} * Q \to \mathcal{A}$ satisfying



Here, the sorted version of a map $\mathsf{dist}\colon (P \oslash \mathcal{Q}) \times R \to (P \times R) \oslash (\mathcal{Q} * R)$ for a pointed presheaf $(\mathcal{Q}, \eta\colon \mathcal{V} \to \mathcal{Q})$ encompasses a distributivity property between the cartesian product and substitution tensor, implemented with a combination of weakening, copairing of $(\mathcal{Q} * R)$-valued substitution rules, and variable embedding:

$$\mathsf{dist}\big((\Gamma, t \in P(\Gamma), \sigma\colon {}^{\Gamma}\mathcal{P}_{\Delta}), r \in R\Delta\big) \triangleq ((\Delta + \Gamma),\ \ (P\langle \iota_1^{\Gamma,\Delta}\rangle\, t, R\langle \iota_2^{\Gamma,\Delta}\rangle\, r), \\ {}_{\mathcal{Q}*R}[x \mapsto (\eta\, x, r),\ y \mapsto (\sigma\, y, r)]_{\Delta}^{\Delta,\Gamma} \in {}^{\Delta+\Gamma}(\mathcal{Q} * R)_{\Delta})$$

Instantiating this traversal with $\mathcal{A} = \mathbf{T}(\mathcal{P} * Q)$, with $m = \mathsf{mvar}\colon (\mathcal{P} * Q) \otimes \mathbf{T}(\mathcal{P} * Q) \to \mathbf{T}(\mathcal{P} * Q)$, we get the cartesian strength $\mathsf{cstr}\colon \mathbf{T}\mathcal{P} * Q \to \mathbf{T}(\mathcal{P} * Q)$ for the syntax monad $\mathbf{T}$; the unit and associativity axioms depend on those of the cartesian strength for Σ, and similar laws for dist. To show that $\mathbf{T}$ is furthermore a *strong monad*, we require the strengths for Σ to be compatible:

$$\begin{array}{ccccc} (\Sigma\mathcal{P} \otimes \mathcal{Q}) * R & \xrightarrow{\mathsf{str}\otimes R} & \Sigma(\mathcal{P} \otimes \mathcal{Q}) * R & \xrightarrow{\mathsf{cstr}} & \Sigma((\mathcal{P} \otimes \mathcal{Q}) * R) \\ \downarrow{\scriptstyle \mathsf{dist}} & & & & \downarrow{\scriptstyle \Sigma\mathsf{dist}} \\ (\Sigma\mathcal{P} * R) \otimes (\mathcal{Q} * R) & \xrightarrow[\mathsf{cstr}\otimes\mathsf{id}]{} & \Sigma(\mathcal{P} * R) \otimes (\mathcal{Q} * R) & \xrightarrow[\mathsf{str}]{} & \Sigma((\mathcal{P} * R) \otimes (\mathcal{Q} * R)) \end{array}$$

From there, meta-renaming and metasubstitution follow:

$$\mathsf{msub}\colon \mathbf{T}\mathcal{P} * \langle \mathcal{P}, \mathbf{T}\mathcal{Q}\rangle \xrightarrow{\mathsf{cstr}} \mathbf{T}\big(\mathcal{P} * \langle \mathcal{P}, \mathbf{T}\mathcal{Q}\rangle\big) \xrightarrow{\mathbf{T}\,\mathsf{eval}} \mathbf{T}(\mathbf{T}\mathcal{Q}) \xrightarrow{\mathsf{join}} \mathbf{T}\mathcal{Q}$$

Moreover, one can show that the forgetful functor $\Sigma\text{-}\mathbf{Mon} \to \mathbf{PSh}_S$ is monadic, establishing an isomorphism between Σ-monoids and $\mathbf{T}$-algebras $\Sigma\text{-}\mathbf{Mon} \cong \mathbf{T}\text{-}\mathbf{Alg}$. With this, the metasubstitution operation is an instance of the *meta-interpretation* operation $\mathsf{mint}\colon \mathbf{T}\mathcal{P} * \langle \mathcal{P}, \mathcal{M}\rangle \to \mathcal{M}$ that internalises $\mathsf{int}_{\varphi}\colon \mathbf{T}\mathcal{P} \to \mathcal{M}$ for $\mathcal{M}$ a Σ-monoid (equivalently a $\mathbf{T}$-algebra $c\colon \mathbf{T}\mathcal{M} \to \mathcal{M}$) and $\varphi\colon \mathcal{P} \to \mathcal{M}$ a metavariable interpretation:

$$\mathsf{mint}\colon \mathbf{T}\mathcal{P} * \langle \mathcal{P}, \mathcal{M}\rangle \xrightarrow{\mathsf{cstr}} \mathbf{T}\big(\mathcal{P} * \langle \mathcal{P}, \mathcal{M}\rangle\big) \xrightarrow{\mathbf{T}\,\mathsf{eval}} \mathbf{T}\mathcal{M} \xrightarrow{c} \mathcal{M}$$

Equivalently, mint can be induced via parametrised initiality immediately: instantiating the traversal with $Q \triangleq \langle \mathcal{P}, \mathcal{M}\rangle$ and $(\mathcal{A}, a, v, m) \triangleq (\mathcal{M}, a, \eta, m)$, with the metavariable operation

$$m\colon (\mathcal{P} * \langle \mathcal{P}, \mathcal{M}\rangle) \otimes \mathcal{M} \xrightarrow{\mathsf{eval}\otimes\mathcal{M}} \mathcal{M} \otimes \mathcal{M} \xrightarrow{\mu} \mathcal{M}$$

The corresponding recursive specification gives a clearer idea of the sequence of actions that metasubstitution performs on metavariables:

$$\begin{aligned}
\mathsf{mint}\,(\mathsf{var}\,x, \quad & \zeta) \;=\; \eta\,x \\
\mathsf{mint}\,(\mathsf{alg}\,t, \quad & \zeta) \;=\; a\,(\Sigma\,\mathsf{mint}\,(\mathsf{cstr}\,(t,\zeta))) \\
\mathsf{mint}\,(\mathsf{mvar}\,(\mathfrak{m},\varepsilon),\zeta) \;&=\; \mu\left(\zeta(\mathfrak{m}),[\eta,\ y \mapsto \mathsf{mint}\,(\varepsilon\,y,\zeta)]_\Gamma^{\Gamma,\Pi}\right)
\end{aligned}$$

First, $\mathfrak{m} : \mathcal{P}_\tau(\Pi)$ is looked up in the meta-interpretation rule $\zeta : \langle \mathcal{P}, \mathcal{M}\rangle\Gamma$ to obtain a term $\zeta(\mathfrak{m}) : \mathcal{M}_\tau(\Gamma + \Pi)$. Then we perform an object-level substitution to populate the metavariable parameter variables in Π with terms given in the metavariable environment ε, while mapping the existing variables in Γ to themselves with η. Since the metavariable environment terms may themselves contain metavariables, we need to apply ζ to them recursively. As an example, take the open term in object context $x : \mathbb{N}$ and metavariable context $\mathfrak{A} = \mathfrak{m} : [\mathbb{N},\mathbb{N}]\mathbb{N}, \mathfrak{n} : [\mathbb{N}]\mathbb{N}\mathfrak{p} : []\mathbb{N}$, and metasubstitution rule from $\mathfrak{A}$ to terms in $\mathfrak{B} \triangleq \mathfrak{q} : [\mathbb{N}]\mathbb{N}$:

$$\triangleright\ x : \mathbb{N} \vdash (\lambda y : \mathbb{N}.\ \mathfrak{m}\{x+1, \mathfrak{n}\{y\}\})\ \mathfrak{p} : \mathbb{N}$$
$$\zeta = (\mathfrak{m}\{m,n\} \mapsto \mathfrak{q}\{m\} \times n, \mathfrak{n}\{m\} \mapsto \mathfrak{q}\{m+x\}, \mathfrak{p} \mapsto 2)$$

The metasubstitution is calculated as follows:

$$\begin{aligned}
&\mathsf{msub}\big((\lambda y : \mathbb{N}.\ \mathfrak{m}\{x+1, \mathfrak{n}\{y\}\})\ \mathfrak{p}, \zeta\big) \\
&= \mathsf{msub}(\lambda y : \mathbb{N}.\ \mathfrak{m}\{x+1, \mathfrak{n}\{y\}\}, \zeta)\ (\mathsf{msub}(\mathfrak{p}, \zeta)) && ① \\
&= (\lambda y : \mathbb{N}.\ \mathsf{msub}(\mathfrak{m}\{x+1, \mathfrak{n}\{y\}\}, \mathsf{wkn}\,\zeta))\ 2 && ② \\
&= \big(\lambda y : \mathbb{N}.\ \mathsf{sub}\,(\mathfrak{q}\{m\} \times n, [m \mapsto x+1, n \mapsto \mathsf{msub}\,(\mathfrak{n}\{y\}, \mathsf{wkn}\,\zeta)])\big)\ 2 && ③ \\
&= \big(\lambda y : \mathbb{N}.\ \mathsf{sub}\,(\mathfrak{q}\{m\} \times n, [m \mapsto x+1, n \mapsto \mathsf{sub}\,(\mathfrak{q}\{m+x\}, [m \mapsto y])])\big)\ 2 && ④ \\
&= \big(\lambda y : \mathbb{N}.\ \mathsf{sub}\,(\mathfrak{q}\{m\} \times n, [m \mapsto x+1, n \mapsto \mathfrak{q}\{y+x\}])\big)\ 2 && ⑤ \\
&= \big(\lambda y : \mathbb{N}.\ \mathfrak{q}\{x+1\} \times \mathfrak{q}\{y+x\}\big)\ 2 && ⑥
\end{aligned}$$

In steps ①-② we push the metasubstitution rule into the application and under the binder. A crucial component of this step is the weakening of the terms in the metasubstitution rule ζ, since the local context changes from $x : \mathbb{N}$ to $y : \mathbb{N}, x : \mathbb{N}$. In step ② we also apply ζ to $\mathfrak{p}$, which simply replaces it with 2 without any changes as $\mathfrak{p}$ has no parameters. In step ③ we look up the mapping from $\mathfrak{m}$ to the term $\mathfrak{q}\{m\} \times n$, substituting the contents of $\mathfrak{m}$'s metavariable environment for m and n, and recursively metasubstituting $\mathsf{wkn}\,\zeta$ into $\mathfrak{n}\{y\}$. Steps ④ and ⑤ evaluate the recursive calls, applying an object-level substitution to $\mathfrak{q}\{m+x\}$ that replaces the parameter m with the variable y. The final substitution is evaluated at step ⑥.

The enriched view of metasubstitution considers $\langle \mathcal{P}, \mathcal{Q}\rangle$ as the hom-presheaf in $\mathbf{PSh}_S$ enriched over $\mathbf{PSh}$, and the meta-interpretation as the enriched free extension:

$$\mathsf{mint} : \mathbf{PSh}_S\langle \overline{\mathfrak{A}}, \mathcal{M}\rangle \to \mathbf{PSh}_S\langle \mathbf{T}\mathfrak{A}, \mathcal{M}\rangle$$

This, with $\mathcal{M} \triangleq \mathbf{T}\mathfrak{B}$, specialises to metasubstitution $\mathsf{msub} : \mathbf{PSh}_S\langle \overline{\mathfrak{A}}, \mathbf{T}\mathfrak{B}\rangle \to \mathbf{PSh}_S\langle \mathbf{T}\mathfrak{A}, \mathbf{T}\mathfrak{B}\rangle$ and, with the unit $\mathsf{unit} : \mathbf{PSh}_S\langle \overline{\mathfrak{A}}, \mathbf{T}\mathfrak{A}\rangle$, it exhibits $\mathbf{T} : \mathbf{Fam}_S \to \Sigma\text{-}\mathbf{Mon}$ as a $(\overline{\cdot})$-relative $\mathbf{PSh}$-enriched monad. This characterisation also suggests the appropriate laws of metasubstitution and meta-interpretation, provable using freeness:

$$\mathsf{msub}\,\mathsf{unit} = \mathrm{id} \qquad \mathsf{mint}\,\zeta \circ \mathsf{unit} = \zeta \qquad \mathsf{mint}\,\xi \circ \mathsf{msub}\,\zeta = \mathsf{mint}\,(\mathsf{mint}\,\xi \circ \zeta)$$

The enriched hom $\mathbf{PSh}_s\langle \mathcal{P}, \mathcal{Q}\rangle$ at context Γ expands as

$$\prod_\alpha \mathbf{PSh}\big(\text{ɬ}\Gamma \times \mathcal{P}_\alpha, \mathcal{Q}_\alpha\big) \cong \prod_\alpha \mathbf{PSh}\big(\mathcal{P}_\alpha, \text{ɬ}\Gamma \supset \mathcal{Q}_\alpha\big) \cong \prod_\alpha \mathbf{PSh}(\mathcal{P}_\alpha, \delta_\Gamma \mathcal{Q}_\alpha) = \mathbf{PSh}_s(\mathcal{P}, \delta_\Gamma \mathcal{Q})$$

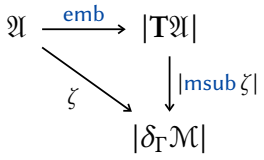
Furthermore, when $\mathcal{P}$ is a free presheaf $\overline{\mathfrak{A}}$, we have the hom-set isomorphism

$$\mathbf{PSh}_s(\overline{\mathfrak{A}}, \delta_\Gamma \mathcal{Q}) \cong \mathbf{Fam}_s(\mathfrak{A}, |\delta_\Gamma \mathcal{Q}|)$$

where $|-|\colon \mathbf{PSh}_s \to \mathbf{Fam}_s$ is the underlying family functor. Altogether, the components of msub are equivalently a family of functions

$$\mathsf{msub}\colon \mathbf{Fam}_s(\mathfrak{A}, |\delta_\Gamma \mathcal{M}|) \to \mathbf{PSh}_s(\mathbf{T}\mathfrak{A}, \delta_\Gamma \mathcal{M})$$

To construct this, it is sufficient to show that δ_Γ lifts to Σ-monoids, and use the free extension to induce a map from $\mathbf{T}\mathfrak{A}$ into the Σ-monoid $\delta_\Gamma \mathcal{M}$:



Since Σ-**Mon** $\cong$ **T-Alg**, the lifting of δ_Γ can be equivalently induced by a distributive law $\mathbf{T}\delta_\Gamma \Longrightarrow \delta_\Gamma \mathbf{T}$, which constructs a $\mathbf{T}$-algebra structure $(\delta_\Gamma \mathcal{M}, \mathbf{T}(\delta_\Gamma \mathcal{M}) \to \delta_\Gamma \mathbf{T}\mathcal{M} \to \delta_\Gamma \mathcal{M})$ for the $\mathbf{T}$-algebra $\mathcal{M}$. Then, as $\delta_\Gamma \mathcal{Q} \cong \text{ɬ}\Gamma \supset \mathcal{Q}$ (where we overload notation for sorted and semi-sorted exponentials $(P \supset \mathcal{Q})_\tau \triangleq P \supset \mathcal{Q}_\tau$) the distributive law equivalently has components $\mathbf{T}(\text{ɬ}\Gamma \supset \mathcal{Q}) \to (\text{ɬ}\Gamma \supset \mathbf{T}\mathcal{Q})$ which is an instance of the *exponential strength*, equivalent to the cartesian strength $\mathbf{T}\mathcal{P} * Q \to \mathbf{T}(\mathcal{P} * Q)$ constructed above.

$$\mathbf{T}(P \supset \mathcal{Q}) \to (P \supset \mathbf{T}\mathcal{Q})$$

Equational logic Using the two-level substitution theory of second-order abstract syntax, Fiore and Hur (2010) introduce *second-order equational logic*, building on the abstract theory of equational systems by Fiore and Hur (2008) and further generalised by Fiore (2013). It allows one to concisely define an equational theory for a particular syntax, only specifying the interesting axioms, from which a congruent equivalence relation is generated.

Given a set of axioms of the form $\mathfrak{A} \triangleright \Gamma \vdash s = t\colon \alpha$, we generate an *equational theory* built on the set of axioms with the following inductive definition:

$$\textsc{ax}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash s = t\colon \alpha}{\mathfrak{A} \triangleright \Gamma \vdash s \equiv t\colon \alpha}$$

$$\textsc{refl}\ \frac{}{\mathfrak{A} \triangleright \Gamma \vdash s \equiv s\colon \alpha} \qquad \textsc{sym}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash s \equiv t\colon \alpha}{\mathfrak{A} \triangleright \Gamma \vdash t \equiv s\colon \alpha} \qquad \textsc{trans}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash s \equiv t\colon \alpha \qquad \mathfrak{A} \triangleright \Gamma \vdash t \equiv u\colon \alpha}{\mathfrak{A} \triangleright \Gamma \vdash s \equiv u\colon \alpha}$$

$$\textsc{msub}\ \frac{\mathfrak{A} \triangleright \Delta \vdash s \equiv t\colon \alpha \qquad \zeta, \xi\colon \mathfrak{A} \to \delta_\Gamma(\mathbf{T}\mathfrak{B}) \qquad \forall \mathfrak{m} \in \mathfrak{A}_\tau \Pi.\ \mathfrak{B} \triangleright \Gamma + \Pi \vdash \zeta(\mathfrak{m}) \equiv \xi(\mathfrak{m})\colon \tau}{\mathfrak{B} \triangleright \Gamma + \Delta \vdash \mathsf{msub}\,\zeta\, s \equiv \mathsf{msub}\,\xi\, t\colon \alpha}$$

The first rule lifts axioms into equivalences, while the middle rules make it into an equivalence relation. The key rule is $\mathfrak{msub}$, which allows for an arbitrary instantiation of an equivalence with metavariables, given equivalent metasubstitution rules. In particular, this enables instantiation of axioms with arbitrary terms that fit the type and context of metavariables:

$$\textsc{msub}\ \frac{\textsc{ax}\ \dfrac{\mathfrak{m} : [\mathbb{N}]\mathbb{N}, \mathfrak{n} : []\mathbb{N} \triangleright \emptyset \vdash (\lambda x : \mathbb{N}.\ \mathfrak{m}\{x\})\mathfrak{n} = \mathfrak{m}\{\mathfrak{n}\} : \mathbb{N}}{\mathfrak{m} : [\mathbb{N}]\mathbb{N}, \mathfrak{n} : []\mathbb{N} \triangleright \emptyset \vdash (\lambda x : \mathbb{N}.\ \mathfrak{m}\{x\})\mathfrak{n} \equiv \mathfrak{m}\{\mathfrak{n}\} : \mathbb{N}} \qquad \zeta = \mathfrak{m}\{x\} \mapsto x + 1, \mathfrak{n} \mapsto 5}{\triangleright \emptyset \vdash (\lambda x : \mathbb{N}.\ x + 1)\ 5 \equiv 5 + 1 : \mathbb{N}}$$

Another use of the $\mathfrak{msub}$ rule is encoding the congruence of the equivalence relation: if $s \equiv t : \alpha$, then $\mathfrak{m}\{s\} \equiv \mathfrak{m}\{t\}$ for any metavariable $\mathfrak{m} : [\alpha]\beta$, which may represent an arbitrarily complex evaluation context. The equational theory then allows us to build a library for second-order equational reasoning, populating it with admissible rules that are proved abstractly. For example, β-reduction with three arguments is derivable as follows:

$$\begin{aligned}
&(\lambda x : \alpha.\ \lambda y : \beta.\ \lambda z : \gamma.\ \mathfrak{m}\{x, y, z\})\ \mathfrak{n}\ \mathfrak{p}\ \mathfrak{q} \\
&= (\lambda y : \beta.\ \lambda z : \gamma.\ \mathfrak{m}\{\mathfrak{n}, y, z\})\ \mathfrak{p}\ \mathfrak{q} && (\beta \text{ with } \mathfrak{m}\{-\} \mapsto \mathfrak{m}\{-, y, z\}, \mathfrak{n} \mapsto \mathfrak{n} \text{ in context } (-)\ \mathfrak{p}\ \mathfrak{q}) \\
&= (\lambda z : \gamma.\ \mathfrak{m}\{\mathfrak{n}, \mathfrak{p}, z\})\ \mathfrak{p}\ \mathfrak{q} && (\beta \text{ with } \mathfrak{m}\{-\} \mapsto \mathfrak{m}\{\mathfrak{n}, -, z\}, \mathfrak{n} \mapsto \mathfrak{p} \text{ in context } (-)\ \mathfrak{q}) \\
&= \mathfrak{m}\{\mathfrak{n}, \mathfrak{p}, \mathfrak{q}\} && (\beta \text{ with } \mathfrak{m}\{-\} \mapsto \mathfrak{m}\{\mathfrak{n}, \mathfrak{p}, -\}, \mathfrak{n} \mapsto \mathfrak{q})
\end{aligned}$$

A model of an equational theory is a Σ-monoid $\mathcal{M}$ that, for all axioms $\mathfrak{A} \triangleright \Gamma \vdash s = t : \alpha$ and metavariable interpretation rules $\omega : \langle \overline{\mathfrak{A}}, \mathcal{M} \rangle$, satisfies the equation $\mathsf{mint}(s, \omega) = \mathsf{mint}(t, \omega)$ in $\mathcal{M}$. We then establish the soundness of the equational logic: if $\mathfrak{A} \triangleright \Gamma \vdash s \equiv t : \alpha$ is a derivable equivalence in the equational theory, then their interpretations in any model of the theory are equal as well. The proof is by induction on the rules of the equational logic: for the $\mathfrak{ax}$ case we use the fact that $\mathcal{M}$ satisfies the axioms by assumption, and for the $\mathfrak{msub}$ case we argue using the associativity law of metasubstitution:

$$\begin{aligned}
\mathsf{mint}(\mathsf{msub}(s, \zeta), \omega) &= \mathsf{mint}(s, \mathfrak{m} \mapsto \mathsf{mint}(\zeta\ \mathfrak{m}, \omega)) && \text{(relative monad associativity)} \\
&= \mathsf{mint}(s, \mathfrak{m} \mapsto \mathsf{mint}(\xi\ \mathfrak{m}, \omega)) && \text{(induction with } \zeta\ \mathfrak{m} \equiv \zeta\ \mathfrak{m} \text{ and } \omega) \\
&= \mathsf{mint}(t, \mathfrak{m} \mapsto \mathsf{mint}(\xi\ \mathfrak{m}, \omega)) && \text{(induction with } s \equiv t \text{ and } \mathfrak{m} \mapsto \mathsf{mint}(\xi\ \mathfrak{m}, \omega)) \\
&= \mathsf{mint}(\mathsf{msub}(t, \xi), \omega) && \text{(relative monad associativity)}
\end{aligned}$$

The generic theory of second-order equational logic also proves that the logic is not only sound, but complete too: if an equivalence is satisfied by every model of the equational presentation, it is also derivable in the logic.

To summarise the presheaf model, we list the main steps of the development:

1. A set of sorts S and second-order signature induces the category of (sorted) presheaves and a signature endofunctor $\Sigma : \mathbf{PSh}_S \to \mathbf{PSh}_S$, constructed from products, coproducts, and context extension $\delta_\Gamma P \triangleq P(\Gamma + -)$.
2. The category of presheaves is monoidal, with variables $\mathcal{V}$ as the unit and the substitution tensor $\otimes$ as the product. Σ is pointed-strong with respect to this monoidal structure.

3. The category of presheaves is cartesian closed, and exponentiation by representables $\yoneda\Gamma \supset P$ is isomorphic to context extension $\delta_\Gamma P$. Σ is strong with respect to the cartesian structure.
4. A Σ-monoid $\mathcal{M}$ is a monoid $\mathcal{V} \to \mathcal{M} \leftarrow \mathcal{M} \otimes \mathcal{P}$ with Σ-algebra structure $\Sigma\mathcal{M} \to \mathcal{M}$: a presheaf with substitution structure, variables, and term constructors.
5. Given a presheaf of metavariables $\mathcal{P}$, the initial $(\Sigma{+}\mathcal{V}{+}\mathcal{P}\otimes)$-algebra $\mathbf{T}\mathcal{P}$ encodes the syntax of the language with terms, variables, and parametrised metavariables. It comes with recursion and induction principles arising from initiality $\mathbf{T}\mathcal{P} \to \mathcal{A}$, monoidal-parametrised initiality $\mathbf{T}\mathcal{P} \otimes \mathcal{Q} \to \mathcal{A}$ and cartesian-parametrised initiality $\mathbf{T}\mathcal{P} * Q \to \mathcal{A}$.
6. $\mathbf{T}\mathcal{P}$ is the free Σ-monoid on $\mathcal{P}$: given a Σ-monoid $\mathcal{M}$ and interpretation of metavariables $\varphi\colon \mathcal{P} \to \mathcal{M}$, there is a unique Σ-monoid homomorphism $\mathbf{T}\mathcal{P} \to \mathcal{M}$ factoring φ.
7. Freeness makes $\mathbf{T}\colon \mathbf{PSh}_S \to \mathbf{PSh}_S$ into a monad, and induces an external meta-interpretation operation $\mathbf{PSh}_S(\mathcal{P}, \mathcal{M}) \to \mathbf{PSh}_S(\mathbf{T}\mathcal{P}, \mathcal{M})$ in any Σ-monoid $\mathcal{M}$.
8. Using cartesian-parametrised initiality, we show that $\mathbf{T}$ is a strong monad, and meta-interpretation thus internalises to a natural transformation $\langle \mathcal{P}, \mathcal{M}\rangle \Longrightarrow \langle \mathbf{T}\mathcal{P}, \mathcal{M}\rangle$ between presheaf exponentials.
9. Second-order axioms in the syntax induce a sound and complete equational logic with respect to Σ-monoids that satisfy the axioms.

The familial model introduced in this thesis will address each of these points, adapting the well-understood techniques of the presheaf approach to an implementation-friendly theory.

2.3 Related work

The study of abstract syntax and variable binding is rich and extensive, yet surprisingly young. Although the foundational notions of variables, substitution, and binding permeate all of mathematics, it was not until Church's (1936) λ-calculus that a formal system was explicitly devised to study abstraction and variable binding. Efforts to place these concepts on rigorous mathematical foundations only gained traction in the 1990s, largely due to the challenges of formalising capture-avoiding substitution – especially for computer-based implementations. Intriguingly, many core themes of the field trace back to its earliest publications: Church (1932, Section 4) invokes an intuitive notion of substitution (presumably capture-avoiding) without defining it; Barendregt (1985, Moral 2.1.14) adopts α-equivalence and the variable convention as blanket remedies for renaming, capture, and shadowing; and de Bruijn (1972) introduces his de Bruijn indexing scheme precisely to handle the practical difficulties of renaming and substitution in implementations.

In this section, we provide an overview of major approaches to abstract syntax formalisation. For further surveys that complement this account, see Aydemir et al. (2005), Kmett (2015), Cockx (2021), Popescu (2023), and Lamiaux and Ahrens (2025).

2.3.1 Named approaches

Named-variable formalisation seeks to mirror traditional mathematical practice, where binding is represented by matching variable names. While appealing in its naturalness, this ap-

proach introduces several technical challenges for formalisation, such as dealing with string equality, shadowing, and explicit α-conversion. Nevertheless, a major development came with the axiomatisation of permutation models in Fraenkel–Mostowski set theory, applied to syntax by Gabbay and Pitts (1999, 2002) and later developed into *nominal logic* by Pitts (2003, 2013). Gabbay and Pitts observed that essential notions such as freshness, capture, and α-equivalence could be defined purely in terms of atom swapping, with *finite support* – the property that each term contains only finitely many atoms – serving as a central concept. These foundations enable the introduction of a freshness quantifier $\mathit{N}a \in \mathbb{A}.\, \varphi$ and name abstraction $[\mathbb{A}]X$.

Nominal techniques have since found broad applicability across logic and computer science, including equational theories and unification (Urban et al., 2003; Clouston and Pitts, 2007; Gabbay and Mathijssen, 2009; Clouston, 2010), structural induction and recursion (Pitts, 2006, 2011), rewriting systems (Fernández and Rubio, 2012; Domínguez and Fernández, 2019), game semantics (Abramsky et al., 2004; Murawski and Tzevelekos, 2016), probabilistic programming (Sabok et al., 2021), and even cubical type theory (Pitts, 2014, 2015). Nominal syntax has been implemented in practical systems such as FreshML (Pitts and Gabbay, 2000; Shinwell et al., 2003) and FreshOCaml (Shinwell, 2006), enabling the definition of datatypes with binding. It also underpins the Nominal Isabelle framework (Urban, 2008; Huffman and Urban, 2010; Urban and Kaliszyk, 2011), which has been successfully applied in numerous formalisation efforts (Urban and Norrish, 2005; Bengtson and Parrow, 2007; Tobin-Hochstadt and Felleisen, 2008; Urban et al., 2011; Paulson, 2015).

Adapting nominal methods to dependently typed settings is nontrivial, as they often depend on classical axioms and set-theoretic reasoning. Constructive implementations have been explored via several routes, including an experimental Rocq axiomatisation (Aydemir et al., 2007), an initial setoid-based Agda development (Choudhury, 2015), a Rocq implementation using type classes and rewriting to manage equivalence without descending into "setoid hell" (Paranhos, 2022; Paranhos and Ventura, 2022), and an ongoing Agda project modelling permutations as bijections (Pagano and Solsona, 2023).

Other significant named approaches include the following.

- The *locally named* approach of Randy Pollack et al. (2012) separates the syntactic constructs for free (global) and bound (local) variables, making use of nominal predicates in the meta-syntactic operations.
- A line of work continues on the Agda formalisation of named binders in the syntax of the STLC using ideas from nominal techniques, Stoughton's (1988) parallel substitution, and explicit α-conversion (Tasistro et al., 2015; Copello et al., 2016, 2017, 2018[b]; Urciuoli et al., 2020; Copello et al., 2021), extended to generic signatures by Copello (2017) and Copello et al. (2018[a]).
- Gheri's (2019) general theory of syntax with bindings defines the α-equivalence class of syntax-generic quasi-terms using nominal swapping and substitution, using bounded natural functors (Blanchette et al., 2019) to construct complex binding patterns.
- Renaming-enriched sets or *rensets* (Popescu, 2023) associate every term in the set with a renaming operator that generalises the swapping operation on nominal sets, simplifying the freshness predicate and recursion principles;

- Wan and Cao (2024) develops Rocq formalisation of a syntax-driven, decidable α-compatibility relation that traverses terms in parallel and compares variable names in identical positions directly, taking variable shadowing into account.
- Anand and Rahli (2014) presents a Rocq axiomatisation of context-free grammars annotated with variable-binding information (CFGV), along with a term language and named syntactic metatheory parametrised by such a CFGV.

For a broad comparison of named and nameless frameworks, see Berghofer and Urban (2007); recursion principles for nominal representations are also discussed in Popescu (2024). Overall, named approaches are theoretically rich and flexible but present obstacles in constructive and dependently-typed contexts. Formalisations in Agda remain in early stages or are hindered by the need for setoid-based reasoning (Choudhury, 2015; Pagano and Solsona, 2023). Even more complete efforts, such as Copello et al.'s (2021), involve low-level name manipulations that can be difficult to scale and use.

2.3.2 Nameless approaches

Nameless representations encode binding without using variable names, enabling simpler mechanisation at the expense of readability. The main example is de Bruijn indices (de Bruijn, 1972), where variables are represented as natural numbers relative to their binding depth.

§ Numeric de Bruijn indices

De Bruijn indices are compact and sidestep α-conversion entirely. They are widely used in compilers and interpreters and are straightforward to use in a datatype declaration:

```
data Tm : Set where
  var : ℕ → Tm
  lam : Tm → Tm
  app : Tm → Tm → Tm
```

However, they lack static guarantees: indices may refer outside the scope of binders, and such errors are not syntactically evident. Terms are also difficult to read and debug, so such errors are not as readily evident as in named approaches. For instance, the expression $\lambda f.\, f(\lambda x g.\, f(g, x))$ translates to $\lambda 0(\lambda\lambda 2(01))$, where the variable f is encoded as both 0 and 2, and the de Bruijn index 0 encodes both f and g.

Formalising substitution with de Bruijn indices is error-prone, requiring delicate shifting operations and strengthened induction hypotheses. This complexity has been widely noted in the literature (Shankar, 1988; Altenkirch, 1993; Huet, 1994; Berghofer and Urban, 2007; Matache, 2017), with some authors reflecting on the tediousness of such work:

> *"It is annoying to spend so much time on uninteresting details."* – (Rasmussen, 1995)

Although criticised by the PoplMark challenge (Aydemir et al., 2005), several authors have defended the practicality of de Bruijn indices (Vouillon, 2011; Berghofer, 2012), showing their viability for formalisation despite the arithmetic overhead and fragility of implementation.

Recent theoretical work attempts to formalise de Bruijn indices more abstractly. Hirschowitz et al. (2022) define *de Bruijn monads* – sets A equipped with variables $\mathbb{N} \to A$, substitution $A \times A^{\mathbb{N}} \to A$, and syntax algebra $\Sigma A \to A$ – in a manner similar to the presheaf model. De Bruijn monads are simplified ("extrinsic") representations of abstract clones (Taylor, 1993; Arkor and McDermott, 2021), though the authors do not make this connection clear.

Another significant development is the calculus of *explicit substitutions*. Originally introduced in the $\lambda\sigma$-calculus (Abadi et al., 1991; Rose, 1996), substitutions become part of the syntax with terms for composition and de Bruijn shifting operations, allowing controlled application of simultaneous substitutions. This approach improves implementation efficiency but adds complexity to the metatheory, as substitution is no longer a meta-operation and must be handled in all definitions and proofs.

A practical application of this idea is the Autosubst framework for Rocq. Its first version (Schäfer et al., 2015) generates substitution operations and lemmas from annotated syntax. Autosubst 2 (Kaiser et al., 2017; Stark, 2020) compiles an expressive higher-order syntax representation to Rocq code with well-scoped terms, modularity, first-class renamings and traversal (Kaiser et al., 2018; Forster and Stark, 2020). While powerful, these systems still rely on numeric indices, which limits the static guarantees that dependently typed systems can provide.

Other frameworks that use de Bruijn indices include:

- Pottier's (2014) DBLib uses Rocq's type classes to generate lifting, substitution, and other definitions and lemmas from a user-supplied datatype for the syntax, a traversal operation, and six easily dischargeable lemmas.
- Polonowski (2013) introduces a grammar called *de Bruijn with Explicit Binding (DBEB)* to express syntaxes with binding and generic infrastructure operations and lemmas thereon, and the tool DBGen for augmenting annotated Rocq declarations with a variety of metatheoretic definitions up to and including the substitution lemma.
- Keuchel et al. (2016) present Knot, an expressive specification language for syntaxes with binders and accompanying specification-generic syntactic metatheory (shifting, substitution, well-scopedness, interaction lemmas), and Needle, a code generation tool turning Knot specifications into de Bruijn-encoded Rocq terms and their metatheoretic boilerplate.

All in all, while numeric de Bruijn indices remain a popular and practical choice – particularly in the Rocq community – and have been successfully deployed in large-scale formalisation projects, their use can feel somewhat uneasy. When working in a dependently typed setting, where one typically enjoys rich static guarantees, the reliance on natural numbers to represent something as fundamental as variable binding can feel precarious – much like reaching into unsafe code in a language like C. The mismatch between the rigid structure of syntax, and the flexibility (or flimsiness) of natural numbers motivates the investigation of *well-scoped syntax*.

§ Endofunctors and monadic substitution

The integration of monads into computer science originated with Moggi (1991) and his computational λ-calculus, later popularised by Wadler (1990, 1992) within the functional programming community. A key insight came from Bellegarde and Hook (1994), who proposed defining syntax as a monad $\mathsf{Tm} : \mathsf{Set} \to \mathsf{Set}$, parametrised uniformly by a variable set:

```
data Tm : Set → Set where
  var :     X → Tm X
  lam : Tm X → Tm X
  app : Tm X → TmX → Tm X
```

This encoding enables a generic definition of substitution for de Bruijn-indexed terms, Tm, ℕ, via a polymorphic function:

```
map-with-policy : ((X → Y) → (X → Y)) → (X → Y) → Tm X → Tm Y
```

This operation simultaneously handles de Bruijn shifting and substitution, with a final monadic join Tm(Tm X) → Tm X collapsing nested layers. The idea lay mostly dormant until Bird and Paterson (1999[a]) applied the notion of *nested datatypes* (Bird and Meertens, 1998) to syntactic terms with binding. Their insight was to bound de Bruijn indices at the type level, encoding scope directly into types by using the Maybe monad:

```
data Tm : Set → Set where
  var :     X → Tm X
  lam : Tm (Maybe X) → Tm X
  app : Tm X → Tm X → Tm X
```

This deceptively small change has a profound effect: bound variables are now characterised precisely, and scope is enforced statically. For instance, when $X = \bot$ (the empty type), any term t : Tm $\bot$ must be closed, since there are no free variables to reference. Under a binder, the scope is extended by wrapping X in Maybe; the bound variable is accessible as var none, and nested scopes are encoded via nested applications of some. This design ensures that variables cannot escape their scope – an invariant enforced by the type system rather than manually checked via shifting and renaming. Compared to raw numeric indices, this provides a vastly improved user and proof experience.

Bird and Paterson (1999[a]) went on to define a generalised recursion scheme (Bird and Paterson, 1999[b]) and used a distributive law Maybe(Tm X) → Tm(Maybe X) to derive monadic multiplication Tm(Tm X) → Tm X, from which single-variable substitution Tm X → Tm(Maybe X) → Tm X follows naturally. Building on this, Altenkirch and Reus (1999) provided categorical foundations for this approach, offering two constructions of the monadic Kleisli structure for Tm. One is a mutually recursive definition of the operations

```
bind : (X → Tm Y) → (Tm X → Tm Y)
lift : (X → Tm Y) → (Maybe X → Tm (Maybe Y))
```

which involves intricate termination and monad law proofs. The alternative construction is more modular: it leverages the functoriality of Maybe to prove the same property for Tm, defines lift via the functorial lifting Tm some : Tm X → Tm (Maybe X), and then constructs bind in terms of lift, avoiding mutual recursion.

Already, clear connections to the presheaf model emerge – indeed, as the authors remark, "Our work seems to be closely related to recent work by Fiore et al. (1999)." Altenkirch and

Reus (1999) also extend their approach to simply typed syntax, generalising from Kleisli triples to Kleisli structures on indexed sets. This anticipates the development of *relative monads* (Altenkirch et al., 2010), and underpins the intrinsically typed, intrinsically scoped syntax for λ-terms used in many modern developments.

Two further strands of research have continued to explore and refine monadic abstract syntax, both still active and evolving. For detailed comparisons between these methods, the nested monad approach, and the presheaf model discussed next, see the surveys by Zsido (2010) and Lamiaux and Ahrens (2025).

Modules over monads The *modules-over-monads* framework models syntax with binding by capturing both syntactic structure and substitution in a unified categorical setting. The core challenge lies in reconciling the algebraic structure of syntax (from a signature) with the substitution structure provided by a monad.

A syntactic signature is presented by a functor $\Sigma\colon \mathbf{Set}^{\mathbf{Set}} \to \mathbf{Set}^{\mathbf{Set}}$, combining the domains of syntactic constructors. These constructors can be retrieved from a Σ-algebra structure $a\colon \Sigma(\mathsf{Tm}) \to \mathsf{Tm}$. For example, in the STLC, the constructor algebra is the copairing:

$$[\mathsf{lam}, \mathsf{app}]\colon \delta(\mathsf{Tm}) + (\mathsf{Tm} \times \mathsf{Tm}) \to \mathsf{Tm},$$

where $\delta(X) \triangleq X \circ \mathsf{Maybe}$. However, as observed by Hirschowitz and Maggesi (2007), the interaction between:

- the substitution unit $\eta\colon X \to \mathsf{Tm}(X)$,
- substitution itself $\mu\colon \mathsf{Tm}(\mathsf{Tm}(X)) \to \mathsf{Tm}(X)$, and
- the syntax constructor map $a\colon \Sigma(\mathsf{Tm}) \to \mathsf{Tm}$

does not follow standard homomorphism patterns. Specifically:

- μ is not a Σ-algebra homomorphism, since $\mathsf{Tm} \circ \mathsf{Tm}$ is not a Σ-algebra.
- a is not a monad morphism, as $\Sigma(\mathsf{Tm})$ is not a monad.

These failures are illustrated by the non-commuting diagrams:

$$\begin{array}{ccc} \Sigma(\mathsf{Tm} \circ \mathsf{Tm}) & \xrightarrow{\Sigma\mu} & \Sigma\mathsf{Tm} \\ {\scriptstyle ?}\downarrow & & \downarrow{\scriptstyle a} \\ \mathsf{Tm} \circ \mathsf{Tm} & \xrightarrow[\mu]{} & \mathsf{Tm} \end{array} \qquad \begin{array}{ccc} \Sigma\mathsf{Tm} \circ \Sigma\mathsf{Tm} & \xrightarrow{a \circ a} & \mathsf{Tm} \\ {\scriptstyle ?}\downarrow & & \downarrow{\scriptstyle \mu} \\ \Sigma\mathsf{Tm} & \xrightarrow[a]{} & \mathsf{Tm} \end{array}$$

To address this, Hirschowitz and Maggesi (2007) and collaborators (Ahrens and Zsido, 2011; Hirschowitz and Maggesi, 2012; Ahrens, 2016; Hirschowitz et al., 2020; Ahrens et al., 2021; Lamiaux and Ahrens, 2025) proposed using *modules over a monad* $T\colon \mathbf{Set}^{\mathbf{Set}} \to \mathbf{Set}^{\mathbf{Set}}$, i.e. functors $S\colon \mathbf{Set}^{\mathbf{Set}} \to \mathbf{Set}^{\mathbf{Set}}$ equipped with a natural transformation $\alpha\colon S \circ T \Longrightarrow S$, satisfying the coherence conditions:

$$\begin{array}{ccc} S\mathrm{Id} & \xrightarrow{S\eta} & ST \\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle \alpha} \\ & & S \end{array} \qquad \begin{array}{ccc} ST & \xrightarrow{S\mu} & ST \\ {\scriptstyle \alpha T}\downarrow & & \downarrow{\scriptstyle \alpha} \\ ST & \xrightarrow[\alpha]{} & S \end{array}$$

A *signature* in this setting assigns to each monad T a T-module $(\Sigma T, \Sigma T \circ T \to \Sigma T)$. A *model* consists of a monad T together with a module morphism $\Sigma T \to T$. For the STLC, the signature maps T to $\Sigma_\Lambda T = \delta T + T \times T$, with module structure defined by:

$$(\delta T + T \times T) \circ T \to \delta(T \circ T) + (T \times T) \circ T \to \delta(TT) + TT \times TT \to \delta T + T \times T.$$

A model is then a monad T with a module morphism $a \colon \Sigma_\Lambda T \to T$ satisfying:

$$\begin{array}{ccccc} \Sigma_\Lambda T \circ T & \xrightarrow{\sigma} & \Sigma_\Lambda(T \circ T) & \xrightarrow{\Sigma_\Lambda \mu} & \Sigma_\Lambda T \\ {\scriptstyle a \circ T}\downarrow & & & & \downarrow{\scriptstyle a} \\ T \circ T & & \xrightarrow[\mu]{} & & T \end{array}$$

The initial model Tm gives rise to a unique syntax- and substitution-preserving model homomorphism $\mathsf{sem} \colon \mathsf{Tm} \to T$ for any model (T, a):

$$\begin{array}{ccc} \Sigma(\mathsf{Tm}) & \xrightarrow{\Sigma \mathsf{sem}} & \Sigma(T) \\ {\scriptstyle \mathsf{alg}}\downarrow & & \downarrow{\scriptstyle a} \\ \mathsf{Tm} & \xrightarrow[\mathsf{sem}]{} & T \end{array}$$

This framework is well-developed and has been extended to handle typing and operational semantics (Zsido, 2010; Ahrens, 2012, 2015, 2016; Ahrens et al., 2019; Hirschowitz et al., 2020; Ahrens et al., 2021; Lamiaux and Ahrens, 2025). However, it also exhibits practical drawbacks that can limit the scalability of the formalism:

- Flat hierarchy: All components – signatures, contexts, terms, variables – are encoded sets or endofunctors on **Set** or $\mathbf{Set}^{\mathbf{Set}}$, making it hard to know what level of the metatheory to operate on and how to distinguish between application and composition:

$$(\delta(S) \circ T)(TX) \quad \text{vs.} \quad (\delta(ST) \circ T)(X) \quad \text{vs.} \quad \delta(S \circ TT)(X)$$

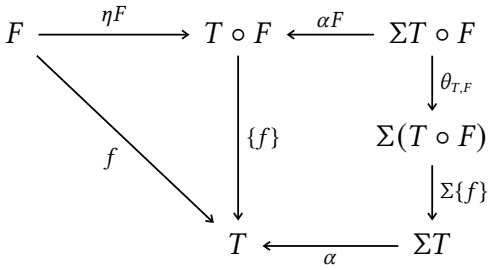
- Excessive flexibility: since the syntax can be instantiated over any set, bizarre terms like

  $\mathsf{lam}\,(\mathsf{app}\,(\mathsf{var}\,\mathsf{none})(\mathsf{var}\,(\mathsf{some}\,[\mathsf{var}\,[], \mathsf{lam}\,(\mathsf{var}\,(\mathsf{some}\,(-0.381i))]))))) \in \mathsf{Tm}(\mathsf{List}(\mathsf{Tm}(\mathbb{C})))$

  are type-correct but meaningless in practice. This level of generality can introduce errors that are difficult to catch with a type system.

Heterogeneous substitution systems An alternative approach, pioneered by Matthes and Uustalu (2004), focuses on substitution for non-wellfounded grammars using what are called *heterogeneous substitution* systems (HSS). This generalises iteration principles such as Bird and Paterson's (1999) folds to a categorical setting.

Given a pointed strong endofunctor $(\Sigma, \varphi \colon \mathrm{Id} \Longrightarrow \Sigma, \theta_{F,G} \colon \Sigma(F) \circ G \to \Sigma(F \circ G))$ on $\mathbf{Set}^{\mathbf{Set}}$, an HSS assigns to every pointed morphism $f \colon (F, \varphi) \to (T, \eta)$ into a Σ-algebra $(T \in \mathbf{Set}^{\mathbf{Set}}, \alpha \colon \Sigma T \to T)$ a unique extension $\{f\} \colon T \circ F \to T$, satisfying:



The module-over-monads approach naturally yields an HSS: the module action $\Sigma T \circ T \longrightarrow \Sigma T$ is generally composed of a strength $\Sigma T \circ T \longrightarrow \Sigma(T \circ T)$ followed by a monad multiplication. The coherence pentagon of algebraic and monadic structure is an instance of the axiom above for $\{\mathrm{id}\} : T \circ T \longrightarrow T$. As Matthes and Uustalu show, the setting of HHSs allow for modelling more complex syntactic constructs, such as explicit flattening: the endofunctor $\Sigma T \triangleq T \circ T$ with strength $\Sigma T \circ F = TTF \xrightarrow{T\varphi TF} TFTF$ defines a heterogeneous substitution system that is coherent with an internal monadic flattening operation $\Sigma T = TT \longrightarrow T$.

Ahrens and Matthes (2018) extended the theory by defining homomorphisms of heterogeneous substitution systems and relating initial algebras to initial substitution systems. They highlight important connections between the generalised iteration of Matthes and Uustalu (2004), the even more generalised folds of Bird and Paterson (1999[b]), and the Mendler-style induction of Mendler (1991), which handles recursive syntaxes by enforcing termination via typing rather than structural constraints. Abel et al. (2005) subsumed generalised folds by extending Mendler-style iteration to higher-order and nested datatypes, applying it to the nested structure of lambda-terms. Matthes (2011) further explored syntactic operations and laws on monadic syntax using induction principles derived from this generalised Mendler framework.

The most recent development is by Matthes et al. (2023), who abstract heterogeneous substitution systems to monoidal and modular categories equipped with pointed strengths. This general setting subsumes both heterogeneous substitution systems and the substitution monoids of Fiore et al. (1999), covering simple types and non-wellfounded syntax. However, it does not address the challenges posed by quotienting in the presheaf model.

2.3.3 Presheaves and monoidal substitution

The presheaf and family-based approach aligns naturally with the intrinsic formalisation of syntax, representing contexts as structured lists of type- and scope-safe de Bruijn indices. This avoids the two main drawbacks of the monadic approach: variables are inductively defined and listed (rather than generated via the Maybe monad in the broad category **Set**), terms live in sets functorially indexed by contexts (not as endofunctors on **Set**), and signatures are encoded as endofunctors on presheaves (not endofunctors on endofunctors on **Set**).

Fiore et al. (1999) were the first to lay out a rigorous algebraic theory of abstract syntax. A central insight was to treat terms and their contexts of free variables inseparably, with variable renaming a first-class syntactic operation. Substitution emerges via a monoidal structure that models simultaneous substitution: monoids in this structure are presheaves equipped with variable embeddings and substitution operations that respect renaming both externally (renaming commutes with substitution) and internally (renaming fuses with substitution). While

monad multiplication plays a similar role in the monadic setting, the simultaneous substitution rule is not spelled out explicitly: the terms to be substituted are embedded directly under their associated variable constructor, so the internal renaming coherence is not needed. In the presheaf setting, syntax models are substitution monoids with algebraic structure, echoing the modules-over-monads and HSS frameworks. The initiality theorem shows that the initial model of a syntax naturally carries this substitution structure, supporting the intuition that substitution and its properties follow "freely" from the syntax itself.

This foundational work sparked several important lines of research, summarised below.

Linear syntax Tanaka (2000) extended the theory to linear syntax, prompting broader generalisations to cover a wide array of systems (Tanaka, 2005; Tanaka and Power, 2006; Power, 2007; Power and Tanaka, 2008). These works developed pseudo-distributive laws that abstractly construct the substitution monoidal structure when the base category's structure is known, covering linear syntax, typing, substructural systems, and nominal syntax. However, the high level of abstraction made these developments less suited for mechanisation. While Fiore et al. (1999) also introduced single-variable substitution algebras, Tanaka (2000) claimed they could not generalise to linear settings – a claim later refuted by Fiore and Ranchod (2024), who used symmetric endofunctors and distributive laws to do precisely that.

Second-order syntax and algebraic theories The introduction of Σ-monoids – syntax models with substitution – led Hamana (2004) to study free constructions turning presheaves into Σ-monoids. This resulted in a general calculus of parametrised metavariables (Sato et al., 2003), supporting binding-aware opaque terms and metasubstitution operations. Fiore (2008) developed an enriched axiomatisation of metasubstitution alongside a generalised theory of strengths over modular categories, which underpins our work. Parametrised metavariables enable second-order equational presentations encompassing operations like β-reduction, formalised in a corresponding equational logic theory (Fiore and Hur, 2007, 2008, 2009, 2010; Hur, 2010). Parallel work defined second-order algebraic theories à la Lawvere (1963), extending to logic, universal algebra, and categorical algebra (Fiore and Mahmoud, 2010; Mahmoud, 2011; Fiore and Mahmoud, 2014; Fiore, 2017).

Typed syntax Typed extensions of abstract syntax began with Fiore (2002) and Miculan and Scagnetto (2003), motivated by normalisation-by-evaluation and higher-order abstract syntax. Zsido (2010) clarified the relationship between typed presheaf models and monadic syntax. Arkor and Fiore (2020) developed a theory for inductively defined simple types using polynomial functors. Polymorphic abstract syntax emerged from Hamana (2011) and was expanded by Fiore and Hamana (2013), incorporating metavariables and equational theories. A preliminary development of dependently-typed second-order syntax is outlined in Fiore (2008), though it lacks full generality and detail.

2.3.4 Higher-order abstract syntax

A conceptually elegant approach to representing syntax with binders in functional languages is higher-order abstract syntax (HOAS). In HOAS, binding in the object language is encoded

directly using the host language's abstraction mechanism. For example:

```
data Tm : Set where
  lam : (Tm → Tm) → Tm
  app : Tm → Tm → Tm
```

Here, variables are represented by variables of the metalanguage, and a lambda abstraction in the object language is encoded by a metalanguage function. As a result, operations like substitution can be delegated to the host language, avoiding the need to explicitly define them. An example HOAS term is:

$$\mathsf{app}\ (\mathsf{lam}\ (\lambda x.\ \mathsf{lam}\ (\lambda y.\ x)))\ (\mathsf{lam}(\lambda x.\ x)) \to_\beta\ (\lambda y.\ (\mathsf{lam}(\lambda x.\ x)))$$

This technique, introduced by Pfenning and Elliott (1988) but already suggested by Church (1940), is remarkably concise. However, this simplicity comes with significant drawbacks for the purposes of formal reasoning.

First, terms are no longer purely syntactic: some subterms are regular syntax (e.g. arguments of app), while others are functions (e.g. in lam). This hybrid nature complicates recursive definitions and structural induction – pattern-matching on binders becomes impossible, and defining operations like equality or term size is obstructed.

Second, HOAS permits the construction of exotic terms – values that do not correspond to any valid term in the object language. For example:

$$\mathsf{lam}\ (\lambda x.\ \mathsf{case}\ x\ \mathsf{of}\ (\mathsf{lam}\ t) \to \mathsf{app}\ t\ t \mid (\mathsf{app}\ f\ a) \to f)$$

Here, host-level pattern matching is used to inspect and distinguish between object-level constructs, which breaks the abstraction. Additionally, such a definition violates the positivity restriction required by many proof assistants: in the example above, the argument of the lam constructor features Tm in a negative position (to the left of an arrow). This is generally forbidden as it can introduce non-terminating behaviour:

```
loop : Tm → Tm → Tm
loop (lam  b) t  = b t
loop (app  f a) t = t
```

applied to t = lam ($\lambda x.$ loop x x) would result in the diverging reduction

$$\mathsf{loop\ t\ t} \to_\beta\ (\lambda x.\ \mathsf{loop}\ x\ x)\ \mathsf{t} \to_\beta\ \mathsf{loop\ t\ t}$$

This contradicts normalization and undermines the consistency of the type theory.

Weak HOAS To address these issues, Despeyroux and Hirschowitz (1994) proposed *weak HOAS*, where the function argument in lam is restricted to an abstract variable type V:

```
data Tm : Set where
  var  : V → Tm
  lam : (V → Tm) → Tm
  app : Tm → Tm → Tm
```

Variables are explicitly embedded into terms:

```
app (lam (λx. lam (λy. (var x)))) (lam (λx. (var x)))
```

Since V is abstract, we cannot pattern match on its values, thus avoiding exotic terms. Moreover, Tm is now strictly positive, so the type is accepted by proof assistants. However, since binders are encoded as functions from V, we can no longer define recursive functions structurally on terms, and substitution must be implemented explicitly.

Parametric HOAS *Parametric HOAS* (Chlipala, 2008) refines weak HOAS by parametrising the term syntax over the variable type, as first proposed by Washburn and Weirich (2003):

```
data Tm (V : Set) : Set where
  var : V → Tm V
  lam : (V → Tm V) → Tm V
  app : Tm V → Tm V → Tm V

Term : Set
Term = (V : Set) → Tm V
```

This formulation retains abstraction while enabling operations by instantiating V with a concrete type. For example, to count the number of applications in a term:

```
count' : Tm ⊤ → ℕ
count' (var v)   = 0
count' (lam b)   = count' (b tt)
count' (app f a) = count' f + count' a

count : Term → ℕ
count t = count' (t ⊤)
```

Because V is abstract in Term, exotic terms are ruled out. But when needed, we can instantiate V a variety of operations, including substitution (Despeyroux et al., 1995).

Semantics and proof assistants The semantics of HOAS were first formally developed by Hofmann (1999), using presheaves over finite sets and substitutions to model higher-order encodings. This framework encompasses weak and parametric HOAS, and modal approaches for equipping HOAS with recursion and induction principles (Despeyroux et al., 1997).
Many proof systems adopt HOAS and its variants, offering trade-offs in terms of abstraction, automation, and expressiveness:

- Twelf (Pfenning and Schürmann, 1999) encodes syntax in the logical framework LF (Harper et al., 1993) using HOAS, though metatheory is expressed using relations.
- Delphin (Poswolsky and Schürmann, 2009) replaces relational proofs with functional ones and includes a coverage and termination checker.

- Beluga (Pientka and Dunfield, 2008) further incorporates contextual modal type theory (Nanevski et al., 2008) for explicitly reasoning about variable environments.
- Cocon (Pientka et al., 2021) is a dependent type theory that enables intentional and extensional HOAS definitions and admits a Hofmann-style categorical semantics (Pientka and Schöpp, 2020).
- Mœbius (Jang et al., 2022) enables metaprogramming and object-level manipulation of HOAS syntax via contextual types.
- Abella (Gacek, 2008) separates the specification and reasoning logic into a two-level system based on λ-tree syntax (Miller, 2000), a restricted HOAS with nominal abstraction (Tiu, 2009; Gacek et al., 2011).
- Hybrid (Felty and Momigliano, 2012) encodes a similar two-level calculus in Rocq.

2.3.5 Locally nameless representation

An alternative approach is the locally nameless representation, which combines the benefits of named and nameless approaches by representing bound variables with de Bruijn indices, and free variables with explicit names.

```
data Tm : Set where
  fvar : String → Tm
  bvar : ℕ → Tm
  lam  : Tm → Tm
  app  : Tm → Tm → Tm
```

This style avoids the complexities of name binding while retaining readability for free variables. Variable opening and closing operations, combined with predicates of closure and freshness enable the definition of capture-avoiding substitution without the headaches of de Bruijn arithmetic, but with increased metatheoretic boilerplate.

Locally nameless was first hinted at by de Bruijn (1972) and later refined by Huet (1989), McKinna and Robert Pollack (1993), and Gordon (1994).McBride and McKinna (2004) describe a LN syntax representation library in Haskell that formed the foundation of Epigram. Leroy (2007) gave a prominent example in his POPLMark challenge solution. A comprehensive tutorial is provided by Charguéraud (2012), introducing techniques like cofinite quantification (Aydemir et al., 2008) to simplify reasoning about binders.

Despite its flexibility, the approach involves significant metatheoretical overhead. For complex languages, the number of supporting lemmas and operations can scale quadratically with the number of binding constructs. Rossberg et al. (2014) report this as a major pain point in their formalisation efforts. To reduce this burden, tools like LNGen (Aydemir and Weirich, 2010) generate boilerplate Coq code from Ott specifications (Sewell et al., 2010). Although verbose, this has been effectively used in several large-scale projects (Greenberg et al., 2010; Busenius, 2011; Bosman et al., 2023).

2.3.6 Representation-generic

Given the rich history of research into syntax representations, it is natural to ask whether one can reason about syntax generically, without first committing to nominal, de Bruijn, higher-order, or other encodings. Several efforts aim to support signature- and representation-generic metatheory by abstracting over the details of variable binding:

- GMeta (Lee et al., 2012) uses datatype-generic programming to derive first-order infrastructure and lemmas from Rocq encodings annotated with isomorphisms into a generic universe. It supports a range of binding styles (nominal, de Bruijn, locally named/nameless), though reasoning requires special tactics and operates over encodings rather than inductive types.
- Keuchel and Jeuring (2012) define a generic translation between de Bruijn and PHOAS in Agda. Syntax is described in a universe of codes, while PHOAS terms use a generalised higher-order Church encoding. The translation allows switching between representations for convenience (higher-order for programming, de Bruijn for proving), though the work doesn't cover substitution or other metatheory.
- Tealeaves (Dunn et al., 2023[b]) axiomatise binder-aware traversals via decorated traversable monads (DTMs) – compositions of syntax monads, writer monads, and applicative functors (McBride and Paterson, 2008). Backends like de Bruijn or locally nameless are instantiated by supplying a traversal operation satisfying DTM laws. A categorical account is developed in Dunn et al. (2023[a]).

As the preceding survey shows, no representation avoids all trade-offs. Every approach has its own form of "fiddliness" – whether it's manual α-conversion, de Bruijn arithmetic, locally nameless bookkeeping, or parameter management in higher-order encodings. That most programming language research is still published without formalisation underscores both the inherent complexity of the problem and the convenience of the Barendregt variable convention on paper. This thesis does not claim a universal solution, but within its scope, offers a mathematically robust treatment grounded in a principled foundation.

PART I

MATHEMATICAL FOUNDATIONS

The familial model of abstract syntax offers a lightweight and accessible framework for formalising languages in proof assistants. However, it departs significantly from the presheaf model in ways that merit closer study, especially since the main feature of presheaves – a uniform renaming operation – is not present in the family setting.

The first half of the thesis develops the mathematical infrastructure needed to describe the familial model at a level of generality comparable to that of presheaves, but without referring directly to contexts, variables, or substitution. By aligning the generality of both models, we enable formal comparisons and justify the translation of definitions and results from the rigid setting of presheaves to the more flexible world of indexed families – while preserving the core properties essential for the model theory of syntax. This part focuses on algebraic and biclosed structure, while the skew-monoidal aspects are deferred to Part II.

Chapter 3 introduces liftings and their associated distributive laws, which are central to transporting constructions between families and presheaves. These tools ultimately justify approximating syntax in the category of presheaves by initial algebras in the category of families. Chapter 4 explores categories with strong monoidal closed structure and establishes an equivalence between algebras for strong monads and morphisms into clone monads – forming the foundation of the metasubstitution theory for families.

While the internal mechanics of the presheaf model can be intricate, our goal is to present the theory as abstractly and modularly as possible. One of our key contributions is to identify and isolate the essential components of the model, so they can be reassembled in new categorical settings. To support clarity and intuition, we accompany the abstract development with a running example based on unsorted abstract syntax, assuming familiarity with the high-level overview of the presheaf model presented in Section 2.2.

CHAPTER 3

Lifting of algebras

One of the central aims of this thesis is to show that the presheaf theory of abstract syntax – summarised in Section 2.2 – can be faithfully reconstructed in the simpler setting of indexed families of sets. This chapter focuses on initial syntactic models in both presheaves and families, culminating in a proof that term syntax calculated in the category of families can be lifted to the initial model in the category of presheaves. This result bridges a key conceptual and practical gap: it ensures in intrinsically-typed syntax – typically implemented as an inductive data type without native support for renaming – the law-abiding renaming operation can be computed by structural recursion.

To build up to the initial algebra lifting theorem in Section 3.3, we first recall foundational notions of distributive laws and liftings in Section 3.1, then examine their relationship to adjunctions between algebra categories in Section 3.2. Finally, Section 3.4 discusses how distributive laws can be constructed over a free monad.

3.1 Distributive laws and liftings

The term "lifting" has a variety of meanings in category theory, so we will fix it to refer to the process of transporting some structure (functors, distinguished objects, categorical structure) from one category to another, most often along a functor that preserves the lifted structure. For example, we will say that an initial object lifts from category $\mathcal{D}$ to category $\mathcal{C}$ along $U : \mathcal{C} \to \mathcal{D}$ if there is an initial object $\bot_{\mathcal{C}} \in \mathcal{C}$ (often calculated from $\bot_{\mathcal{D}} \in \mathcal{D}$) such that $U\bot_{\mathcal{C}} = \bot_{\mathcal{D}}$. As presented in this section, liftings to endofunctor algebras are closely related to the notion of distributive laws.

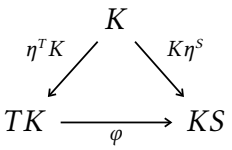
3.1.1 Distributive laws

Distributive laws were introduced by Beck (1969) to investigate compositions of monads and their liftings to algebras. Some results can be weakened to algebras for arbitrary endofunctors, motivating the following generalisation – the name was chosen in connection to liftings, and because most other alternatives (distributor, commutator, interchange law, etc.) were taken.

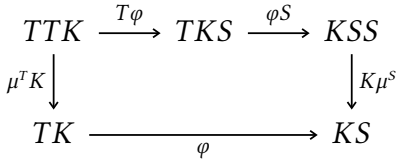
Definition 3.1.1 For two functors $F : \mathcal{A} \to \mathcal{C}$ and $G : \mathcal{B} \to \mathcal{D}$, an *elevator* from G to F is a pair of functors $K : \mathcal{A} \to \mathcal{B}$ and $L : \mathcal{C} \to \mathcal{D}$ and a natural transformation $\varphi : GK \Longrightarrow LF$. A *co-elevator* is the same pair of functors K, L with a natural transformation $\psi : LF \Longrightarrow GK$. ⌟

We will be particularly interested in cases where $F : \mathcal{C} \to \mathcal{C}$ and $G : \mathcal{D} \to \mathcal{D}$ are endofunctors, with co/elevators and liftings between them involving only one functor $K : \mathcal{C} \to \mathcal{D}$. When these endofunctors have additional – e.g. monad or comonad – structure, the elevator between them can be "upgraded" to a structure-preserving transformation.

Definition 3.1.2 A *distributive law* from a monad $S : \mathcal{C} \to \mathcal{C}$ to a monad $T : \mathcal{D} \to \mathcal{D}$ is an elevator $(K, \varphi : TK \Longrightarrow KS)$ between the underlying endofunctors, further satisfying

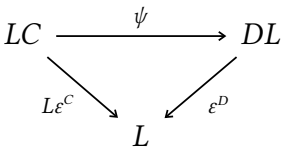


$(\varphi \lfloor \eta \rceil)$

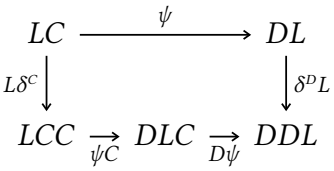


$(\varphi \lfloor \mu \rceil)$

Dually, a codistributive law from a comonad $C : \mathcal{C} \to \mathcal{C}$ to a comonad $D : \mathcal{D} \to \mathcal{D}$ is a co-elevator $(L, \psi : LC \Longrightarrow DL)$ between the underlying endofunctors, further satisfying



$(\psi \lfloor \varepsilon \rceil)$



$(\psi \lfloor \delta \rceil)$

⌟

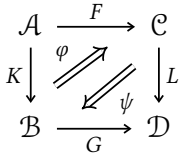
The theory of distributive laws comes with many elegant results (Beck, 1969; Tanaka, 2005), but we are most interested in their relationship to liftings.

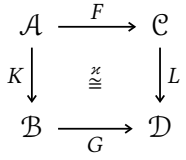
3.1.2 Liftings

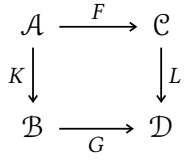
Definition 3.1.3 A functor $F : \mathcal{A} \to \mathcal{C}$ is a *(strong) lifting* of $G : \mathcal{B} \to \mathcal{D}$ along (K, L) if there is an elevator $(K : \mathcal{A} \to \mathcal{B}, L : \mathcal{C} \to \mathcal{D}, \varkappa)$ from F to G with $\varkappa$ an isomorphism $GK \cong LF$. ⌟

Definition 3.1.4 A functor $F : \mathcal{A} \to \mathcal{C}$ is a *strict lifting* of $G : \mathcal{B} \to \mathcal{D}$ along (K, L) simply when $GK = LF$ – that is, (K, L) is a morphism in the arrow category of **Cat** from F to G. ⌟

Following standard terminology, a weak lifting is then a non-invertible natural transformation – this is precisely an elevator as given in the section before. The weak, strong, and strict situations are shown below:







Notation. For a functor $L : F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$ and F-algebra (A, a), the carrier of the G-algebra $L(A, a)$ will be denoted A^L, and its structure map $a^L : G(A^L) \to A^L$. For an F-algebra homomorphism $f : (A, a) \to (B, b)$, $Lf : L(A, a) \to L(B, b) = (A^L, a^L) \to (B^L, b^L)$ is a homomorphism of G-algebras also denoted $f^L : A^L \to B^L$. ⌟

Definition 3.1.5 Given two endofunctors $F\colon \mathcal{C} \to \mathcal{C}$ and $G\colon \mathcal{D} \to \mathcal{D}$ and a functor $K\colon \mathcal{C} \to \mathcal{D}$, its *strict lifting to algebras* is a functor $\widehat{K}\colon F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$ such that $U^G\widehat{K} = KU^F$.

$$\begin{array}{ccc} F\text{-}\mathbf{alg}(\mathcal{C}) & \xrightarrow{\widehat{K}} & G\text{-}\mathbf{alg}(\mathcal{D}) \\ {\scriptstyle U^F}\downarrow & & \downarrow{\scriptstyle U^G} \\ \mathcal{C} & \xrightarrow[K]{} & \mathcal{D} \end{array}$$

Thus, for an F-algebra (A, a), $A^{\widehat{K}} = U^G(\widehat{K}(A, a)) = KA$ and $a^{\widehat{K}}\colon GKA \to KA$, which we will also denote $\hat{a}\colon GKA \to KA$ when K is clear from the context. ⌟

Strict liftings to algebras can be constructed from a lifting, up to isomorphism.

Proposition 3.1.1 *Let $K\colon \mathcal{C} \to \mathcal{D}$ be a functor and L a strong lifting to $F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$ via $\varkappa\colon U^GL \cong KU^F$. Then, there exists a strict lifting $\widehat{K}$ of K to $F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$ along the forgetful functors such that $L \cong \widehat{K}$.*

$$\begin{array}{ccc} F\text{-}\mathbf{alg}(\mathcal{C}) & \xrightarrow{L} & G\text{-}\mathbf{alg}(\mathcal{D}) \\ {\scriptstyle U^F}\downarrow & \overset{\varkappa}{\cong} & \downarrow{\scriptstyle U^G} \\ \mathcal{C} & \xrightarrow[K]{} & \mathcal{D} \end{array} \quad \Rightarrow \quad \begin{array}{ccc} F\text{-}\mathbf{alg}(\mathcal{C}) & \xrightarrow{\widehat{K}} & G\text{-}\mathbf{alg}(\mathcal{D}) \\ {\scriptstyle U^F}\downarrow & & \downarrow{\scriptstyle U^G} \\ \mathcal{C} & \xrightarrow[K]{} & \mathcal{D} \end{array}$$

Proof The natural isomorphism $\varkappa\colon U^GL \cong KU^F$ has components $A^L \cong KA$ at an F-algebra (A, a) We define the strict lifting $\widehat{K}\colon F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$ as mapping $(A, a\colon FA \to A)$ to the carrier KA with G-algebra structure

$$GKA \overset{\varkappa}{\cong} GA^L \xrightarrow{a^L} A^L \overset{\varkappa}{\cong} KA$$

and an F-algebra homomorphism $f\colon (A, a) \to (B, b)$ to $Kf : KA \to KB$ with the G-algebra homomorphism condition

$$\begin{array}{ccccccc} GKA & \xrightarrow{\cong} & GA^L & \xrightarrow{a^L} & A^L & \xrightarrow{\cong} & KA \\ {\scriptstyle GKf}\downarrow & & {\scriptstyle Gf^L}\downarrow & f^L\lfloor\vec{G}\rceil & \downarrow{\scriptstyle f^L} & & \downarrow{\scriptstyle Kf} \\ GKB & \xrightarrow[\cong]{} & GB^L & \xrightarrow[b^L]{} & B^L & \xrightarrow[\cong]{} & KB \end{array}$$

To show that this is a strict lifting, we need to prove that $U^G\widehat{K} = KU^F$: indeed, for all $(A, a) \in F\text{-}\mathbf{alg}(\mathcal{C})$, $U^G\widehat{K}(A, a) = KA = KU^F(A, a)$. Furthermore, the natural isomorphism $L \cong \widehat{K}$ holds because $\widehat{K}(A, a) = (KA, \varkappa^{-1} \circ a^L \circ \varkappa) \cong (A^L, a^L) = L(A, a)$. □

The definitions and properties above can also be dualised to categories of coalgebras.

3.1.3 Equivalence

The notions of elevators/distributive laws and liftings are closely connected – indeed the former can be used to induce the latter, and in the case of monad algebras, the result goes the other way around too.

Proposition 3.1.2 *An elevator $(K\colon \mathcal{C} \to \mathcal{D}, \varphi)$ from $F\colon \mathcal{C} \to \mathcal{C}$ to $G\colon \mathcal{D} \to \mathcal{D}$ strictly lifts to a functor $F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$.*

$$\begin{array}{ccc} F\text{-}\mathbf{alg}(\mathcal{C}) & \xrightarrow{K^\varphi} & G\text{-}\mathbf{alg}(\mathcal{D}) \\ {\scriptstyle U^F}\downarrow & & \downarrow{\scriptstyle U^G} \\ \mathcal{C} & \xrightarrow[K]{} & \mathcal{D} \end{array}$$

Proof Let $(K\colon \mathcal{C} \to \mathcal{D}, \varphi\colon GK \Longrightarrow KF)$ be an elevator. Define the lifting $K^\varphi\colon F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$ along the forgetful functors $U^F\colon F\text{-}\mathbf{alg}(\mathcal{C}) \to \mathcal{C}$ and $U^G\colon G\text{-}\mathbf{alg}(\mathcal{D}) \to \mathcal{D}$ as mapping an F-algebra $(A, a\colon FA \to A)$ to the carrier KA with G-algebra structure

$$GKA \xrightarrow{\varphi_A} KFA \xrightarrow{Ka} KA$$

and an F-algebra homomorphism $f\colon (A, a) \to (B, b)$ to the G-algebra homomorphism Kf

$$\begin{array}{ccccc} GKA & \xrightarrow{\varphi_A} & KFA & \xrightarrow{Ka} & KA \\ {\scriptstyle GKf}\downarrow & \boxed{\varphi} & {\scriptstyle KFf}\downarrow & f\lfloor\vec{F}\rceil & \downarrow{\scriptstyle Kf} \\ GKB & \xrightarrow[\varphi_B]{} & KFB & \xrightarrow[Kb]{} & KB \end{array}$$

□

The proposition can be strengthened to distributive laws between (co)monads.

Proposition 3.1.3 *A distributive law $K\colon \mathcal{C} \to \mathcal{D}$ from a monad $S\colon \mathcal{C} \to \mathcal{C}$ to a monad $T\colon \mathcal{D} \to \mathcal{D}$ strongly lifts to a functor $S\text{-}\mathbf{Alg}(\mathcal{C}) \to T\text{-}\mathbf{Alg}(\mathcal{D})$.*

Proof Let $(K, \varphi\colon TK \Longrightarrow KS)$ be an elevator between monads. Proposition 3.1.2 induces a lifting $K^\varphi\colon S\text{-}\mathbf{alg}(\mathcal{C}) \to T\text{-}\mathbf{alg}(\mathcal{D})$, and we can further show that the induced algebra structure $(A \in \mathcal{C}, SA \to A) \mapsto (KA \in \mathcal{D}, TKA \xrightarrow{\varphi_A} KSA \xrightarrow{Ka} KA)$ is compatible with the monad T:

$$\begin{array}{ccccc} KA & \xrightarrow{\eta^T_{KA}} & TKA & & \\ \| & \searrow & \boxed{\varphi\lfloor\eta\rceil} & \searrow & \downarrow{\scriptstyle \varphi_A} \\ KA & & \xleftarrow[Ka]{} & & KSA \end{array} \qquad \begin{array}{ccccc} TTKA & & \xrightarrow{\mu^T_{KA}} & & TKA \\ {\scriptstyle T\varphi_A}\downarrow & & \boxed{\varphi\lfloor\mu\rceil} & & \downarrow{\scriptstyle \varphi_A} \\ TKSA & \xrightarrow{\varphi_{SA}} & KSSA & \xrightarrow{K\mu^S_A} & KSA \\ {\scriptstyle TKa}\downarrow & \boxed{\varphi} & {\scriptstyle KSa}\downarrow & \vec{S} & \downarrow{\scriptstyle Ka} \\ TKA & \xrightarrow[\varphi_A]{} & KSA & \xrightarrow[Ka]{} & KA \end{array}$$

Thus, the lifting is a functor $K^\varphi\colon S\text{-}\mathbf{Alg}(\mathcal{C}) \to T\text{-}\mathbf{Alg}(\mathcal{D})$. □

A classic result of the theory of distributive laws is that not only do they induce liftings, they are *equivalent* to liftings. We show the inverse construction below; for the formal statement and proof of equivalence, see e.g. Beck (1969).

Proposition 3.1.4 *Given two monads $S\colon \mathcal{C} \to \mathcal{C}$ and $T\colon \mathcal{D} \to \mathcal{D}$, if a functor $K\colon \mathcal{C} \to \mathcal{D}$ has a strong lifting $\widehat{K}\colon S\text{-}\mathbf{Alg}(\mathcal{C}) \to T\text{-}\mathbf{Alg}(\mathcal{D})$ along the forgetful functors of S and T, there is a distributive law $\varphi\colon TK \Longrightarrow KS$ from T to S.*

PROOF Given an algebra $(A, a\colon SA \to A)$ for the monad S, the T-algebra $\widehat{K}(A,a)$ is of the form $(KA, \hat{a}\colon TKA \to KA)$, since the strong lifting condition $U^T(\widehat{K}(A,a)) = K(U^S(A,a)) = KA$ forces the carrier of $\widehat{K}(A,a)$ to be KA.

We first show that the collection of lifted structure maps $\widehat{\mu}_A\colon TKSA \to KSA$ for the free S-algebra $(SA, \mu_A\colon SSA \to SA)$ constitute a natural transformation $\widehat{\mu}\colon TKS \Longrightarrow KS$. For a $\mathcal{C}$-morphism $f\colon A \to B$, $Sf\colon SA \to SB$ is an S-algebra homomorphism between the free S-algebras $(SA, \mu_A) \to (SB, \mu_B)$. The lifting $\widehat{K}$ lifts this to a T-algebra homomorphism $\widehat{K}(Sf)\colon (KSA, \widehat{\mu}_A\colon TKSA \to KSA) \to (KSB, \widehat{\mu}_B\colon TKSB \to KSB)$. The naturality square for $\widehat{\mu}$ simplifies to the square below (since $U^T\widehat{K}Sf = KU^SSf = KSf$) which is nothing but the T-algebra homomorphism condition for $\widehat{K}Sf = KSf$:

$$\begin{array}{ccc} TKSA & \xrightarrow{\widehat{\mu}_A} & KSA \\ {\scriptstyle TKSf}\downarrow & & \downarrow{\scriptstyle KSf} \\ TKSB & \xrightarrow[\widehat{\mu}_B]{} & KSB \end{array}$$

With $\widehat{\mu}\colon TKS \Longrightarrow KS$ in hand, components $\widehat{\mu}_A\colon TKSA \to KSA$ of which are T-algebra structure maps of objects KSA, the distributive law $\varphi\colon TK \Longrightarrow KS$ is the composite

$$\varphi\colon TK \xrightarrow{TK\eta^S} TKS \xrightarrow{\widehat{\mu}} KS$$

which satisfies the distributive law axioms by naturality, monad and algebra laws. □

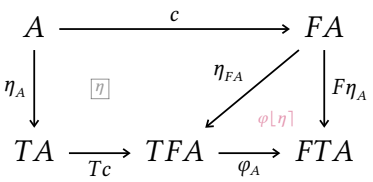
Given endofunctors F, G on the same category $\mathcal{C}$, an elevator $GF \Longrightarrow FG$ between them may induce liftings both to algebras and coalgebras. If one of the functors is a monad, we obtain a way of "swapping" liftings through a distributive law, which can be used to lift joint algebra-coalgebra structures on an object.

Proposition 3.1.5 *If $F\colon \mathcal{D} \to \mathcal{D}$ is an endofunctor and $T\colon \mathcal{D} \to \mathcal{D}$ is a monad, a lifting of F to T-algebras $\widehat{F}\colon T\text{-}\mathbf{Alg} \to T\text{-}\mathbf{Alg}$ gives rise to a monad $\widehat{T}\colon F\text{-}\mathbf{coalg} \to F\text{-}\mathbf{coalg}$.*

PROOF $\widehat{F}\colon T\text{-}\mathbf{Alg} \to T\text{-}\mathbf{Alg}$ induces an elevator $\varphi\colon TF \Longrightarrow FT$ by Proposition 3.1.4, which we may also consider as a *co-elevator* from F to F, inducing a lifting $\widehat{T}\colon F\text{-}\mathbf{coalg} \to F\text{-}\mathbf{coalg}$:

$$\widehat{T}(A, c\colon A \to FA) \mapsto (TA, TA \xrightarrow{Tc} TFA \xrightarrow{\varphi_A} FTA)$$

We show that $\widehat{T}$ is a monad on F-**coalg**. The unit of the monad $\widehat{T}\colon T\colon F\text{-}\mathbf{coalg} \to F\text{-}\mathbf{coalg}$ is a natural transformation $\eta_{(A,c)}\colon (A,c) \to (\widehat{T}(A,c) = (TA, \hat{c}\colon TA \to FTA))$ with components given by $\eta_A\colon A \to TA$, which is an F-coalgebra homomorphism:



The multiplication $\mu_{(A,c)} : \widehat{T}\widehat{T}(A,c) \to \widehat{T}(A,c)$ is an F-coalgebra homomorphism

$$(TTA, TTA \xrightarrow{TTc} TTFA \xrightarrow{T\varphi_A} TFTA \xrightarrow{\varphi_{TA}} FTTA) \to (TA, TA \xrightarrow{Tc} TFA \xrightarrow{\varphi_A} FTA)$$

defined, between the carriers, as $\mu_A : TTA \to TA$ and having the homomorphism condition

$$\begin{array}{ccccccc} TTA & \xrightarrow{TTc} & TTFA & \xrightarrow{T\varphi_A} & TFTA & \xrightarrow{\varphi_{TA}} & FTTA \\ \downarrow{\scriptstyle \mu_A} & \boxed{\mu} & \downarrow{\scriptstyle \mu_{FA}} & & \varphi\lfloor\mu\rceil & & \downarrow{\scriptstyle F\mu_A} \\ TA & \xrightarrow[Tc]{} & TFA & \xrightarrow[\varphi_A]{} & & & FTA \end{array}$$

□

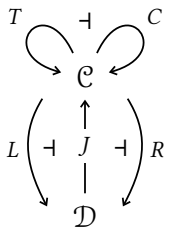
All results in this chapter can be dualiased to an equivalence between codistributive laws and liftings to coalgebras. Given $K : \mathcal{C} \to \mathcal{D}$,

- if K is a co-elevator from endofunctors $F : \mathcal{C} \to \mathcal{C}$ to $G : \mathcal{D} \to \mathcal{D}$ with $\psi : KF \Longrightarrow GK$, then K has a lifting $K^\psi : F\text{-}\mathbf{coalg}(\mathcal{C}) \to G\text{-}\mathbf{coalg}(\mathcal{D})$ to categories of algebras that maps $(A, a : A \to FA)$ to $(KA, KA \xrightarrow{Ka} KFA \xrightarrow{\psi_A} GKA)$;
- if K is a codistributive law between comonads $C : \mathcal{C} \to \mathcal{C}$ to $D : \mathcal{D} \to \mathcal{D}$, it has a lifting $K^\psi : C\text{-}\mathbf{Coalg}(\mathcal{C}) \to D\text{-}\mathbf{Coalg}(\mathcal{D})$ to categories of comonad-algebras;
- if K has a lifting $\widehat{K} : C\text{-}\mathbf{Coalg}(\mathcal{C}) \to D\text{-}\mathbf{Coalg}(\mathcal{D})$, the lifting of the free C-coalgebra $\widehat{K}(CA, \delta_A : CA \Longrightarrow CCA) = (KCA, \widehat{\delta}_A : KCA \to DKCA)$ induces a codistributive law between the comonads C and D: $\psi : KC \xrightarrow{\widehat{\delta}} DKC \xrightarrow{DK\varepsilon^C} DK$;
- if $G : \mathcal{C} \to \mathcal{C}$ is an endofunctor and $C : \mathcal{C} \to \mathcal{C}$ is a comonad, a lifting of G to coalgebras $\widehat{G} : C\text{-}\mathbf{Coalg} \to C\text{-}\mathbf{Coalg}$ induces a comonad $\widehat{C} : G\text{-}\mathbf{alg} \to G\text{-}\mathbf{alg}$.

The results established in this chapter apply whenever we have algebra and coalgebra structures on the same object – this will be the case in the proof of the initial-algebra lifting theorem in Section 3.3. Before that, we investigate the relationship between liftings and adjunctions.

3.2 Adjunctions

As will be seen in Section 7.2.1, our work features intricate adjoint situations in the presence of *adjoint triples*: functors $L, R : \mathcal{C} \to \mathcal{D}$ and $J : \mathcal{D} \to \mathcal{C}$ with $L \dashv J \dashv R$ and an induced monad-comonad pair $T \triangleq JL$ and $C \triangleq JR$.



In this section we prove properties concerning adjoint triples, liftings, and distributive laws.

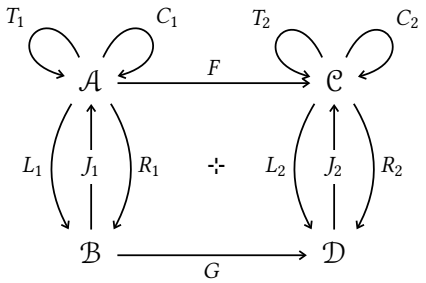
Proposition 3.2.1 *Assume two adjoint triples* $L_1 \dashv J_1 \dashv R_1 \colon \mathcal{B} \to \mathcal{A}$ *and* $L_2 \dashv J_2 \dashv R_2 \colon \mathcal{D} \to \mathcal{C}$ *with induced adjoint monad-comonad pairs* $T_1 \dashv C_1 \colon \mathcal{A} \to \mathcal{A}$ *and* $T_2 \dashv C_2 \colon \mathcal{C} \to \mathcal{C}$*, respectively. If* $F \colon \mathcal{A} \to \mathcal{C}$ *lifts to* $G \colon \mathcal{B} \to \mathcal{D}$ *along* (J_1, J_2)*, then we have co/elevators*

$$L_2F \Longrightarrow GL_1 \qquad GR_1 \Longrightarrow R_2F$$

and co/distributive laws

$$T_2F \Longrightarrow FT_1 \qquad FC_1 \Longrightarrow C_2F$$

Proof We have the following situation:



(†)

The lifting of F to G along (J_1, J_2) satisfies the natural isomorphism $\kappa \colon FJ_1 \cong J_2G$. The co/elevators are transposes of the following composites:

$$F \xrightarrow{F\eta} FJ_1L_1 \xrightarrow{\kappa L_1} J_2GL_1 \qquad J_2GR_1 \xrightarrow{\kappa^{-1}R_1} FJ_1R_1 \xrightarrow{F\varepsilon} F$$

and using these the co/distributive laws derive from these as:

$$T_2F = J_2L_2F \to J_2GL_1 \xrightarrow{\kappa^{-1}L_1} FJ_1L_1 = FT_1 \qquad FC_1 = FJ_1R_1 \xrightarrow{\kappa R_1} J_2GR_1 \to J_2R_2F = C_2F$$

Since the co/distributive laws and co/monad operations are all constructed from co/units, the compatibility laws reduce to zig-zag identities. □

The full generality of the above theorem is needed to cover the cases when F and G are functors of multiple arguments: for example, with a bifunctor $F \colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, an isomorphism $J(F(A,B)) \cong F(JA, JB)$ will induce a distributive law $T(F(A,B)) \to F(TA, TB)$. When $F \colon \mathcal{C} \to \mathcal{C}$ and $G \colon \mathcal{D} \to \mathcal{D}$ are endofunctors on an adjoint triple $L \dashv J \dashv R \colon \mathcal{C} \to \mathcal{D}$ in an adjoint situation (†) in Proposition 3.2.1, the elevators calculated give rise to liftings to co/algebras:

$$\begin{array}{ll} L \text{ to } F\text{-}\mathbf{coalg}(\mathcal{C}) \to G\text{-}\mathbf{coalg}(\mathcal{D}) & R \text{ to } F\text{-}\mathbf{alg}(\mathcal{C}) \quad \to G\text{-}\mathbf{alg}(\mathcal{D}) \\ F \text{ to } T\text{-}\mathbf{Alg}(\mathcal{C}) \quad \to T\text{-}\mathbf{Alg}(\mathcal{C}) & G \text{ to } C\text{-}\mathbf{Coalg}(\mathcal{C}) \to C\text{-}\mathbf{Coalg}(\mathcal{C}) \end{array}$$

The following corollary of Backhouse et al. (1995, Fusion Theorem) and Hermida and Jacobs (1995, Theorem 2.3.1) relates the first two of these liftings.

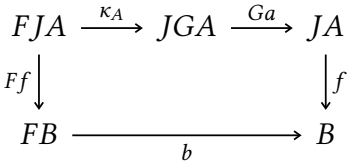
Proposition 3.2.2 *A lifting of* $F \colon \mathcal{C} \to \mathcal{C}$ *to* $G \colon \mathcal{D} \to \mathcal{D}$ *induces adjunctions between* F-$\mathbf{alg}(\mathcal{C})$ *and* G-$\mathbf{alg}(\mathcal{D})$*, as well as* F-$\mathbf{coalg}(\mathcal{C})$ *and* G-$\mathbf{coalg}(\mathcal{D})$*.*

Proof The two sides of the isomorphism $\kappa\colon FJ \cong JG : \varkappa$ are elevators that induce liftings $J^\kappa\colon G\text{-}\mathbf{alg}(\mathcal{D}) \to F\text{-}\mathbf{alg}(\mathcal{C})$ and $J^\varkappa\colon G\text{-}\mathbf{coalg}(\mathcal{D}) \to F\text{-}\mathbf{coalg}(\mathcal{C})$ (Proposition 3.1.2). We show that these have a right and left adjoint, respectively.

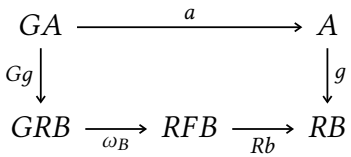
By Proposition 3.2.1, we have an elevator $\omega\colon GR \Longrightarrow RF$ which lifts $R\colon \mathcal{C} \to \mathcal{D}$ to $R^\omega\colon F\text{-}\mathbf{alg}(\mathcal{C}) \to G\text{-}\mathbf{alg}(\mathcal{D})$; we now show this to be the right adjoint of the lifted functor $J^\kappa\colon G\text{-}\mathbf{alg}(\mathcal{D}) \to F\text{-}\mathbf{alg}(\mathcal{C})$ by establishing the natural hom-set isomorphism

$$\frac{J^\kappa(A,a) \to (B,b) \quad \in F\text{-}\mathbf{alg}(\mathcal{C})}{(A,a) \to R^\omega(B,b) \quad \in G\text{-}\mathbf{alg}(\mathcal{D})}$$

An F-algebra homomorphism $f\colon J^\kappa(A,a) \to (B,b)$ is a morphism $JA \to B \in \mathcal{C}$ that satisfies the left diagram below. It must be mapped to a morphism $g\colon A \to RB \in \mathcal{D}$ that satisfies the diagram on the right.

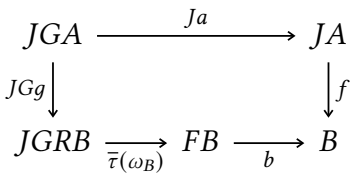


(†)

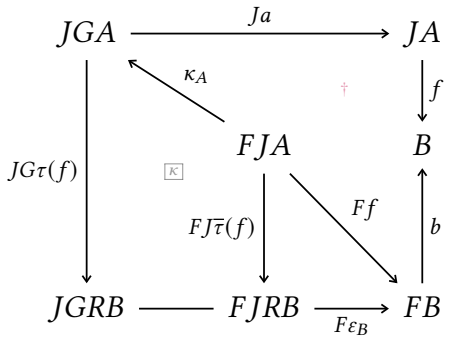


(‡)

Set g to be $\tau(f)\colon A \to RB$, the transpose of $f\colon JA \to B$ under $\tau\colon J \dashv R$. The G-algebra homomorphism condition (‡) can be transposed to



which, expanding $\overline{\tau}(\omega_B)$, can be proved as follows:



The reverse mapping $g\colon A \to RB \mapsto \overline{\tau}g\colon JA \to B$ satisfies (†) with similar reasoning, and the naturality of hom-set equivalence follows from the adjunction $\mathcal{C}(JA,B) \cong \mathcal{D}(A,RB)$.

The dual proof shows that the lifting of L to $F\text{-}\mathbf{coalg}(\mathcal{C}) \to G\text{-}\mathbf{coalg}(\mathcal{D})$ is the left adjoint of $J^\varkappa\colon G\text{-}\mathbf{coalg}(\mathcal{D}) \to F\text{-}\mathbf{coalg}(\mathcal{C})$, establishing the required adjunctions. □

The main application of this result is the lifting of co/limit preservation results from functors to liftings between algebras. In the next section we consider a particularly important colimit: the initial algebra of an endofunctor, which will eventually correspond to syntax.

3.3 Initial algebras

We now study the behaviour of initial algebras under liftings, establishing that under weak assumptions on the elevating functor, initial algebras lift to initial algebras. As expected, every result can be dualised to terminal coalgebras, but in the context of initial-algebra semantics, only one side of the story is of relevance. We therefore assume one side of the adjoint triple introduced above: the adjunction $J \dashv R \colon \mathcal{D} \to \mathcal{C}$ and the induced comonad C.

Notation. The initial algebra for $F \colon \mathcal{C} \to \mathcal{C}$ will be denoted $\mu F = (F, f \colon FF \to F)$. ⌟

Lemma 3.3.1 *If $F \colon \mathcal{C} \to \mathcal{C}$ lifts to $G \colon \mathcal{D} \to \mathcal{D}$ along J via $\kappa \colon FJ \cong JG$ and G has an initial algebra, so does F with $J^\kappa(\mu G) \cong \mu F$.*

Proof Proposition 3.2.2 shows that the adjunction $J \dashv R \colon \mathcal{D} \to \mathcal{C}$ induces an adjunction $J^\kappa \dashv R^\varkappa \colon G\text{-}\mathbf{alg}(\mathcal{D}) \to F\text{-}\mathbf{alg}(\mathcal{C})$ and, J^κ – being a left adjoint – preserves colimits, in particular initial objects in $G\text{-}\mathbf{alg}(\mathcal{D})$. Thus, $J^\kappa(G, g) = (JG, FJG \xrightarrow{\kappa_G} JGG \xrightarrow{Jg} JG) \in F\text{-}\mathbf{alg}(\mathcal{C})$ is an initial F-algebra, and furthermore, any initial F-algebra μF is isomorphic to $J^\kappa(G, g)$. □

The lemma above lets us "lower" initial algebras between adjoint categories, and it will be used in conjunction with the following initial algebra-lifting result.

Proposition 3.3.1 *Let $C \colon \mathcal{C} \to \mathcal{C}$ be a comonad and $F \colon \mathcal{C} \to \mathcal{C}$ an endofunctor with $\widehat{F}$ a strict lifting to $C\text{-}\mathbf{Coalg}(\mathcal{C})$. Then, an initial F-algebra $(F, f) \in F\text{-}\mathbf{alg}(\mathcal{C})$ lifts to an initial $\widehat{F}$-algebra $(\hat{F}, \hat{f}) \in \widehat{F}\text{-}\mathbf{alg}(C\text{-}\mathbf{Coalg}(\mathcal{C}))$ satisfying $U(\hat{F}) = F$.*

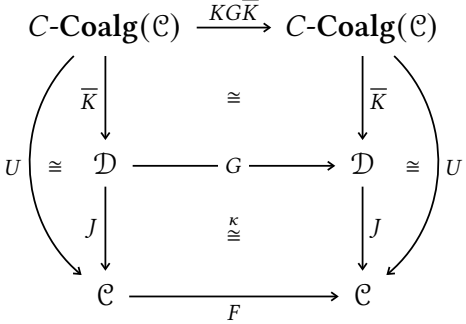
Proof See the Appendix on page 315. □

Using Proposition 3.3.1, we can lift initial algebras to the category of coalgebras for a comonad. In general, we may have an arbitrary lifting G of F along some functor J; we now show that as long as J is comonadic, the initial F-algebra can be lifted to an initial G-algebra by first lifting it to a category of coalgebras, then lowering it to $G\text{-}\mathbf{alg}(\mathcal{C})$.

Theorem 3.3.1

Let $F \colon \mathcal{C} \to \mathcal{C}$ be an endofunctor and $G \colon \mathcal{D} \to \mathcal{D}$ a lifting along a comonadic $J \colon \mathcal{D} \to \mathcal{C}$ via $\kappa \colon JG \xrightarrow{\cong} FJ$. Then, the initial F-algebra μF lifts to an initial G-algebra μG such that $J^\kappa \mu G \cong \mu F$.

Proof J is (weakly) comonadic, so it has a right adjoint $R \colon \mathcal{C} \to \mathcal{D}$ and an induced comonad $C \triangleq JR$ such that the canonical comparison functor $K \colon \mathcal{D} \to C\text{-}\mathbf{Coalg}(\mathcal{C})$ is an equivalence of categories, i.e. there is a pseudo-inverse functor $\overline{K} \colon C\text{-}\mathbf{Coalg}(\mathcal{C}) \to \mathcal{D}$ for which we have the natural isomorphisms $\overline{K}K \cong \mathrm{Id}_{\mathcal{D}}$ and $K\overline{K} \cong \mathrm{Id}_{C\text{-}\mathbf{Coalg}(\mathcal{C})}$. Consider the situation



where outermost isomorphisms are due to the property $UK \cong J$ of the comparison functor K. We have the natural isomorphism on the left below, which, by Proposition 3.1.1, induces the strict lifting on the right for an endofunctor $H \cong KG\overline{K}$.

$$\begin{array}{ccc} C\text{-}\mathbf{Coalg}(\mathcal{C}) & \xrightarrow{KG\overline{K}} & C\text{-}\mathbf{Coalg}(\mathcal{C}) \\ {\scriptstyle U}\downarrow & \cong & \downarrow{\scriptstyle U} \\ \mathcal{C} & \xrightarrow[F]{} & \mathcal{C} \end{array} \qquad \begin{array}{ccc} C\text{-}\mathbf{Coalg}(\mathcal{C}) & \xrightarrow{H} & C\text{-}\mathbf{Coalg}(\mathcal{C}) \\ {\scriptstyle U}\downarrow & & \downarrow{\scriptstyle U} \\ \mathcal{C} & \xrightarrow[F]{} & \mathcal{C} \end{array}$$

H is a lifting of $F\colon \mathcal{C} \to \mathcal{C}$ to C-coalgebras, putting us in the context of Proposition 3.3.1: the initial F-algebra (F, g) lifts to an initial H-algebra (H, h) with ① $U\mathit{H} = \mathit{F}$, and through the isomorphism $H \cong KG\overline{K}$, there is an associated initial algebra for $KG\overline{K}$ too, which we will also denote H. Moreover, since

$$\begin{array}{ccc} C\text{-}\mathbf{Coalg}(\mathcal{C}) & \xrightarrow{KG\overline{K}} & C\text{-}\mathbf{Coalg}(\mathcal{C}) \\ {\scriptstyle \overline{K}}\downarrow & \overset{\alpha}{\cong} & \downarrow{\scriptstyle \overline{K}} \\ \mathcal{D} & \xrightarrow[G]{} & \mathcal{D} \end{array}$$

and the equivalence K can be strengthened to an an adjoint equivalence $\overline{K} \dashv K$, we can use Lemma 3.3.1 to map the initial $KG\overline{K}$-algebra to the initial G-algebra $(\mathit{G}, f) \triangleq \overline{K}^{\alpha}(\mathit{H}, h)$. Applying Lemma 3.3.1 again, we get that $J^{\kappa}(\mathit{G}, f) \cong (\mathit{F}, g)$, as was required.

In summary, the initial F-algebra F lifts to an initial $KG\overline{K}$-algebra H which $\overline{K}$ maps to the initial G-algebra G, which can further be mapped back to F up to isomorphism. □

The theorem above shows that if G is a strong lifting of F along a weakly comonadic J, an initial F-algebra μF induces an initial G-algebra that J^{α} maps to an initial object isomorphic to the original μF. With a stronger assumption on J, the isomorphism can be strengthened to an equality on the carrier objects.

Corollary 3.3.1 *If in the above theorem J is strictly comonadic, we also have the equality of carrier objects of the initial algebras $J\mathit{G} = \mathit{F}$.*

Proof If J is strictly comonadic, $K\colon \mathcal{D} \to C\text{-}\mathbf{Coalg}(\mathcal{C})$ is an isomorphism of categories with ② $K\overline{K} = \mathrm{Id}$ and ③ $UK = J$. We then have the following calculation:

$$J\mathit{G} \triangleq J\overline{K}\mathit{H} \overset{③}{=} UK\overline{K}\mathit{H} \overset{②}{=} U\mathit{H} \overset{①}{=} \mathit{F}$$ □

Remark. The initial-algebra lifting theorem will be applied that the term syntax of a signature endofunctor Σ computed in families may be equipped with a C-coalgebra structure by initiality/structural recursion, as long as Σ itself lifts to coalgebras. If the comonad is the cofree presheaf comonad $\square\colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ (whose coalgebras are families with renaming structure, i.e. presheaves), the initial Σ-algebra lifts to an initial $\widehat{\Sigma}$-algebra in $\square$-**Coalg** by Proposition 3.3.1. This, by Theorem 3.3.1, gives the initial Ξ-algebra in presheaves, assuming the underlying family mapping of $\Xi\colon \mathbf{PSh}_S \to \mathbf{PSh}_S$ is Σ. For formalisation purposes, Proposition 3.3.1 ensures that the inductive datatype of terms admits a lawful recursive renaming operation. ⌟

3.4 Free distributive laws

As shown in this chapter, there is an equivalence of liftings of functors to monad algebras, and distributive laws between the functor and the monad. In practice we will need a way to "initialise" this equivalence by constructing a distributive law or lifting from first principles, and freely translate between them afterwards. In this section we will prove that constructing a distributive law for a *free* monad is possible by giving an arbitrary lifting, not just to algebras. We start by recalling some standard definitions and properties of free constructions – for more details, see e.g. Mac Lane (1971, Section IV).

Definition 3.4.1 Let $U : \mathcal{C} \to \mathcal{D}$ be a functor. A *free $\mathcal{C}$-object on $A \in \mathcal{D}$* is an object $FA \in \mathcal{C}$ and a morphism $\eta_A : A \to UFA \in \mathcal{D}$ such that for all $Y \in \mathcal{C}$ and $f : A \to UY \in \mathcal{D}$ there exists a unique extension $f^\sharp : FA \to Y \in \mathcal{C}$ such that

$$\begin{array}{ccc} A & \xrightarrow{\eta_A} & UFA \\ & \searrow^{f} & \downarrow{\scriptstyle Uf^\sharp} \\ & & UY \end{array} \tag{$F^\sharp$}$$

Existence of free objects for all $A \in \mathcal{D}$ implies a *free-forgetful adjunction* $\mathcal{C}(FA, Y) \cong \mathcal{D}(A, UY)$ for $A \in \mathcal{D}, Y \in \mathcal{C}$ and further induces the *free monad* $TX = UFX : \mathcal{D} \to \mathcal{D}$. ⌟

As shown by Mac Lane (1971, Section VI.3), the comparison functor $K : \mathcal{C} \to T\text{-}\mathbf{Alg}(\mathcal{D})$ maps every object $X \in \mathcal{C}$ to the T-algebra $(UX, TUX \triangleq UFUX \xrightarrow{U\varepsilon_X} UX)$. Freeness is employed both to construct free extension functions, and to prove equalities via uniqueness: to show that two morphisms $g, h : FA \to Y \in \mathcal{C}$ are equal, it is sufficient to show that they both factorise a map $A \to UY \in \mathcal{D}$ in the sense that $Ug \circ \eta_A = f^\sharp = Uh \circ \eta_A$. By the uniqueness of the free extension $f^\sharp$, both g and h must equal to $f^\sharp$ and therefore each other. This will be used in the following theorem to construct a distributive law for the free monad.

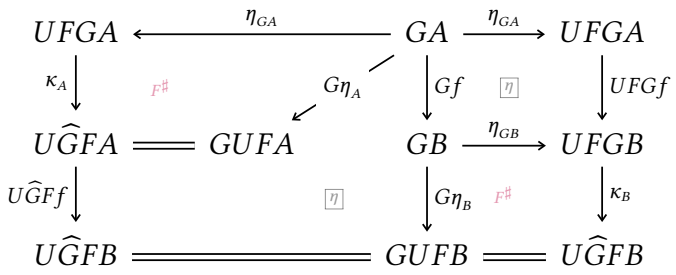
Theorem 3.4.1

In the context of of the above, let $G : \mathcal{D} \to \mathcal{D}$ be an endofunctor with $\widehat{G} : \mathcal{C} \to \mathcal{C}$ a strict lifting along U. Then, there is monad-functor distributive law $TG \Longrightarrow GT : \mathcal{D} \to \mathcal{D}$.

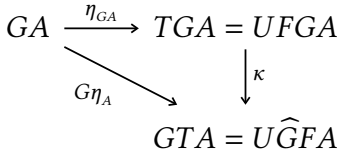
Proof $\widehat{G}$ is a strict lifting of G, so $GTA = GUFA = U\widehat{G}FA$, so the components of the distributive law will have the form $UFGA \to U\widehat{G}FA$ which is the image of a morphism $FGA \to \widehat{G}FA \in \mathcal{C}$ under the forgetful functor. This latter morphism out of the free functor can be induced by freeness, as the extension $\kappa_A : FGA \to \widehat{G}FA$ of $GA \xrightarrow{G\eta_A} GTA = U\widehat{G}FA$. Naturality requires that for all $f : A \to B \in \mathcal{D}$, $\kappa_B \circ TGf = GTf \circ \kappa_A$. We show that both composites come from morphisms in $\mathcal{C}$:

$$\begin{array}{ccc} FGA & & \\ {\scriptstyle TGf}\downarrow & & F : \mathcal{D} \to \mathcal{C} \\ FGB & & \\ {\scriptstyle \kappa_B}\downarrow & & \text{freeness} \\ \widehat{G}FB & & \end{array} \qquad \begin{array}{ccc} FGA & & \\ {\scriptstyle \kappa_A}\downarrow & & \text{freeness} \\ \widehat{G}FA & & \\ {\scriptstyle \widehat{G}Ff}\downarrow & & \widehat{G} : \mathcal{C} \to \mathcal{C} \\ \widehat{G}FB & & \end{array}$$

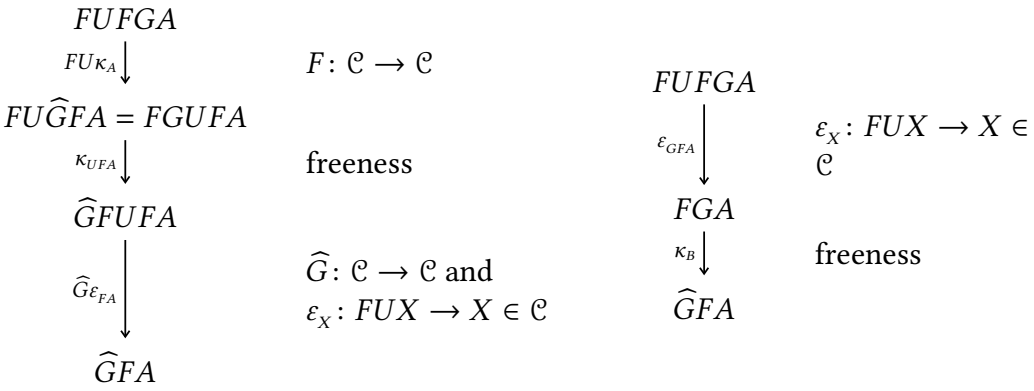
Moreover, they both factorise $GA \xrightarrow{Gf} GB \xrightarrow{G\eta_B} GUFB$, so the maps must be equal:



The unit axiom ($\varphi\lfloor\eta\rceil$) for the distributive law is simply Diagram ($F^\sharp$):



The multiplication axiom ($\varphi\lfloor\mu\rceil$) is established by the equality of the following two maps in $\mathcal{C}$:

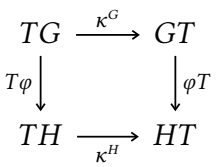


They both factorise the embedding $UFGA \xrightarrow{\kappa_A} U\widehat{G}FA$ and must therefore be equal. For simplicity, we will write κ both for the extension $(G\eta_A)^\sharp : FGA \to \widehat{G}FA$ and its value at U, the distributive law $TG \Longrightarrow GT$. □

A similar argument can be used to relate liftings of natural transformations, and maps of distributive laws for a free monad T.

Proposition 3.4.1 *For $G, H : \mathcal{D} \to \mathcal{D}$ with strict liftings $\widehat{G}, \widehat{H} : \mathcal{C} \to \mathcal{C}$, if a natural transformation $\varphi : G \Longrightarrow H : \mathcal{D} \to \mathcal{D}$ lifts to $\widehat{\varphi} : \widehat{G} \Longrightarrow \widehat{H} : \mathcal{C} \to \mathcal{C}$, the free monad $T : \mathcal{D} \to \mathcal{D}$ distributes over φ with respect to the induced $\kappa^G : TG \Longrightarrow GT$ and $\kappa^H : TH \Longrightarrow HT$.*

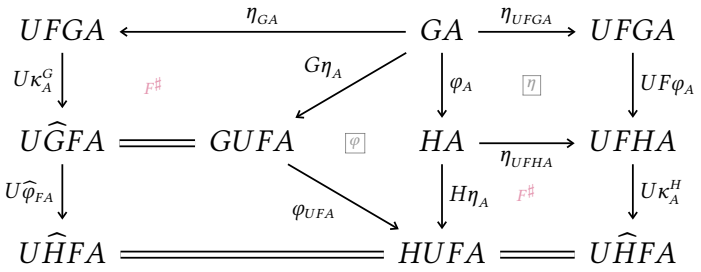
Proof Given the assumptions, we need to show that the following square commutes:



We reason by freeness: the two composites above are values of the following morphisms at U:

$$\begin{array}{l} FGA \\ \kappa^G_A \downarrow \quad \text{freeness} \\ \widehat{G}FA \\ \widehat{\varphi}_{TA} \downarrow \quad \widehat{\varphi}_X \in \mathcal{C}(\widehat{G}X, \widehat{H}X) \\ \widehat{H}FA \end{array} \qquad \begin{array}{l} FGA \\ F\varphi \downarrow \quad F\colon \mathcal{C} \to \mathcal{C} \\ FHA \\ \kappa^H_A \downarrow \quad \text{freeness} \\ \widehat{H}FA \end{array}$$

Both maps factorise the embedding $GA \xrightarrow{\varphi_A} HA \xrightarrow{H\eta_A} HUFA$, and are therefore equal:



□

Any lifting of $G\colon \mathcal{D} \to \mathcal{D}$ to $\widehat{G}\colon \mathcal{C} \to \mathcal{C}$ induces a distributive law $\kappa\colon TF \Longrightarrow FT$, but by Proposition 3.1.4, this distributive law itself gives rise to a lifting of G to T-algebras. The two liftings for G are related by the comparison functor $K\colon \mathcal{C} \to T\text{-}\mathbf{Alg}(\mathcal{D})$ as follows.

Corollary 3.4.1 *In the situation above, a lifting of $G\colon \mathcal{D} \to \mathcal{D}$ to $H\colon \mathcal{C} \to \mathcal{C}$ induces a lifting of G to $\widehat{G}\colon T\text{-}\mathbf{Alg}(\mathcal{D}) \to T\text{-}\mathbf{Alg}(\mathcal{D})$, and they commute with $K\colon \mathcal{C} \to T\text{-}\mathbf{Alg}(\mathcal{D})$:*

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{H} & \mathcal{C} \\ {\scriptstyle K}\downarrow & & \downarrow{\scriptstyle K} \\ T\text{-}\mathbf{Alg}(\mathcal{D}) & \xrightarrow[\widehat{G}]{} & T\text{-}\mathbf{Alg}(\mathcal{D}) \end{array}$$

Proof From above we know that $H\colon \mathcal{C} \to \mathcal{C}$ induces a distributive law $TG \Longrightarrow GT$, which, by Proposition 3.1.3, gives rise to a lifting of G to $T\text{-}\mathbf{Alg}(\mathcal{D}) \to T\text{-}\mathbf{Alg}(\mathcal{D})$. To relate the two liftings, we calculate using $K(X) \triangleq (UX, U\varepsilon_X\colon TUX \to UX) \in T\text{-}\mathbf{Alg}(\mathcal{D})$, with the algebra maps $U\varepsilon_{HX}$ and $GU\varepsilon_X \circ \kappa_{UX}$ equal by a series of zig-zag identities.

$$\begin{aligned} & K(HX) \\ &= \big(UHX, U\varepsilon_{HX}\colon UFHX \to UHX\big) \\ &= \big(GUX, U\varepsilon_{HX}\colon UF(GUX) \to GUX\big) \\ &= \big(GUX, UF(GUX) \xrightarrow{\kappa_{UX}} GUFX \xrightarrow{GU\varepsilon_X} GUX\big) \\ &= \widehat{G}\big(UX, U\varepsilon_X\colon UFX \to UX\big) \\ &= \widehat{G}(KX) \end{aligned}$$

□

The utility of these theorems will be demonstrated at the end of the next chapter at a high level, and as part of the familial model in Part III in more formal detail. We next address the structure of biclosed, powered, and enriched categories that underlie the theory of metasubstitution.

CHAPTER 4

Powering and enrichment

This section develops the theory of monoidal biclosed categories and powered functors between them (Section 4.1), continuing with the definition of the clone monad in Section 4.2 and powered monad maps into it in Section 4.3. This will underlie the formal derivation of metasubstitution in the familial model in Section 11.3, establishing the laws of metasubstitution purely from the construction of algebras for the free term monad.

4.1 Biclosed modular categories

We start by recalling the definition of monoidal categories, and right modular categories.

Definition 4.1.1 A *monoidal category* $\mathcal{V}$ is equipped with a *unit* $I \in \mathcal{V}$ and a *tensor* $\otimes\colon \mathcal{V}\times\mathcal{V} \to \mathcal{V}$, structural natural isomorphisms

$$\lambda_B\colon I \otimes B \cong B \qquad \rho_A\colon A \otimes I \cong A \qquad \alpha_{A,B,C}\colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$$

satisfying the two coherence conditions

$$\begin{array}{ccc} (A \otimes I) \otimes B & \xrightarrow{\alpha_{A,I,B}} & A \otimes (I \otimes B) \\ & {\scriptstyle \rho_A \otimes B} \searrow \quad \swarrow {\scriptstyle A \otimes \lambda_B} & \\ & A \otimes B & \end{array}$$

$$\begin{array}{ccc} ((A \otimes B) \otimes C) \otimes D & \xrightarrow{\alpha_{A,B,C} \otimes D} & (A \otimes (B \otimes C)) \otimes D \\ {\scriptstyle \alpha_{A \otimes B,C,D}} \downarrow & & \downarrow {\scriptstyle \alpha_{A,B \otimes C,D}} \\ (A \otimes B) \otimes (C \otimes D) & & A \otimes ((B \otimes C) \otimes D) \\ & {\scriptstyle \alpha_{A,B,C \otimes D}} \searrow \quad \swarrow {\scriptstyle A \otimes \alpha_{B,C,D}} & \\ & A \otimes (B \otimes (C \otimes D)) & \end{array}$$

Definition 4.1.2 Given a monoidal category $(\mathcal{V}, I, \otimes)$, a *left $\mathcal{V}$-modular category* is a category $\mathcal{C}$ with a *left action* $\oslash\colon \mathcal{V} \times \mathcal{C} \to \mathcal{C}$, and structural natural transformations

$$\lambda^{\oslash}_X\colon I \oslash X \cong X\colon \mathcal{C} \to \mathcal{C} \qquad \alpha^{\oslash}_{A,B,X}\colon (A \otimes B) \oslash X \cong A \oslash (B \oslash X)\colon \mathcal{V}\times\mathcal{V}\times\mathcal{C} \to \mathcal{C}$$

satisfying the coherence conditions

$$\begin{array}{ccc} (A \otimes I) \oslash X & \xrightarrow{\alpha^{\oslash}_{A,I,X}} & A \oslash (I \oslash X) \\ & {\scriptstyle \rho^{\otimes}_{A} \oslash X} \searrow \quad \swarrow {\scriptstyle A \oslash \lambda^{\oslash}_{X}} & \\ & A \oslash X & \end{array} \qquad \begin{array}{ccc} (I \otimes B) \oslash X & \xrightarrow{\alpha^{\oslash}_{I,B,X}} & I \oslash (B \oslash X) \\ & {\scriptstyle \lambda^{\otimes}_{B} \oslash X} \searrow \quad \swarrow {\scriptstyle \lambda^{\oslash}_{A \oslash X}} & \\ & B \oslash X & \end{array}$$

$$\begin{array}{ccccc} ((A \otimes B) \otimes C) \oslash X & & \xrightarrow{\alpha^{\otimes}_{A,B,C} \oslash X} & & (A \otimes (B \otimes C)) \oslash X \\ {\scriptstyle \alpha^{\oslash}_{A \otimes B, C, X}} \downarrow & & & & \downarrow {\scriptstyle \alpha^{\oslash}_{A, B \otimes C, X}} \\ (A \otimes B) \oslash (C \oslash X) & \xrightarrow[\alpha^{\oslash}_{A,B,C \oslash X}]{} & A \oslash (B \oslash (C \oslash X)) & \xrightarrow[A \oslash \alpha^{\oslash}_{B,C,X}]{} & A \oslash ((B \otimes C) \oslash X) \end{array}$$

A monoidal category is closed if for all $B \in \mathcal{V}$, the functor $(-) \otimes B$ has a right adjoint $[B, -]$. If the monoidal structure is not symmetric, right adjoints to $A \otimes (-)$ may give a different closed structure to the internal hom $[-, =]$.

Definition 4.1.3 A $\mathcal{V}$-modular category $(\mathcal{C}, \oslash)$ is *biclosed* if it is both left and right closed: for all $X \in \mathcal{C}$ and $B \in \mathcal{V}$, both functors $(-) \oslash X \colon \mathcal{V} \to \mathcal{C}$ and $A \oslash (-) \colon \mathcal{C} \to \mathcal{C}$ have right adjoints called the *right hom* $\langle X, - \rangle \colon \mathcal{C} \to \mathcal{V}$ and *left hom* $B \multimap (=) \colon \mathcal{C} \to \mathcal{C}$ respectively:

$$(-) \oslash X \dashv \langle X, - \rangle \qquad A \oslash (-) \dashv A \multimap (=)$$

Closure on both sides gives rise to a triangle of adjunctions between the action and the homs.

Proposition 4.1.1 *We have a two-variable adjunction between the tensor and the two homs:*

$$\begin{array}{ccc} & \mathcal{C}(A \oslash X, Y) & \\ & {\scriptstyle \varkappa} \swarrow \quad \searrow {\scriptstyle \kappa} & \\ \mathcal{V}(A, \langle X, Y \rangle) & \xrightarrow[\beta]{} & \mathcal{C}(X, A \multimap Y) \end{array}$$

Proof The unit and counit of the dual adjunction $\langle -, Y \rangle \vdash (-) \multimap Y$ derive, by the exponential transposes, from the adjunctions involving $\oslash$:

$$\frac{\beta^{Y}_{X} : X \to (\langle X, Y \rangle \multimap Y)}{\overline{\varkappa}^{X}_{Y} : (\langle X, Y \rangle \oslash X) \to Y} \qquad \frac{\overline{\beta}^{Y}_{A} : A \to \langle A \multimap Y, Y \rangle}{\overline{\kappa}^{A}_{Y} : (A \oslash (A \multimap Y)) \to Y}$$

The hom-set isomorphisms above are natural in $A \in \mathcal{V}^{\mathrm{op}}$, $X \in \mathcal{C}^{\mathrm{op}}$, and $Y \in \mathcal{C}$. Thus, given $f \colon B \to A$, $g \colon W \to X$, and $h \colon Y \to Z$, naturality amounts to the following composite transpositions for a morphism $k \colon A \oslash X \to Y$:

$$\frac{B \xrightarrow{f} A \xrightarrow{\overline{\varkappa}(k)} \langle X, Y \rangle \xrightarrow{\langle g, h \rangle} \langle W, Z \rangle}{B \oslash W \xrightarrow{f \oslash g} A \oslash X \xrightarrow{k} Y \xrightarrow{h} Z}$$

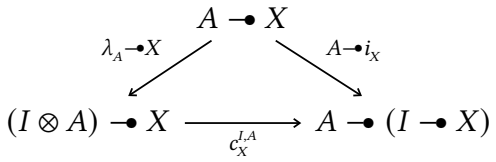
$$\frac{B \oslash W \xrightarrow{f \oslash g} A \oslash X \xrightarrow{k} Y \xrightarrow{h} Z}{W \xrightarrow{f} X \xrightarrow{\kappa(k)} A \multimap Y \xrightarrow{g \multimap h} B \multimap Z}$$

Left and right homs are closely connected to *powering* and *enrichment* respectively (Janelidze and Kelly, 2001; McDermott and Uustalu, 2022), and in Chapter 5 we will discuss how these notions relate to closed modular categories. For the theory presented in this chapter, it will be beneficial to focus on powered and enriched categories, and take biclosed categories to possess both powering and enrichment.

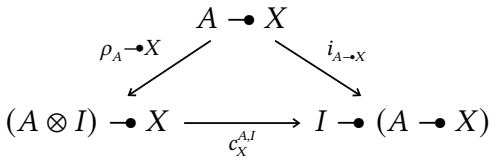
Proposition 4.1.2 *The left hom* $(-) \multimap (=) \colon \mathcal{V}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{C}$ *forms a [powering] of* $\mathcal{C}$ *over* $\mathcal{V}$*, in that it is equipped with natural isomorphisms*

$$i_B \colon (I \multimap B) \cong B \qquad c_X^{A,B} \colon \big((A \otimes B) \multimap X\big) \cong \big(B \multimap (A \multimap X)\big)$$

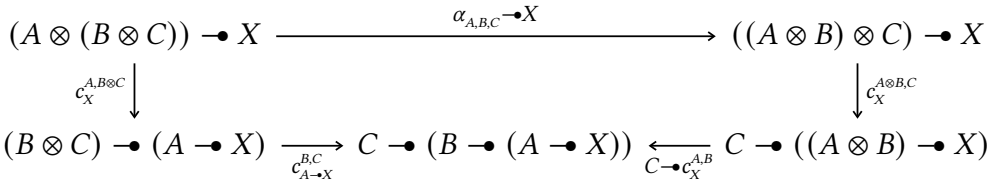
that satisfy the following coherence conditions:



$(c\lambda)$



$(c\rho)$

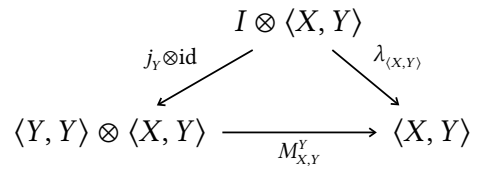


$(c\alpha)$

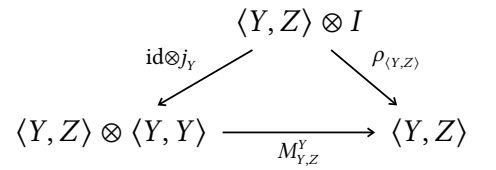
The right hom $\langle -, = \rangle \colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$ *forms an* enrichment *of* $\mathcal{C}$ *over* $\mathcal{V}$*, in that it is equipped structural transformations (natural and dinatural in the appropriate components)*

$$j_X \colon I \to \langle X, X \rangle \qquad M_{X,Z}^{Y} \colon \langle Y, Z \rangle \otimes \langle X, Y \rangle \to \langle X, Z \rangle$$

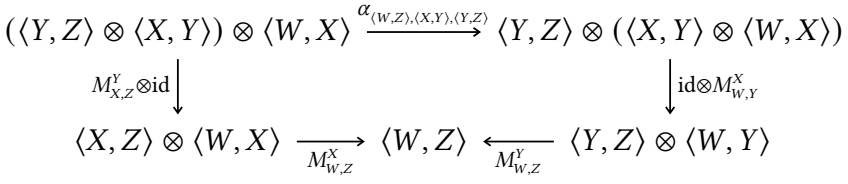
such that the functions $\mathcal{C}(X, Y) \to \mathcal{V}(I, \langle X, Y \rangle)$ *that map* $\mathcal{C}$*-morphisms* $f \colon X \to Y$ *to the composite* $I \xrightarrow{j_X} \langle X, X \rangle \xrightarrow{\langle X, f \rangle} \langle X, Y \rangle$ *are bijections, and the following coherence conditions are satisfied:*



$(M\lambda)$



$(M\rho)$

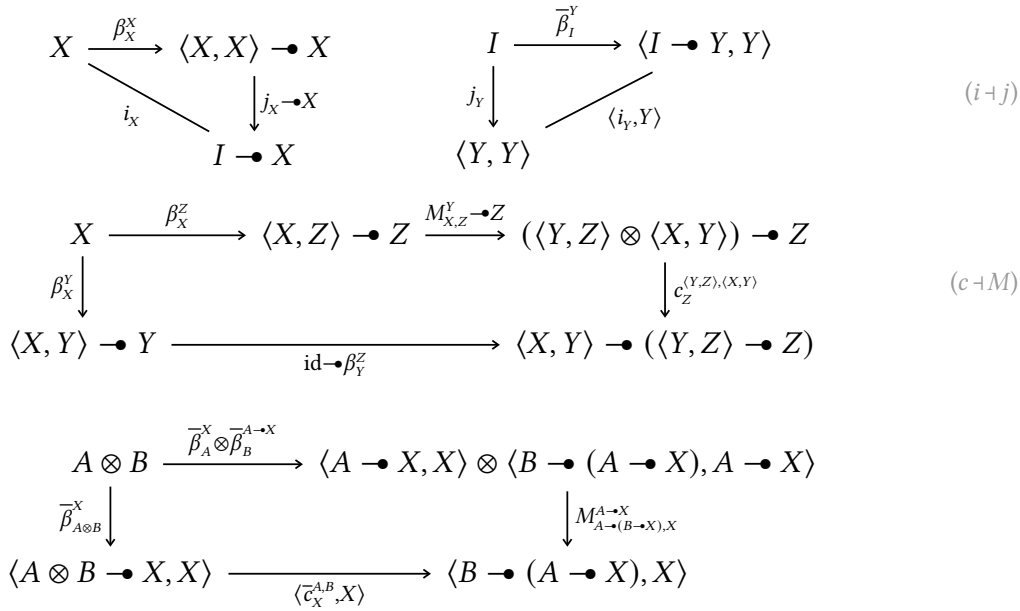


$(M\alpha)$

As shown by McDermott and Uustalu (2022), the data involved with powering and being a left hom are equivalent, but the equivalence between right homs and enrichment holds with the further assumption that $(\mathcal{V}, [-, =])$ is closed and the adjunction $(-) \oslash X \dashv \langle X, - \rangle$ internalises as an isomorphism $\langle A \oslash X, Y \rangle \cong [A, \langle X, Y \rangle]$. This asymmetry is discussed further in a skew setting by Uustalu et al. (2020) and in Chapter 5. For the purposes of this chapter, we will use

the language of powering, enrichment and biclosure, rarely using the module structure. In particular, we will benefit from the following interderivability lemmas between the powering and enrichment structure, derived from the naturality of the adjunctions in Proposition 4.1.1.

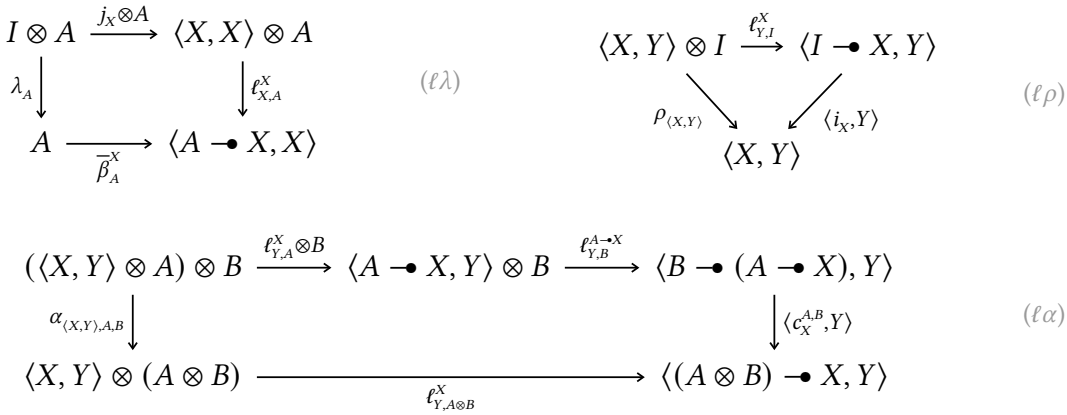
Corollary 4.1.1 *The transformations above are related as follows:*



Next, we axiomatise a utility operation that combines tensors, powering, and enrichment – it will simplify constructions later in the chapter.

Lemma 4.1.1 *For a biclosed $\mathcal{V}$-modular category $\mathcal{C}$ and all $A \in \mathcal{V}$, $X, Y \in \mathcal{C}$, we have a natural family of maps that is compatible with the biclosed structure:*

$$\ell^X_{Y,A} \colon \langle X, Y\rangle \otimes A \to \langle A \multimap X, Y\rangle \colon \mathcal{V} \times \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$$



Proof See the Appendix on page 317. □

The notion of a powered functor given by McDermott and Uustalu (2022, Definition 6.3) is equivalent to the following presentation.

Definition 4.1.4 A *powered functor* $F\colon \mathcal{C} \to \mathcal{D}$ between two right-closed $\mathcal{V}$-modular categories $(\mathcal{C}, \multimap)$ and $(\mathcal{D}, \mathbin{-\!\!\blacksquare})$ is equipped with a *powering* natural transformation

$$p_{A,Y}\colon F(A \multimap X) \to (A \mathbin{-\!\!\blacksquare} FX)\colon \mathcal{V}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$$

that satisfies the coherence conditions

$$\begin{array}{ccc} & FX & \\ {\scriptstyle F\overrightarrow{i_X}}\swarrow & & \searrow{\scriptstyle \overrightarrow{i_{FX}}} \\ F(I \multimap X) & \xrightarrow[p_{I,X}]{} & I \mathbin{-\!\!\blacksquare} FX \end{array} \qquad (pi)$$

$$\begin{array}{ccccc} F\big((A \otimes B) \multimap X\big) & & \xrightarrow{p_{A\otimes B,X}} & & (A \otimes B) \mathbin{-\!\!\blacksquare} FX \\ {\scriptstyle Fc_X^{\multimap,A,B}}\downarrow & & & & \downarrow{\scriptstyle c_{FX}^{\mathbin{-\!\!\blacksquare},A,B}} \\ F\big(A \multimap (B \multimap X)\big) & & & & A \mathbin{-\!\!\blacksquare} (B \mathbin{-\!\!\blacksquare} FX) \\ & {\scriptstyle p_{A,B\multimap X}}\searrow & & \nearrow{\scriptstyle A\multimap p_{B,X}} & \\ & & A \mathbin{-\!\!\blacksquare} F(B \multimap X) & & \end{array} \qquad (pc)$$

A morphism of powered functors is a natural transformation $\varphi\colon F \Longrightarrow G$ satisfying

$$\begin{array}{ccc} F(A \multimap X) & \xrightarrow{p^F_{A,X}} & A \mathbin{-\!\!\blacksquare} FX \\ {\scriptstyle \varphi_{A\multimap X}}\downarrow & & \downarrow{\scriptstyle A\mathbin{-\!\!\blacksquare}\varphi_X} \\ G(A \multimap X) & \xrightarrow[p^G_{A,X}]{} & A \mathbin{-\!\!\blacksquare} GX \end{array} \qquad (\varphi \lfloor p \rceil)$$

The 2-category of $\mathcal{V}$-powered categories (equivalently, right-closed $\mathcal{V}$-modular categories), powered functors and natural transformations between them will be denoted $\mathcal{V}$-**Pow** or just **Pow**, as will be its full subcategory of biclosed $\mathcal{V}$-modular categories. ⌟

Definition 4.1.5 A *powered monad* $T = (T, p, \eta, \mu)$ on a $\mathcal{V}$-powered category $\mathcal{C}$ is a monad $T\colon \mathcal{C} \to \mathcal{C}$ with a powering p that is compatible with the monad structure:

$$\begin{array}{ccc} & A \multimap X & \\ {\scriptstyle \eta_{A\multimap X}}\swarrow & & \searrow{\scriptstyle A\multimap\eta_X} \\ T(A \multimap X) & \xrightarrow[p_{A,X}]{} & A \multimap TX \end{array} \qquad (p\eta)$$

$$\begin{array}{ccccc} TT(A \multimap X) & \xrightarrow{Tp_{A,X}} & T(A \multimap TX) & \xrightarrow{p_{A,TX}} & A \multimap TTX \\ {\scriptstyle \mu_{A\multimap X}}\downarrow & & & & \downarrow{\scriptstyle A\multimap\mu_X} \\ T(A \multimap X) & & \xrightarrow[p_{A,X}]{} & & A \multimap TX \end{array} \qquad (p\mu)$$

A morphism $\varphi\colon (S, p^S, \eta^S, \mu^S) \to (T, p^T, \eta^T, \mu^T)$ of powered monads is a powered functor morphism $(S, p^S) \to (T, p^T)$ that is also monad morphism:

$$\begin{array}{ccc} & I & \\ {\scriptstyle \eta^S}\swarrow & & \searrow{\scriptstyle \eta^T} \\ S & \xrightarrow[\varphi]{} & T \end{array} \qquad (\varphi \lfloor \eta \rceil) \qquad\qquad \begin{array}{ccc} SS & \xrightarrow{\varphi\varphi} & TT \\ {\scriptstyle \mu^S}\downarrow & & \downarrow{\scriptstyle \mu^T} \\ S & \xrightarrow[\varphi]{} & T \end{array} \qquad (\varphi \lfloor \mu \rceil)$$

The category of powered monads on a $\mathcal{V}$-powered category $\mathcal{C}$ and powered monad morphisms will be denoted $\mathbf{PowM}(\mathcal{C})$. ⌟

Remark. The laws $(p\eta)$ and $(p\mu)$ express η and μ as morphisms of powered functors, so as expected, a powered monad is a monoid in the category of powered endofunctors $\mathbf{Pow}(\mathcal{C}, \mathcal{C})$. ⌟

Although the construction of powered monad structure will be easier in practice, a more useful axiomatisation will be that of an enriched Kleisli triple and its corresponding algebras (cf. extension systems of Marmolejo and Wood (2010)).

Definition 4.1.6 An endofunctor $T\colon \mathcal{C} \to \mathcal{C}$ forms an *enriched Kleisli triple* if it is equipped with natural transformations $\eta_X\colon X \to TX$ and $\Upsilon^X_Y\colon \langle X, TY\rangle \to \langle TX, TY\rangle$ satisfying

$$\begin{array}{ccc} \langle X,TY\rangle & \xrightarrow{\Upsilon^X_Y} & \langle TX,TY\rangle \\ & \searrow & \downarrow{\scriptstyle \langle \eta_X,\mathrm{id}\rangle} \\ & & \langle X,TY\rangle \end{array} \qquad (\eta\Upsilon)$$

$$\begin{array}{ccc} I & \xrightarrow{j_{TX}} & \langle TX,TX\rangle \\ {\scriptstyle j_X}\downarrow & & \uparrow{\scriptstyle \Upsilon^X_X} \\ \langle X,X\rangle & \xrightarrow[\langle X,\eta_X\rangle]{} & \langle X,TX\rangle \end{array} \qquad (\Upsilon\eta)$$

$$\begin{array}{ccccc} \langle Y,TZ\rangle \otimes \langle X,TY\rangle & \xrightarrow{\Upsilon^Y_Z\otimes\mathrm{id}} & \langle TY,TZ\rangle \otimes \langle X,TY\rangle & \xrightarrow{M^{TY}_{X,TZ}} & \langle X,TZ\rangle \\ {\scriptstyle \Upsilon^Y_Z\otimes\Upsilon^X_Y}\downarrow & & & & \downarrow{\scriptstyle \Upsilon^X_Z} \\ \langle TX,TY\rangle \otimes \langle TY,TZ\rangle & & \xrightarrow[M^{TY}_{TX,TZ}]{} & & \langle TX,TZ\rangle \end{array} \qquad (\Upsilon\Upsilon)$$

⌟

Definition 4.1.7 An *algebra for an enriched Kleisli triple* (T, η, Υ) is an object $A \in \mathcal{C}$ with a natural extension operator $\sharp^X\colon \langle X, A\rangle \to \langle TX, A\rangle$, satisfying

$$\begin{array}{ccc} \langle X,A\rangle & \xrightarrow{\sharp^X} & \langle TX,A\rangle \\ & \searrow & \downarrow{\scriptstyle \langle \eta_X,\mathrm{id}\rangle} \\ & & \langle X,A\rangle \end{array} \qquad (\eta\Upsilon)$$

$$\begin{array}{ccccc} \langle Y,A\rangle \otimes \langle X,TY\rangle & \xrightarrow{\sharp^Y\otimes\mathrm{id}} & \langle TY,A\rangle \otimes \langle X,TY\rangle & \xrightarrow{M^{TY}_{X,A}} & \langle X,A\rangle \\ {\scriptstyle \sharp^Y\otimes\Upsilon^X_Y}\downarrow & & & & \downarrow{\scriptstyle \sharp^X} \\ \langle TY,A\rangle \otimes \langle TX,TY\rangle & & \xrightarrow[M^{TY}_{TX,A}]{} & & \langle TX,A\rangle \end{array} \qquad (\Upsilon\sharp)$$

⌟

The relationship of powered monad and enriched Kleisli structure will be developed next.

4.2 Powered clone monad

The presence of two adjoint closed structures lends itself naturally to the study of their induced monad, for which $\beta^Y_X\colon X \to (\langle X, Y\rangle \multimap Y)$ is the unit. Fiore (2013) formally connected A. Kock's (1970) work of this *double-dualisation monad* with the right adjoint of functor application studied by Kelly and Power (1993), recognising them as a generalisation of *abstract clones* (Taylor, 1993; Arkor and McDermott, 2021). We adapt these to the right $\mathcal{V}$-module setting, streamlining the proofs using the transformations introduced in the previous section.

Definition 4.2.1 Let $(\mathcal{C}, \langle -,=\rangle)$ and $(\mathcal{D}, \pitchfork)$ be categories enriched and powered over $\mathcal{V}$, respectively. For $Y \in \mathcal{C}$ and $Z \in \mathcal{D}$, the *clone functor* $(\!|Y, Z|\!)\colon \mathcal{C} \to \mathcal{D}$ is defined as

$$(\!|Y,Z|\!)X \triangleq \langle X, Y\rangle \pitchfork Z$$

If $(\mathcal{C}, \langle -,=\rangle, \multimap)$ is biclosed, for $Y \in \mathcal{C}$, the *endoclone functor* $(\!|Y, Y|\!)\colon \mathcal{C} \to \mathcal{C}$ will usually be abbreviated as $(\!|Y|\!)$; note that $(\!|-|\!) = Y \mapsto (\!|Y|\!)$ is not functorial due to the mixed variance. ⌟

Definition 4.2.2 The *clone evaluation map* $(\!|\varepsilon|\!)^Y_Z\colon (\!|Y,Z|\!)Y \to Z$ is the composite

$$(\!|\varepsilon|\!)^Y_Z\colon \langle Y,Y\rangle \pitchfork Z \xrightarrow{j^{\langle\rangle}_Y \pitchfork Z} I \pitchfork Z \xrightarrow{i^{\pitchfork}_Z} Z$$

⌟

Being defined in terms of the biclosed structure, it is perhaps not surprising that the clone functor is itself compatible with the right hom and right action in that it is powered. Note below the interplay between the c and ℓ operations we defined above, and how their laws fit together to give relatively simple proofs of the powering axioms.

Proposition 4.2.1 *For all $Y \in \mathcal{C}, Z \in \mathcal{D}$, the clone functor $(\!|Y, Z|\!) \colon \mathcal{C} \to \mathcal{D}$ possesses a powering*

$$p^{(\!|Y,Z|\!)}_{A,X} \colon (\!|Y, Z|\!)(A \multimap X) \;\to\; (A \mathbin{-\!\!\blacksquare} (\!|Y, Z|\!)X)$$

Proof The powering is defined as the following composite, with laws given below:

$$\langle A \multimap X, Y\rangle \mathbin{-\!\!\blacksquare} Z \xrightarrow{\ell^X_{Y,A} \mathbin{-\!\!\blacksquare} Z} \langle X, Y\rangle \otimes A \mathbin{-\!\!\blacksquare} Z \xrightarrow{c^{A,\langle X,Y\rangle}_Z} A \mathbin{-\!\!\blacksquare} (\langle X, Y\rangle \mathbin{-\!\!\blacksquare} Z)$$

$$\begin{array}{ccc}
\langle I \multimap X, Y\rangle \mathbin{-\!\!\blacksquare} Z & \xleftarrow{\langle i^{\rightarrow}_X, Y\rangle \mathbin{-\!\!\blacksquare} Z} & \langle X, Y\rangle \mathbin{-\!\!\blacksquare} Z \\
\ell\rho & & \\
{\scriptstyle \ell^X_{Y,I} \mathbin{-\!\!\blacksquare} Z}\downarrow & \swarrow {\scriptstyle \rho_{\langle X,Y\rangle} \mathbin{-\!\!\blacksquare} Z} & \downarrow{\scriptstyle i^{\blacksquare}_{\langle X,Y\rangle} \mathbin{-\!\!\blacksquare} Z} \\
(\langle X, Y\rangle \otimes I) \mathbin{-\!\!\blacksquare} Z & \xrightarrow[c^{I,\langle X,Y\rangle}_Z]{c\rho} & I \mathbin{-\!\!\blacksquare} (\langle X, Y\rangle \mathbin{-\!\!\blacksquare} Z)
\end{array}$$

$$\begin{array}{ccc}
\langle A \otimes B \multimap X, Y\rangle \mathbin{-\!\!\blacksquare} Z & \xrightarrow{\langle c^{A,B}_X, Y\rangle \mathbin{-\!\!\blacksquare} Z} & \langle B \multimap (A \multimap X), Y\rangle \mathbin{-\!\!\blacksquare} Z \\
{\scriptstyle \ell^X_{Y,A\otimes B} \mathbin{-\!\!\blacksquare} Z}\downarrow & t\alpha & \downarrow{\scriptstyle \ell^{B\multimap X}_{Y,A} \mathbin{-\!\!\blacksquare} Z} \\
(\langle X, Y\rangle \otimes (A \otimes B)) \mathbin{-\!\!\blacksquare} Z & \xleftarrow{\alpha_{\langle X,Y\rangle,A,B} \mathbin{-\!\!\blacksquare} Z} & (\langle A \multimap X, Y\rangle \otimes B) \mathbin{-\!\!\blacksquare} Z \\
{\scriptstyle c^{A\otimes B,\langle X,Y\rangle}_Z}\downarrow & & \downarrow{\scriptstyle c^{A,\langle B\multimap X,Y\rangle}_Z} \\
(A \otimes B) \mathbin{-\!\!\blacksquare} (\!|Y, Z|\!)X & (\langle X, Y\rangle \otimes A) \otimes B) \mathbin{-\!\!\blacksquare} Z & \\
{\scriptstyle c^{A,B}_{(\!|Y,Z|\!)X}}\downarrow & c\alpha \quad {\scriptstyle c^{A,B\otimes\langle X,Y\rangle}_Z}\downarrow & \boxed{c}_2 \\
B \mathbin{-\!\!\blacksquare} (A \mathbin{-\!\!\blacksquare} (\!|Y, Z|\!)X) & \xleftarrow[A \mathbin{-\!\!\blacksquare} c^{B,\langle X,Y\rangle}_Z]{} B \mathbin{-\!\!\blacksquare} (((\langle X, Y\rangle \otimes A) \mathbin{-\!\!\blacksquare} Z) & \xleftarrow[B \mathbin{-\!\!\blacksquare} (\ell^X_{Y,B} \mathbin{-\!\!\blacksquare} Z)]{} B \mathbin{-\!\!\blacksquare} (\!|Y, Z|\!)(B \multimap X)
\end{array}$$

□

Many of our calculations will happen in the category of powered functors, so reframing the powering transformation as an operation on clones will be useful.

Proposition 4.2.2 *For a powered functor $F \colon \mathcal{C} \to \mathcal{D}$ between two biclosed $\mathcal{V}$-modular categories, the* clone powering *natural transformation is a powered functor morphism below, with components $\left((\!|p|\!)^F_{Y,Z}\right)_X \triangleq p^F_{\langle X,Y\rangle,Z} \colon F(\langle X, Y\rangle \multimap Z) \to (\langle X, Y\rangle \mathbin{-\!\!\blacksquare} FZ)$.*

$$(\!|p|\!)^F_{Y,Z} \colon F(\!|Y, Z|\!) \Longrightarrow (\!|Y, FZ|\!) \colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$$

Proof The powering-preservation law expands to the following:

$$\begin{array}{ccc}
F(\langle A \multimap X, Y\rangle \multimap Z) & \xrightarrow{p^F_{\langle A\multimap X,Y\rangle,Z}} & \langle A \multimap X, Y\rangle \multimap FZ \\
{\scriptstyle F(\ell^X_{Y,A} \multimap Z)}\downarrow & \boxed{s}_1 & \downarrow{\scriptstyle \ell^X_{Y,A} \mathbin{-\!\!\blacksquare} FZ} \\
F\big(\langle X, Y\rangle \otimes A \multimap Z\big) & \xrightarrow[p^F_{A\otimes\langle X,Y\rangle,Z}]{} & A \otimes \langle X, Y\rangle \mathbin{-\!\!\blacksquare} FZ \\
{\scriptstyle Fc^{A,\langle X,Y\rangle}_Z}\downarrow & & \\
F(A \multimap \langle X, Y\rangle \multimap Z) & pc & \downarrow{\scriptstyle c^{A,\langle X,Y\rangle}_{FZ}} \\
{\scriptstyle p^F_{A,\langle X,Y\rangle\multimap Z}}\downarrow & & \\
A \mathbin{-\!\!\blacksquare} F(\langle X, Y\rangle \multimap Z) & \xrightarrow[A \mathbin{-\!\!\blacksquare} p^F_{\langle X,Y\rangle,Z}]{} & A \mathbin{-\!\!\blacksquare} \langle X, Y\rangle \multimap FZ
\end{array}$$

□

Clones capture a two-level form of closure: the clone functor $(\!|-,=|\!) \colon \mathcal{C}^{\mathrm{op}} \times \mathcal{D} \to \mathbf{Pow}(\mathcal{C}, \mathcal{D})$ is a functor-valued profunctor that gives a powered functor $(\!|X, Y|\!) \colon \mathcal{C} \to \mathcal{D}$ for all $X \in \mathcal{C}$ and $Y \in \mathcal{D}$, but itself comes with transformations for unit $\mathrm{Id} \Longrightarrow (\!|X, X|\!)$, clone composition $(\!|Y, Z|\!) \circ (\!|X, Y|\!) \Longrightarrow (\!|X, Z|\!)$, and evaluation $(\!|X, Y|\!)X \Longrightarrow Y$. This can be formalised using the notion of enrichment in a (strict) bicategory, a generalisation of enrichment in a monoidal category introduced by Garner and Shulman (2016, Section 15), whose terminology we adopt.

Proposition 4.2.3 *Clones form the morphisms of a category enriched in the 2-category $\mathcal{V}$-**Pow**.*

Proof We define a category **Clones** enriched in $\mathcal{V}$-**Pow** whose objects and extents are $X \in \mathcal{C}$ for a biclosed $\mathcal{V}$-modular category $\mathcal{C}$, and morphisms between $X \in \mathcal{C}$ and $Y \in \mathcal{D}$ are clones $(\!|X, Y|\!) \colon \mathcal{C} \to \mathcal{D}$, which are powered functors by Proposition 4.2.1. For each $X \in \mathcal{C}$, the identity $(\!|\eta|\!)^X \colon \mathrm{Id}_{\mathcal{C}} \Longrightarrow (\!|X, X|\!) \colon \mathcal{C} \to \mathcal{C}$ has components $\beta^X_W \colon W \to \langle W, X\rangle \multimap X$, and for each $X \in \mathcal{C}$, $Y \in \mathcal{D}$ and $Z \in \mathcal{E}$, the composition $(\!|\chi|\!)^{X,Y,Z} \colon (\!|Y, Z|\!) \circ (\!|X, Y|\!) \Longrightarrow (\!|X, Z|\!)$ has components

$$(\!|\chi|\!)^{X,Y,Z}_W \colon \langle\langle W, X\rangle_{\mathcal{C}} \multimap_{\mathcal{D}} Y, Y\rangle_{\mathcal{D}} \multimap_{\mathcal{E}} Z \xrightarrow{\overline{\beta}^Y_{\langle W,X\rangle \multimap Z}} \langle W, X\rangle_{\mathcal{C}} \multimap_{\mathcal{E}} Z$$

These are both powered functor morphisms: the powering-preservation of $(\!|\eta|\!)$ is equivalent to Diagram ($t \dashv c$), while the powering-preservation for $(\!|\chi|\!)$ is

$$\begin{array}{ccc}
\langle\langle A \multimap W, X\rangle \multimap Y, Y\rangle \multimap Z & \xrightarrow{\overline{\beta}^Y_{\langle A \multimap W,X\rangle \multimap Z}} & \langle A \multimap W, X\rangle \multimap Z \\
\downarrow{\scriptstyle \langle \ell^W_{X,A} \multimap Y, Y\rangle \multimap Z} & \boxed{\overline{\beta}}_2 & \downarrow{\scriptstyle \ell^W_{X,A} \multimap Z} \\
\langle A \otimes \langle W, X\rangle \multimap Y, Y\rangle \multimap Z & \xrightarrow[\overline{\beta}^Y_{A \otimes \langle W,X\rangle \multimap Z}]{} & A \otimes \langle W, X\rangle \multimap Z \\
\downarrow{\scriptstyle \langle c^{A,\langle W,X\rangle}_Y, Y\rangle \multimap Z} & & \| \\
\langle A \multimap \langle W, X\rangle \multimap Y, Y\rangle \multimap Z & t \dashv c & \| \\
\downarrow{\scriptstyle \ell^{\langle W,X\rangle \multimap Y}_{Y,A} \multimap Z} & & \| \\
A \otimes \langle\langle W, X\rangle \multimap Y, Y\rangle \multimap Z & \xrightarrow{A \otimes \overline{\beta}^Y_{\langle W,X\rangle \multimap Z}} & A \otimes \langle W, X\rangle \multimap Z \\
\downarrow{\scriptstyle c^{A,\langle\langle W,X\rangle \multimap Y,Y\rangle}_Z} & \boxed{c}_2 & \downarrow{\scriptstyle c^{A,\langle W,X\rangle}_Z} \\
A \multimap \langle\langle W, X\rangle \multimap Y, Y\rangle \multimap Z & \xrightarrow[A \multimap (\overline{\beta}^Y_{\langle W,X\rangle} \multimap Z)]{} & A \multimap \langle W, X\rangle \multimap Z
\end{array}$$

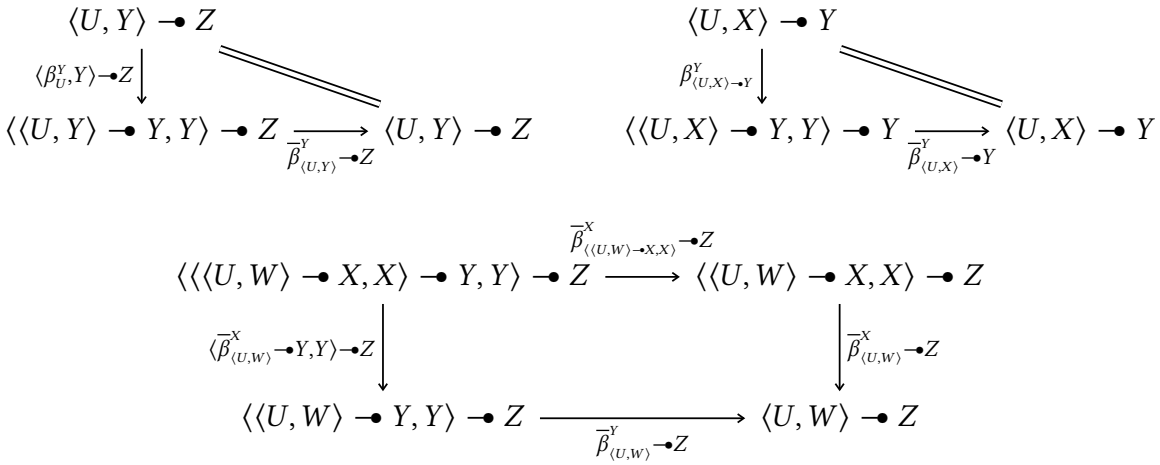
The coherence conditions between the unit and composition operations

$$\begin{array}{ccc}
\mathrm{Id}(\!|X, Y|\!) & & \\
\downarrow{\scriptstyle (\!|\eta|\!)^Y (\!|Y,X|\!)} & \searrow\!\!\searrow & \\
(\!|Y, Y|\!)(\!|X, Y|\!) & \xrightarrow[(\!|\chi|\!)^{X,X,Y}]{} & (\!|X, Y|\!)
\end{array} \qquad (\!|\eta\chi|\!)$$

$$\begin{array}{ccc}
(\!|Y, Z|\!)\mathrm{Id} & & \\
\downarrow{\scriptstyle (\!|Y,X|\!)(\!|\eta|\!)^Y} & \searrow\!\!\searrow & \\
(\!|Y, Z|\!)(\!|Y, Y|\!) & \xrightarrow[(\!|\chi|\!)^{Y,Y,Z}]{} & (\!|Y, Z|\!)
\end{array} \qquad (\!|\chi\eta|\!)$$

$$\begin{array}{ccc}
(\!|Y, Z|\!)(\!|X, Y|\!)(\!|W, X|\!) & \xrightarrow{(\!|\chi|\!)^{X,Y,Z}(\!|W,X|\!)} & (\!|X, Z|\!)(\!|W, X|\!) \\
\downarrow{\scriptstyle (\!|Y,Z|\!)(\!|\chi|\!)^{W,X,Y}} & & \downarrow{\scriptstyle (\!|\chi|\!)^{W,X,Z}} \\
(\!|Y, Z|\!)(\!|W, Y|\!) & \xrightarrow[(\!|\chi|\!)^{W,Y,Z}]{} & (\!|W, Z|\!)
\end{array} \qquad (\!|\chi\chi|\!)$$

reduce to the zig-zag axioms of the dual adjunction in Proposition 4.1.1, and naturality of $\overline{\beta}$:



□

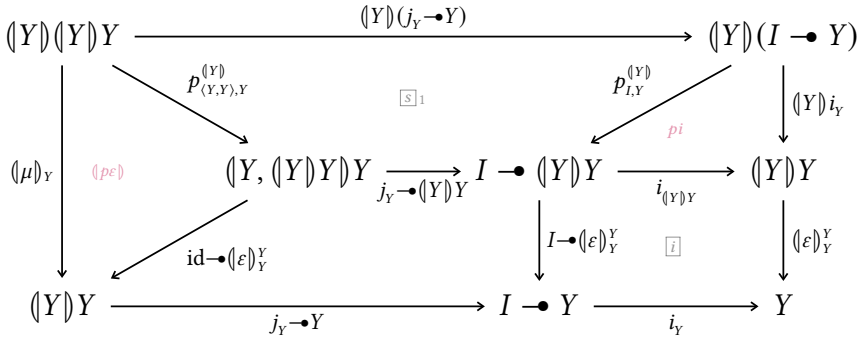
Just as every endo-hom is a monoid in a monoidal closed category, the endo-clone is a monad in the enriched category **Clones**.

Corollary 4.2.1 *For any $Y \in \mathcal{C}$, the biclosed endo-clone $(\!|Y|\!) : \mathcal{C} \to \mathcal{C}$ is a powered monad, and Y is an algebra for this monad.*

Proof One-object categories enriched in the 2-category $\mathcal{V}$-**Pow** correspond precisely to monads in $\mathcal{V}$-**Pow**, i.e. an object $Y \in \mathcal{C}$, with identity $(\!|\eta|\!)^Y : \mathrm{Id}_{\mathcal{C}} \Longrightarrow (\!|Y|\!) \in \mathcal{V}\text{-}\mathbf{Pow}(\mathcal{C},\mathcal{C})$ and composition $(\!|\mu|\!)^Y : (\!|Y|\!)(\!|Y|\!) \Longrightarrow (\!|Y|\!) \in \mathcal{V}\text{-}\mathbf{Pow}(\mathcal{C},\mathcal{C})$ satisfying unit and associativity laws (Garner and Shulman, 2016, Example 15.3). The algebra structure on Y is the evaluation map

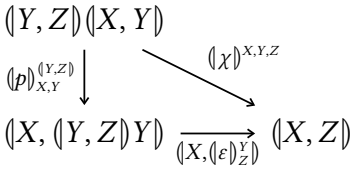
$$(\!|\varepsilon|\!)^Y_Y : (\!|Y|\!)Y \to Y = \big(\langle Y, Y\rangle \multimap Y \xrightarrow{j_Y \multimap Y} I \multimap Y \xrightarrow{i_Y} Y\big)$$

The unit law $Y \xrightarrow{(\!|\eta|\!)^Y_Y} (\!|Y|\!)Y \xrightarrow{(\!|\varepsilon|\!)^Y_Y} Y = \mathrm{id}_Y$ is equivalently the β-transposition Diagram $(i \dashv j)$ between $\langle Y, Y\rangle \multimap Y \xrightarrow{j_Y \multimap Y} I \multimap Y$ and $Y \xrightarrow{\overline{i}_Y} I \multimap Y$, while the multiplication law is:



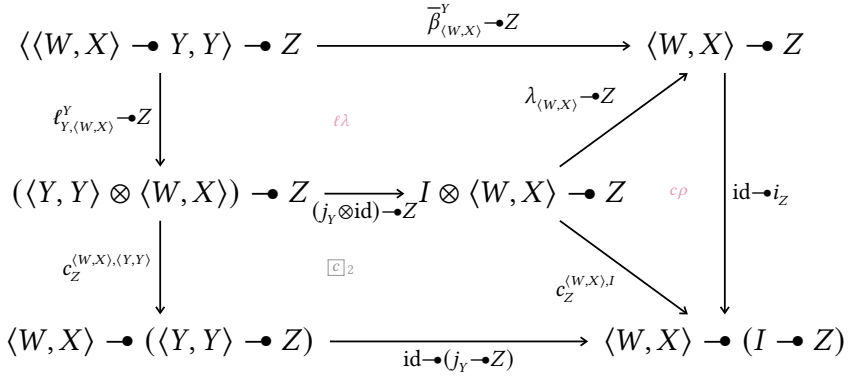
□

Lemma 4.2.1 *Clone composition can be given in terms of the clone powering and evaluation:*



$((\!|p\varepsilon|\!))$

PROOF Expanding the definitions in the diagram, we have



□

The main result of this section establishes an enriched form of the currying adjunction for the action given by functor evaluation, and hom given by the clone.

Theorem 4.2.1

For all $Y \in \mathcal{C}$, the functor $(\!|Y,-|\!) : \mathcal{D} \to \mathbf{Pow}(\mathcal{C},\mathcal{D})$ is right adjoint to the evaluation-at-Y functor $(-)Y : \mathbf{Pow}(\mathcal{C},\mathcal{D}) \to \mathcal{D}$.

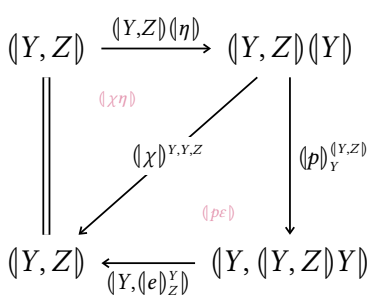
PROOF For all $Y \in \mathcal{C}$, we establish the adjunction

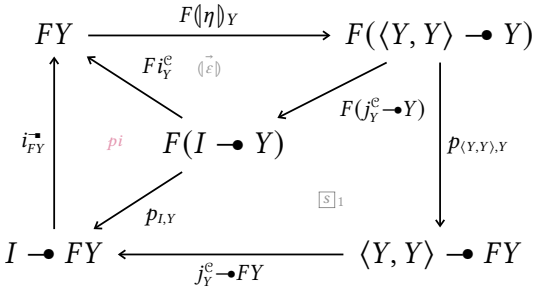
$$(-)Y \dashv (\!|Y,-|\!) : \mathbf{Pow}(\mathcal{C},\mathcal{D}) \to \mathcal{D} \qquad \omega : \mathcal{D}(FY,Z) \cong \mathbf{Pow}(\mathcal{C},\mathcal{D})(F,(\!|Y,Z|\!))$$

with the unit and counit for $F \in \mathbf{Pow}(\mathcal{C},\mathcal{D})$ and $Z \in \mathcal{D}$ defined as

$$(\!|\eta|\!)^{Y}_{F} : F \xrightarrow{F(\!|\eta|\!)} F(\!|Y|\!) \xrightarrow{(\!|p|\!)^{F}_{Y}} (\!|Y,FY|\!) \qquad (\!|\varepsilon|\!)^{Y}_{Z} : (\!|Y,Z|\!)Y \longrightarrow Z$$

(where the unit is a composite of two powered functor morphisms by Propositions 4.2.2 and 4.2.3), with zig-zag identities as follows:





The corresponding hom-set mappings between $f : FY \to Z$ and $\varphi : F \Longrightarrow (\!|Y,Z|\!)$ are:

$$\omega(f) : F \xrightarrow{F(\!|\eta|\!)} F(\!|Y|\!) \xrightarrow{(\!|p|\!)^{F}_{Y}} (\!|Y,FY|\!) \xrightarrow{(\!|Y,f|\!)} (\!|Y,Z|\!) \qquad \overline{\omega}(\varphi) : FY \xrightarrow{\varphi_Y} (\!|Y,Z|\!)Y \xrightarrow{(\!|\varepsilon|\!)^{Y}_{Z}} Z \quad \square$$

The endofunctor category $\mathbf{Pow}(\mathcal{C},\mathcal{C})$ is strict monoidal, and $\mathcal{C}$ is a canonical modular category with the action given by application: $F,X \mapsto F(X) : \mathbf{Pow}(\mathcal{C},\mathcal{C}) \times \mathcal{C} \to \mathcal{C}$. The proposition above therefore corresponds to equipping $\mathcal{C}$ with an enrichment over $\mathbf{Pow}(\mathcal{C},\mathcal{C})$, given by the clone functor $(\!|-,=|\!) : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathbf{Pow}(\mathcal{C},\mathcal{C})$, satisfying the κ-like adjunction $\mathcal{C}(FX,Y) \cong \mathbf{Pow}(\mathcal{C},\mathcal{C})(F,(\!|X,Y|\!))$. This will be used in the next section to induce monad maps from algebras, and in the familial model to construct metasubstitution from a free term algebra.

4.3 Powered monad morphisms

This section presents the equivalence of algebras for a monad, and monad maps into the endoclone of the algebra. The construction will allow us to extract the metasubstitution operation for any model of a syntactic algebra, and the final Theorem 4.3.2 will be key in the proof of the soundness of metasubstitution.

We fix a biclosed $\mathcal{V}$-modular category $\mathcal{C}$ and a powered monad $T\colon \mathcal{C} \to \mathcal{C}$. First, we establish some auxiliary properties about algebras for the postcomposition endofunctor.

Lemma 4.3.1 *The postcomposition endofunctor* $(T\circ)\colon \mathbf{Pow}(\mathcal{C},\mathcal{C}) \to \mathbf{Pow}(\mathcal{C},\mathcal{C})$ *is a monad.*

Proof Since T is a monoid in $\mathbf{Pow}(\mathcal{C},\mathcal{C})$, the monoid action $T \circ (-)$ on powered monads is itself a monad. We will denote its unit and multiplication η° and μ° respectively, but often conflate application of T with post-composition with T. □

The clone powering $(\!|p|\!)^T_{Y,Z}\colon T(\!|Y,Z|\!) \Longrightarrow (\!|Y,TZ|\!)$ is a collection of distributive laws (Section 3.1) for all $Y \in \mathcal{C}$ from the monad $T\colon \mathcal{C} \to \mathcal{C}$ to the monad $T\circ\colon \mathbf{Pow}(\mathcal{C},\mathcal{C}) \to \mathbf{Pow}(\mathcal{C},\mathcal{C})$.

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\quad T \quad} & \mathcal{C} \\ {\scriptstyle (\!|Y,-|\!)}\downarrow & \Rightarrow & \downarrow{\scriptstyle (\!|Y,-|\!)} \\ \mathbf{Pow}(\mathcal{C},\mathcal{C}) & \xrightarrow[T\circ(-)]{} & \mathbf{Pow}(\mathcal{C},\mathcal{C}) \end{array} \qquad (\!|p|\!)^T_{Y,-}\colon T \circ (\!|Y,-|\!) \Longrightarrow (\!|Y,T-|\!)$$

The corresponding lifting result (Prop. 3.1.2) states that $(\!|Y,-|\!)\colon \mathcal{C} \to \mathbf{Pow}(\mathcal{C},\mathcal{C})$ lifts to a functor $T\text{-}\mathbf{Alg}(\mathcal{C}) \to (T\circ)\text{-}\mathbf{Alg}(\mathbf{Pow}(\mathcal{C},\mathcal{C}))$, mapping a T-algebra $(A, a\colon TA \to A)$ to the composite

$$(\!|a|\!)\colon T(\!|Y,A|\!) \xrightarrow{(\!|p|\!)^T_{Y,A}} (\!|Y,TZ|\!) \xrightarrow{(\!|Y,a|\!)} (\!|Y,A|\!) \quad \in \mathbf{Pow}(\mathcal{C},\mathcal{C})$$

Lemma 4.3.2 *For a* T*-algebra* (A,a)*, the clone composition* $(\!|\chi|\!)^{X,Y,A}\colon (\!|Y,A|\!)(\!|X,Y|\!) \Longrightarrow (\!|X,A|\!)$ *is a* $(T\circ)$*-algebra homomorphism.*

$$\begin{array}{ccc} T(\!|Y,A|\!)(\!|X,Y|\!) & \xrightarrow{(\!|a|\!)(\!|X,Y|\!)} & (\!|Y,A|\!)(\!|X,Y|\!) \\ {\scriptstyle T(\!|\chi|\!)^{X,Y,A}}\downarrow & & \downarrow{\scriptstyle (\!|\chi|\!)^{X,Y,A}} \\ T(\!|X,A|\!) & \xrightarrow[(\!|a|\!)]{} & (\!|X,A|\!) \end{array}$$

Proof The condition expands to the following diagram:

$$\begin{array}{ccccc} T(\!|Y,A|\!)(\!|X,Y|\!) & \xrightarrow{(\!|p|\!)^T_{Y,A}(\!|X,Y|\!)} & (\!|Y,TZ|\!)(\!|X,Y|\!) & \xrightarrow{(\!|Y,a|\!)(\!|X,Y|\!)} & (\!|Y,A|\!)(\!|X,Y|\!) \\ {\scriptstyle T(\!|\chi|\!)^{X,Y,A}}\downarrow & (\!|p|\!) & \downarrow{\scriptstyle (\!|\chi|\!)^{X,Y,TZ}} & (\!|\chi|\!) & \downarrow{\scriptstyle (\!|\chi|\!)^{X,Y,A}} \\ T(\!|X,A|\!) & \xrightarrow[(\!|p|\!)^T_{X,A}]{} & (\!|X,TZ|\!) & \xrightarrow[(\!|X,a|\!)]{} & (\!|X,A|\!) \end{array}$$

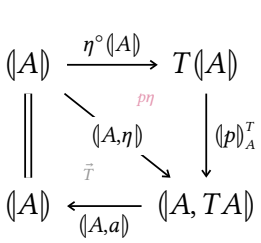
□

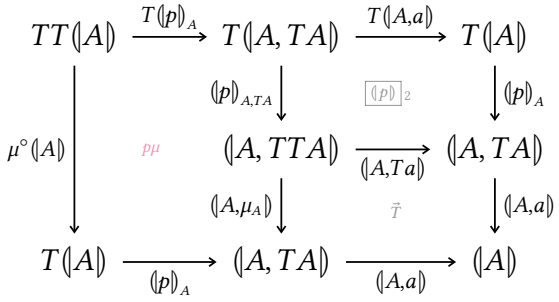
Proposition 4.3.1 *The lifting of a* T*-algebra* A *by* $(\!|A,-|\!)$ *is a* $T\circ$*-algebra in* $\mathbf{PowM}(\mathcal{C})$.

Proof We show that given an algebra $(A, a\colon TA \to A)$ for the monad T, $(\!|A|\!)$ is an algebra for the monad $T\circ\colon \mathbf{PowM}(\mathcal{C}) \to \mathbf{PowM}(\mathcal{C})$ with the structure map denoted

$$(\!|a|\!)\colon T(\!|A,A|\!) \xrightarrow{(\!|p|\!)^T_A} (\!|A,TA|\!) \xrightarrow{(\!|A,a|\!)} (\!|A,A|\!)$$

The unit and multiplication laws are as follows:





□

The following result is easy to establish but has valuable implications: any algebra for a monad induces a powered monad morphism into the endoclone. In the familial model, this will induce the meta-interpretation operation from an algebra structure for the term monad, which is equivalently a Σ-monoid – equipping a syntactic model with substitution structure is sufficient to induce the internal metasubstitution operation.

Theorem 4.3.1

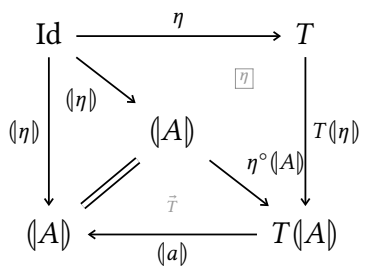
Algebras $TA \to A$ are in bijection with powered monad maps $T \Longrightarrow (\!|A|\!)$.

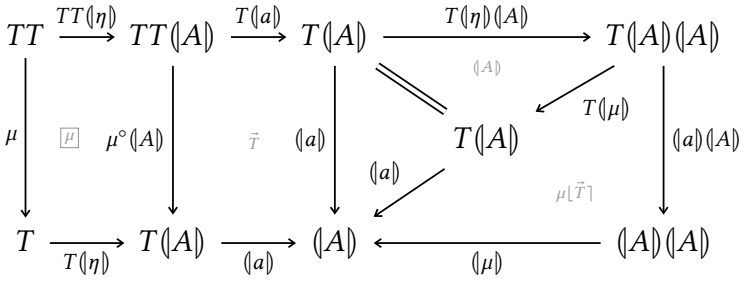
Proof The adjunction in Theorem 4.2.1 tells us that $\mathcal{C}(TA, A) \cong \mathbf{Pow}(\mathcal{C}, \mathcal{C})(T, (\!|A,A|\!))$. We now show that the bijection extends to monad algebras and monad morphisms.

Let $a\colon TA \to A$ be an algebra for the monad T. We show that the composite

$$\omega(a)\colon T \xrightarrow{T(\!|\eta|\!)} T(\!|A|\!) \xrightarrow{(\!|a|\!)} (\!|A|\!)$$

which we will denote $\omega^T_A(a)\colon T \Longrightarrow (\!|A|\!)$ is a monad morphism, i.e. it preserves the monad units and multiplications:

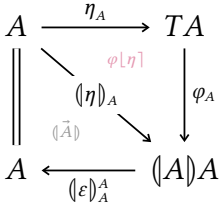


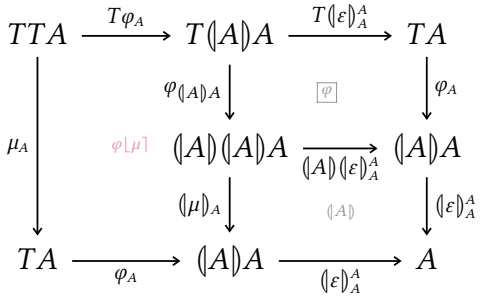


Conversely, let $\varphi\colon T \Longrightarrow (\!|A|\!)$ be a monad morphism. We show that the composite

$$\overline{\omega}(\varphi)\colon TA \xrightarrow{\varphi_A} (\!|A|\!)A \xrightarrow{(\!|\varepsilon|\!)^A_A} A$$

is an algebra for the monad T, i.e. it satisfies the following unit and multiplication laws:





□

Our study of clones and powered monads culminates in the following theorem, connecting algebras for powered monads with enriched Kleisli structures that encapsulate the notion of metasubstitution and meta-interpretation.

Theorem 4.3.2

Every powered monad gives rise to an enriched Kleisli triple, and every algebra for a powered monad induces an algebra for the corresponding enriched Kleisli triple.

Proof See the Appendix on page 318. □

To contextualise the results of the previous sections within the presheaf model (Section 2.2), consider a second-order signature Ω and a corresponding Ω-monoid $\mathcal{M}$ – a model of syntax supporting constructors, variables, and substitution. If $\mathbf{T}$ denotes the free Ω-monoid monad, then every Ω-monoid can be viewed as a $\mathbf{T}$-algebra via the comparison functor $\Omega\text{-}\mathbf{Mon} \to \mathbf{T}\text{-}\mathbf{Alg}$. When $\mathbf{T}$ is a powered monad, Theorem 4.3.2 shows that each Ω-monoid $\mathcal{M}$ supports a meta-interpretation operation $\langle \mathcal{P}, \mathcal{M} \rangle \to \langle \mathbf{T}\mathcal{P}, \mathcal{M} \rangle$: given an interpretation $\langle \mathcal{P}, \mathcal{M} \rangle$ of metavariables in the model, this operation lifts it to an interpretation of terms over $\mathcal{P}$.

To construct the powering map $\mathbf{T}(\mathcal{P} \multimap \mathcal{Q}) \to \mathcal{P} \multimap \mathbf{T}\mathcal{Q}$, we observe that it arises from a family of distributive laws between $\mathbf{T}$ and the functor $\mathcal{P} \multimap (-)$. Since $\mathbf{T}$ is a free monad, we can appeal to the results of Section 3.4: it suffices to find a lifting of $\mathcal{P} \multimap (-)$ to the category of Ω-monoids. Then, by Theorem 3.4.1, we obtain a distributive law $\mathbf{T}(\mathcal{P} \multimap (-)) \Longrightarrow \mathcal{P} \multimap \mathbf{T}(-)$. Furthermore, this construction is natural in $\mathcal{P}$ (Proposition 3.4.1), yielding a coherent family of distributive laws $p^{\mathbf{T}}_{\mathcal{P},\mathcal{Q}} \colon \mathbf{T}(\mathcal{P} \multimap (-)) \Longrightarrow \mathcal{P} \multimap (\mathbf{T}(-))$, compatible with the monad structure of $\mathbf{T}$. With the powering axioms established by more concrete means, we conclude that $\mathbf{T}$ is indeed a powered monad, justifying the meta-interpretation operation in every Ω-monoid.

The abstract treatment of metasubstitution and second-order syntax by Fiore and Hur (2008, 2010) and Fiore (2013) is formally equivalent to our approach, though it relies on cartesian strength rather than powering. Our formulation more naturally extends to the familial model (see Section 11.3.1). The equivalence of lifted signature endofunctor models between presheaves and families will be established in Section 7.2.2, while the equivalence of initial models – i.e. term syntax – is a consequence of the initial algebra lifting theorem in Section 3.3: if an endofunctor $F \colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ lifts to $G \colon \mathbf{PSh}_S \to \mathbf{PSh}_S$, then the initial F-algebra induces the initial G-algebra. When $G = \Omega + \mathcal{V} + \mathcal{P} \otimes (-)$, the initial G-algebra is $\mathbf{T}\mathcal{P}$. Applying the lifting theorem thus reduces to identifying an F that lifts to this G. This task – identifying appropriate functors that lift to Ω, $\mathcal{V}$, and $(\mathcal{P} \otimes)$ – is the focus of the next part of the thesis, which explores the skew-monoidal structure of substitution.

§ Summary of Part I

In this part of the dissertation, we developed key components of the mathematical theory supporting the familial model of abstract syntax. Central to this was the study of liftings to algebras and other categories, particularly in the context of transferring properties of initial algebras for syntactic signatures across forgetful functors.

We also introduced the theory of powered and enriched clone monads in biclosed monoidal categories and established a general method for equipping free monads with distributive laws. These results lay the groundwork for our later treatment of capture-permitting metasubstitution in second-order abstract syntax.

Part II focuses on the most significant divergence between the presheaf and familial models: their respective substitution structures.

PART II

SKEW CONSTRUCTIONS

The main advantage of the algebraic approach to second-order abstract syntax is a systematic treatment of variables and substitution structure: the substitution tensor product equips presheaves with a monoidal structure, monoids in which correspond to syntactic objects associated with a substitution operation. The categorical theory of the presheaf model relies on this monoidal structure being strong, which is implemented using a quotienting construction that represents a form of renaming-invariance. With the technical limitations of the familial model – namely, the lack of quotienting – the substitution structure cannot be translated directly. Reconciling these differences is an intriguing technical challenge, requiring the use and introduction of several categorical tools discussed next.

In the next three chapters we focus on *skew-monoidal closed categories*, the right notion of weak monoidality that the familial model of abstract syntax is set in. Chapter 5 gives the relevant definitions of skew-monoidal and skew-closed categories, modular categories, monoids and modules, also identifying the crucial notion of *parametrised maps* that play an important rule in abstract syntax. A limitation of skew-monoidal structure is that it doesn't transport to categories of algebras for monads we will be interested in: the tensor products of the underlying objects of algebras do not necessarily have an algebra structure themselves. We can nevertheless build a valuable theory of categories that functorially map into monoidal categories, and have objects that in some way represent the monoidal unit and tensor that are preserved by suitably generalised functors. This theory of *synthetic monoidal categories* is introduced in Chapter 6 and will be precisely the axiomatisation we need to analyse the familial model. Finally, Chapter 7 studies *warpings*, a structure relating a category $\mathcal{C}$ to a skew-monoidal category $\mathcal{V}$ which equips $\mathcal{C}$ with an skew-monoidal structure. We also introduce the adjoint notion of a closed warping, formally proving their equivalence and identifying an easily-satisfied sufficient condition for inducing such an adjoint warping.

CHAPTER 5

Skew-monoidal closed structure

This chapter introduces *skew-monoidal closed categories* in the abstract, and presents the main constructions that we will employ in the rest of the work. As a byproduct, focusing on a weaker monoidal structure helps us precisely systematise operations associated with monoidal and closed categories, elucidating discrepancies and unifying disparate definitions found in the literature in a consistent framework.

5.1 Skew categories

Monoidal closed categories (Eilenberg and Kelly, 1966; Mac Lane, 1971) are ubiquitous, but their structure may be too strict in some situations. In this section we introduce an elegant and wide-ranging technique to weakening monoidal structure and related constructions.

5.1.1 Skew-monoidal closed categories

Skew-monoidal categories, introduced by Szlachányi (2012) for the study of bialgebroids, offer a generalisation of monoidal categories better suited to settings where the traditional symmetry and invertibility assumptions do not hold. Since their introduction, they have emerged as a robust framework for redeveloping large portions of monoidal category theory in a weaker, yet richly structured, form – offering elegant dualities and encompassing natural examples beyond the scope of (op/lax) monoidal categories (Lack and Street, 2012[a],[b], 2014[a],[b], 2015). Working with directed structural transformations not only clarifies well-known results in monoidal and closed category theory but also highlights which aspects of structure are essential and which may be unnecessarily restrictive.

Following Street (2013) and Uustalu et al. (2020), we echo the sentiment that skew structure often provides a more appropriate setting for reasoning about monoidal and closed concepts: strong monoidal or closed categories frequently impose excess structure that can obscure rather than aid formalisation. In support of this perspective, we demonstrate how skew-monoidal closed categories preserve the key insights of familiar monoidal constructs while

avoiding some of the technical burdens – showing that weakening the structure not only generalises but also simplifies core categorical ideas.

Definition 5.1.1 A *skew monoidal category* $\mathcal{V}$ is equipped with a *skew unit* $I \in \mathcal{V}$ and a *skew tensor product* $(-) \otimes (=) \colon \mathcal{V} \times \mathcal{V} \to \mathcal{V}$, alongside natural transformations

$$\lambda_B \colon I \otimes B \to B \qquad \rho_A \colon A \to A \otimes I \qquad \alpha_{A,B,C} \colon (A \otimes B) \otimes C \to A \otimes (B \otimes C)$$

that satisfy the following axioms

$$\begin{array}{ccc} I & = & I \\ & {\scriptstyle \rho_I} \searrow \quad \nearrow {\scriptstyle \lambda_I} & \\ & I \otimes I & \end{array} \qquad (\lambda\rho)$$

$$\begin{array}{ccc} A \otimes B & = & A \otimes B \\ {\scriptstyle \rho_A \otimes B} \downarrow & & \uparrow {\scriptstyle A \otimes \lambda_B} \\ (A \otimes I) \otimes B & \xrightarrow[\alpha_{A,I,B}]{} & A \otimes (I \otimes B) \end{array} \qquad (\lambda\alpha\rho)$$

$$\begin{array}{ccc} (I \otimes A) \otimes B & \xrightarrow{\alpha_{I,A,B}} & I \otimes (A \otimes B) \\ & {\scriptstyle \lambda_A \otimes B} \searrow \quad \nearrow {\scriptstyle \lambda_{A \otimes B}} & \\ & A \otimes B & \end{array} \qquad (\lambda\alpha)$$

$$\begin{array}{ccc} & A \otimes B & \\ {\scriptstyle \rho_{A \otimes B}} \swarrow & & \searrow {\scriptstyle A \otimes \rho_B} \\ (A \otimes B) \otimes I & \xrightarrow[\alpha_{A,B,I}]{} & A \otimes (B \otimes I) \end{array} \qquad (\alpha\rho)$$

$$\begin{array}{ccccc} ((A \otimes B) \otimes C) \otimes D & & \xrightarrow{\alpha_{A,B,C} \otimes D} & & (A \otimes (B \otimes C)) \otimes D \\ {\scriptstyle \alpha_{A \otimes B,C,D}} \downarrow & & & & \downarrow {\scriptstyle \alpha_{A,B \otimes C,D}} \\ (A \otimes B) \otimes (C \otimes D) & \xrightarrow[\alpha_{A,B,C \otimes D}]{} & A \otimes (B \otimes (C \otimes D)) & \xleftarrow[A \otimes \alpha_{B,C,D}]{} & A \otimes ((B \otimes C) \otimes D) \end{array} \qquad (\alpha\alpha)$$

⌟

Example 5.1.1 (Fiore and Saville (2018)). For $\mathcal{C}$ cocartesian, the slice category $X/\mathcal{C}$ over $X \in \mathcal{C}$ is left skew-monoidal with unit $(X, \mathrm{id} \colon X \to X)$ and tensor

$$(X \xrightarrow{f} A) \oplus (X \xrightarrow{g} B) \triangleq (X \xrightarrow{f} A \xrightarrow{\iota_2^{A,B}} A + B)$$

since the morphism g cannot be recovered e.g. from the inverse of the left unitor

$$(X \xrightarrow{g} B) \mapsto (X \xrightarrow{\mathrm{id}} X \xrightarrow{\iota_2^{X,B}} X + B) \mapsto (X \xrightarrow{\iota_2^{X,B}} X + B \xrightarrow{[?,\mathrm{id}]} B)$$ ⌟

The following example predates the definition of a skew-monoidal category and provides a rich and abstract source of non-invertible structural transformations – indeed, our definition of skew substitution structure (Chapter 9) ultimately derives from a similar construction.

Example 5.1.2 (Altenkirch et al. (2010, Section 3.2)). For all functors $J \colon \mathcal{A} \to \mathcal{C}$, if the left Kan extension $\mathrm{Lan}_J \colon [\mathcal{A}, \mathcal{C}] \to [\mathcal{A}, \mathcal{C}]$ exists (see Section 8.1.2), the category $[\mathcal{A}, \mathcal{C}]$ is skew-monoidal with unit $J \in [\mathcal{A}, \mathcal{C}]$ and tensor product $F \otimes^J G \triangleq \mathrm{Lan}_J F \circ G$ with the structural transformations induced via the universal property of Kan extensions. ⌟

A running example throughout this part of the thesis will be the category of unsorted families of sets, indexed by $\mathbb{N}$ – the discrete version of the category of finite sets $\widetilde{\mathbb{F}}$ studied in the original presheaf model (Fiore et al., 1999), also discussed by Borthelle et al. (2020).

Example 5.1.3. The category $\widetilde{\mathbb{N}}$ of indexed sets is skew-monoidal, with unit $I_n \triangleq [n]$ (mapping $n \in \mathbb{N}$ to the finite set $\{0, \ldots, n-1\}$ of size n), tensor, unitors and associator defined as follows:

$$(X \oplus Y)(n) \triangleq \sum_{m \in \mathbb{N}} X_m \times (Y_m)^n$$

$$\lambda_Y \colon \sum_{m \in \mathbb{N}} [m] \times (Y_n)^m \to Y_n \qquad \rho_X \colon X_n \to \sum_{m \in \mathbb{N}} X_m \times [n]^m$$
$$(m, k \in [m], (y_i \in Y_n)_{i \le m}) \mapsto y_k \in Y_n \qquad x \in X_n \mapsto (n, x, (i)_{i \le n})$$

$$\alpha_{X,Y,Z} \colon \sum_{m \in \mathbb{N}} \Big(\sum_{k \in \mathbb{N}} X_k \times (Y_m)^k \Big) \times (Z_n)^m \to \sum_{p \in \mathbb{N}} X_p \times \Big(\sum_{q \in \mathbb{N}} Y_q \times (Z_n)^q \Big)^p$$
$$\big(m, (k, x \in X_k, (y_i \in Y_m)_{i \le k}), (z_j \in Z_n)_{j \le m}\big) \mapsto \big(k, x, (m, y_i, (z_j)_{j \le m})_{i \le k}\big)$$

The transformations are not invertible: for example, ρ_X^{-1} would require reindexing operation on an indexed set, and λ_Y^{-1}, while definable, is only a one-sided inverse:

$$y \in Y_n \mapsto (1, 0, (y)) \mapsto y \qquad (m, k, (y_i)_{i \le m}) \mapsto y_k \mapsto (1, 0, (y_k)) \qquad \lrcorner$$

Street (2013) developed the theory of skew-closed categories, which, in some ways, are better behaved than closed categories due to the cleaner definition without any non-diagrammatic axioms (Eilenberg and Kelly, 1966; Laplaza, 1977). Uustalu et al. (2020) gives a detailed analysis of the comparative structure of (skew) monoidal closed categories.

Definition 5.1.2 A *skew closed category* $\mathcal{V}$ is equipped with a *skew unit* $I \in \mathcal{V}$ and a *skew internal hom* $[-,=] \colon \mathcal{V}^{\mathrm{op}} \times \mathcal{V} \to \mathcal{V}$, alongside (extra)natural transformations

$$i_A \colon [I, A] \to A \qquad j_A \colon I \to [A, A] \qquad L^A_{B,C} \colon [B, C] \to [[A, B], [A, C]]$$

that satisfy the following coherence laws:

$$\begin{array}{ccc} I & = & I \\ & {\scriptstyle j_I} \searrow \quad \nearrow {\scriptstyle i_I} & \\ & [I, I] & \end{array} \qquad (ij)$$

$$\begin{array}{ccc} [A,C] & = & [A,C] \\ {\scriptstyle L^A_{A,C}} \downarrow & & \uparrow {\scriptstyle i_{[A,C]}} \\ [[A,A],[A,C]] & \xrightarrow[[j_A,C]]{} & [I,[A,C]] \end{array} \qquad (ijL)$$

$$\begin{array}{ccc} [B,C] & \xrightarrow{L^I_{B,C}} & [[I,B],[I,C]] \\ & {\scriptstyle [i_B,C]} \searrow \quad \swarrow {\scriptstyle [\mathrm{id},i_C]} & \\ & [[I,B],C] & \end{array} \qquad (iL)$$

$$\begin{array}{ccc} & I & \\ {\scriptstyle j_B} \swarrow & & \searrow {\scriptstyle j_{[A,B]}} \\ [B,B] & \xrightarrow[L^A_{B,B}]{} & [[A,B],[A,B]] \end{array} \qquad (Lj)$$

$$\begin{array}{ccccc} [C,D] & & \xrightarrow{\qquad L^B_{C,D} \qquad} & & [[B,C],[B,D]] \\ {\scriptstyle L^A_{C,D}} \downarrow & & & & \downarrow {\scriptstyle [\mathrm{id},L^A_{B,D}]} \\ [[A,C],[A,D]] & \xrightarrow[L^{[A,B]}_{[A,C],[A,D]}]{} & [[[A,B],[A,C]],[[A,B],[A,D]]] & \xrightarrow[[L^A_{B,C},\mathrm{id}]]{} & [[B,C],[[A,B],[A,D]]] \end{array} \qquad (LL)$$

$\lrcorner$

We have the expected notions of monoidal and closed functors between skew categories, with axioms directed in the appropriate way.

Definition 5.1.3 Given skew-monoidal categories $(\mathcal{V}, I, \otimes)$ and $(\mathcal{W}, J, \oplus)$, a *skew-monoidal functor* $F\colon \mathcal{V} \to \mathcal{W}$ comes with a unit morphism and multiplication natural transformation satisfying the unit and associativity axioms:

$$u\colon J \to FI \qquad m_{A,B}\colon FA \oplus FB \to F(A \otimes B)$$

$$\begin{array}{ccc} FA & \xrightarrow{F\rho^{\otimes}_{A}} & F(A \otimes I) \\ {\scriptstyle \rho^{\oplus}_{FA}}\downarrow & & \uparrow{\scriptstyle m_{A,I}} \\ FA \oplus J & \xrightarrow[FA \oplus u]{} & FA \oplus FI \end{array} \quad (mu) \qquad \begin{array}{ccc} J \oplus FB & \xrightarrow{u \oplus FB} & FI \oplus FB \\ {\scriptstyle \lambda^{\oplus}_{FB}}\downarrow & & \downarrow{\scriptstyle m_{I,B}} \\ FB & \xleftarrow[F\lambda^{\otimes}_{B}]{} & F(I \otimes B) \end{array} \quad (um)$$

$$\begin{array}{ccccc} (FA \oplus FB) \oplus FC & \xrightarrow{m_{A,B} \oplus \mathrm{id}} & F(A \otimes B) \oplus FC & \xrightarrow{m_{A \otimes B, C}} & F((A \otimes B) \otimes C) \\ {\scriptstyle \alpha^{\oplus}_{FA,FB,FC}}\downarrow & & & & \downarrow{\scriptstyle F\alpha^{\otimes}_{A,B,C}} \\ FA \oplus (FB \oplus FC) & \xrightarrow[\mathrm{id} \oplus m_{B,C}]{} & FA \oplus F(B \otimes C) & \xrightarrow[m_{A,B \otimes C}]{} & F(A \otimes (B \otimes C)) \end{array} \quad (mm)$$

A *skew-monoidal natural transformation* $\varphi\colon F \Longrightarrow G$ satisfies

$$\begin{array}{ccc} & J & \\ {\scriptstyle u^F}\swarrow & & \searrow{\scriptstyle u^G} \\ FI & \xrightarrow[\varphi_I]{} & GI \end{array} \quad (\varphi \lfloor u \rceil) \qquad \begin{array}{ccc} FA \oplus FB & \xrightarrow{\varphi_A \oplus \varphi_B} & GA \oplus GB \\ {\scriptstyle m^F_{A,B}}\downarrow & & \downarrow{\scriptstyle m^G_{A,B}} \\ F(A \otimes B) & \xrightarrow[\varphi_{A \otimes B}]{} & G(A \otimes B) \end{array} \quad (\varphi \lfloor m \rceil)$$

⌟

Example 5.1.4. The forgetful functor $\star\colon \widetilde{\mathbb{F}} \to \widetilde{\mathbb{N}}$, defined as $(\star P)_n \triangleq P[n]$ is skew-monoidal:

$$k \in I(n) \mapsto k \in \star(V)(n) = [n]$$
$$(m, x \in (\star P)_n, (y_i)_{i \le m} \in ((\star Q)_n)^m) \mapsto [(m, x, (i \in [m] \mapsto y_i))] \in (\star(P \otimes Q))_n = (P \otimes Q)[n]$$

where the tensor output is an equivalence class of tuples. ⌟

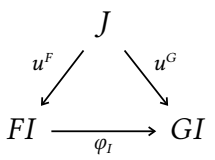
Definition 5.1.4 Given skew-closed categories $(\mathcal{V}, I, [-,=])$ and $(\mathcal{W}, J, [\![-,=]\!])$, a *skew-closed functor* $F\colon \mathcal{V} \to \mathcal{W}$ comes with a unit morphism and natural transformation satisfying:

$$u\colon J \to FI \qquad h_{A,B}\colon F[A,B] \to [\![FA, FB]\!]$$

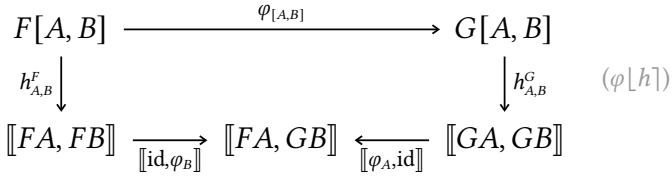
$$\begin{array}{ccc} F[I,B] & \xrightarrow{Fi^{\otimes}_{B}} & FB \\ {\scriptstyle h_{I,B}}\downarrow & & \uparrow{\scriptstyle i^{\oplus}_{FB}} \\ [\![FI, FB]\!] & \xrightarrow[[\![u, \mathrm{id}]\!]]{} & [\![J, FB]\!] \end{array} \quad (hu) \qquad \begin{array}{ccc} J & \xrightarrow{u} & FI \\ {\scriptstyle j^{\oplus}_{FA}}\downarrow & & \downarrow{\scriptstyle Fj^{\otimes}_{A}} \\ [\![FA, FA]\!] & \xleftarrow[h_{A,A}]{} & F[A,A] \end{array} \quad (uh)$$

$$\begin{array}{ccccc} F[B,C] & \xrightarrow{FL^{\otimes A}_{B,C}} & F[[A,B],[A,C]] & \xrightarrow{h_{[A,B],[A,C]}} & [\![F[A,B], F[A,C]]\!] \\ {\scriptstyle h_{B,C}}\downarrow & & & & \downarrow{\scriptstyle [\![\mathrm{id}, h_{A,C}]\!]} \\ [\![FB, FC]\!] & \xrightarrow[L^{\oplus FA}_{FB,FC}]{} & [\![[\![FA, FB]\!], [\![FA, FC]\!]]\!] & \xrightarrow[[\![h_{A,B}, \mathrm{id}]\!]]{} & [\![F[A,B], [\![FA, FC]\!]]\!] \end{array} \quad (hh)$$

A *skew-monoidal natural transformation* $\varphi : F \Longrightarrow G$ satisfies



$(\varphi\lfloor u \rceil)$



$(\varphi\lfloor h \rceil)$

The skew categorification of the set-theoretic concept of a module over a monoid corresponds to a category with a directed bifunctorial action of a skew-monoidal category. Keeping with the monoid→monoidal naming convention, we shall refer to these as *skew-modular categories*.

5.1.2 Skew-monoidal closed modular categories

Fix a skew-monoidal closed category $(\mathcal{V}, I, \otimes, [-,=])$, and denote the skew-monoidal and skew-closed structures with $\mathcal{V}^{\otimes}$ and $\mathcal{V}^{[]}$, respectively.

Definition 5.1.5 A *left/right $\mathcal{V}^{\otimes}$-modular category* $\mathcal{C}$ comes with a left/right $\mathcal{V}^{\otimes}$-action.

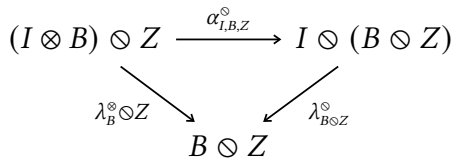
A *left $\mathcal{V}^{\otimes}$-action* is a bifunctor

$$(-) \oslash (=) : \mathcal{V} \times \mathcal{C} \to \mathcal{C}$$

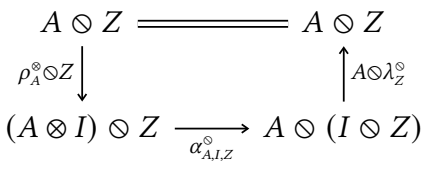
and structure transformations

$$\lambda^{\oslash}_Y : I \oslash Y \to Y$$
$$\alpha^{\oslash}_{A,B,Z} : (A \otimes B) \oslash Z \to A \oslash (B \oslash Z)$$

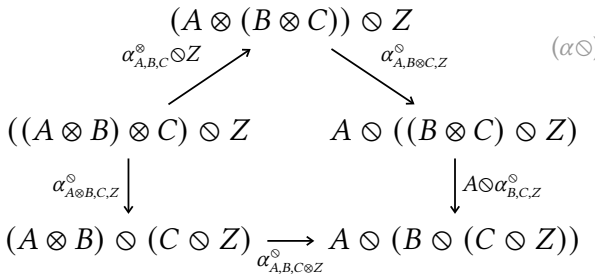
satisfying the laws



$(\lambda\oslash)$



$(\rho\oslash)$



$(\alpha\oslash)$

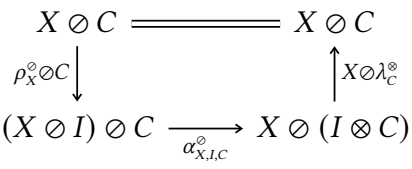
A *right $\mathcal{V}^{\otimes}$-action* is a bifunctor

$$(-) \oslash (=) : \mathcal{C} \times \mathcal{V} \to \mathcal{C}$$

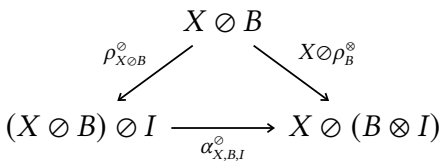
and structure transformations

$$\rho^{\oslash}_X : X \to X \oslash I$$
$$\alpha^{\oslash}_{X,B,C} : (X \oslash B) \oslash C \to X \oslash (B \otimes C)$$

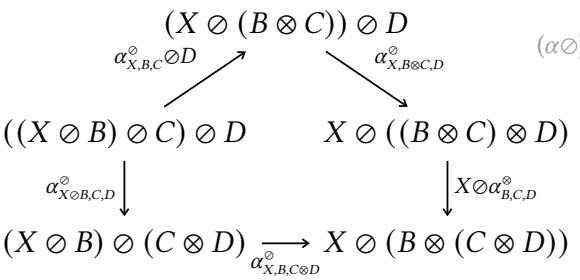
satisfying the laws



$(\lambda\oslash)$



$(\rho\oslash)$



$(\alpha\oslash)$

Definition 5.1.6 A *skew-monoidal $(\mathcal{V}, \mathcal{W})$-bimodular category* $\mathcal{C}$ has a left $\mathcal{V}$-action $\circledcirc$ and a right $\mathcal{W}$-action $\oslash$, with a coherent mixed associator (Capucci and Gavranović, 2022, Sec. 4.3).

$$\alpha_{A,Y,C} \colon (A \circledcirc Y) \oslash C \to A \circledcirc (Y \oslash C) \colon \mathcal{V} \times \mathcal{C} \times \mathcal{W} \to \mathcal{C}$$ ⌟

Modular categories often arise from changing or forgetting some of the structure of monoidal categories, but the action can be highly heterogeneous too.

Example 5.1.5. For any category $\mathcal{C}$, the strictly monoidal endofunctor category $[\mathcal{C}, \mathcal{C}]$ is a two-sided action on $\mathcal{C}$. For example, $\mathcal{C}$ presented as a left $[\mathcal{C}, \mathcal{C}]$-modular category has the action $\circledcirc \colon [\mathcal{C}, \mathcal{C}] \times \mathcal{C} \to \mathcal{C}$ defined by evaluation: $F \circledcirc X \triangleq FX$. The left unit $\mathrm{Id} \colon X \to X$ and associator $(G \circ F)X \to G(FX)$ are the identities. If $\mathcal{C}$ is a *skew bicategory* (Lack and Street, 2014[a]), the same construction gives rise to a skew action, without the invertibility conditions. ⌟

Example 5.1.6. Given two skew-monoidal categories $(\mathcal{V}, I, \otimes)$ and $(\mathcal{W}, J, \oplus)$ and a skew-monoidal functor $U \colon \mathcal{W} \to \mathcal{C}$, $\mathcal{W}$ acts on $\mathcal{V}$ with $A, Q \mapsto A \otimes UQ \colon \mathcal{V} \times \mathcal{W} \to \mathcal{V}$. ⌟

Example 5.1.7. The substitution tensor product $\oplus$ in sorted families is defined via the *substitution action* $(-) \ominus (=) \colon \mathbf{Fam} \times \mathbf{Fam}_S \to \mathbf{Fam}$ which, "retrospectively", is a right $\mathbf{Fam}_S$-action. ⌟

Example 5.1.8. Presheaves act on families via the mixed tensor action $(-) \circledcirc (=) \colon \widetilde{\mathbb{F}} \times \widetilde{\mathbb{N}} \to \widetilde{\mathbb{N}}$:

$$(P \circledcirc Y)(n) \triangleq \sum_{[m] \in \mathbb{F}} P[m] \times ([m] \Rightarrow Y_n)$$

Families also act on presheaves along $\star \colon \widetilde{\mathbb{F}} \to \widetilde{\mathbb{N}}$, using the action $X \oslash P \triangleq X \otimes \star P$. ⌟

It is well-known that (strong) left modular categories $(\mathcal{C}, \circledcirc \colon \mathcal{V} \times \mathcal{C} \to \mathcal{C})$ – also known as left actegories (Capucci and Gavranović, 2022) – are equivalently strong monoidal functors from $\mathcal{V}$ to the endofunctor category $[\mathcal{C}, \mathcal{C}]$ (Janelidze and Kelly, 2001). Then, a skew left modular category is equivalently an oplax skew-monoidal functor $F \colon \mathcal{V} \to [\mathcal{C}, \mathcal{C}]$, with directed unit $FI \to \mathrm{Id} \colon \mathcal{C} \to \mathcal{C}$ and multiplication $F(A \otimes B) \to FA \circ FB \colon \mathcal{C} \to \mathcal{C}$ satisfying the usual axioms. Writing $F(A)(X)$ as $A \circledcirc X$, we get exactly the structure defined above. Graded comonads (Fujii, 2016) are similarly axiomatised as oplax functors $\mathcal{V} \to [\mathcal{C}, \mathcal{C}]$, but usually for *strict* monoidal $\mathcal{V}$. We may wonder if right (skew) modular categories $(\mathcal{C}, \oslash \colon \mathcal{C} \times \mathcal{V} \to \mathcal{C})$ have a similar natural characterisation, other than as lax monoidal functors $\mathcal{V}^{\mathrm{rev}} \to [\mathcal{C}, \mathcal{C}]$; we shall return to this question in Section 5.2.1.

Before proceeding, it is natural to ask whether one can define modules over closed categories using only the structure provided by the transformations i, j, and L. Eilenberg and Kelly (1966, Chapter 5) introduce categories over a closed category, a notion extended to the skew setting by Street (2013); this corresponds to what we define below as a left $\mathcal{V}^{[]}$-modular category. Campbell (2018) presents an equivalent definition using the language of enrichment, as formalised in Section 4.1. A skew axiomatisation of powering, framed in terms of closed structure, naturally fills the gap to yield the expected definition of a right $\mathcal{V}^{[]}$-modular category.

Definition 5.1.7 A *left/right $\mathcal{V}^{[]}$-modular category* $\mathcal{C}$ comes with a left/right $\mathcal{V}^{[]}$-action.

A *left $\mathcal{V}^{[]}$-action* is a bifunctor

$$\langle -,= \rangle \colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$$

and structure transformations

$$j^{\langle\rangle}_X \colon I \to \langle X, X\rangle$$
$$L^{\langle\rangle X}_{Y,Z} \colon \langle Y, Z\rangle \to [\langle X, Y\rangle, \langle X, Z\rangle]$$

satisfying the laws

$$\begin{array}{ccc} & I & \\ {\scriptstyle j^{\langle\rangle}_Y}\swarrow & & \searrow{\scriptstyle j^{[]}_{\langle X,Y\rangle}} \\ \langle Y, Y\rangle & \xrightarrow[L^{\langle\rangle X}_{Y,Y}]{} & [\langle X, Y\rangle, \langle X, Y\rangle] \end{array} \qquad (j\langle\rangle)$$

$$\begin{array}{ccc} \langle X, Y\rangle & = & \langle X, Y\rangle \\ {\scriptstyle L^{\langle\rangle X}_{X,Y}}\downarrow & & \uparrow{\scriptstyle i^{[]}_{\langle X,Y\rangle}} \\ [\langle X, X\rangle, \langle X, Y\rangle] & \xrightarrow[[j^{\langle\rangle}_A, \mathrm{id}]]{} & [I, \langle X, Y\rangle] \end{array} \qquad (i\langle\rangle)$$

$$\begin{array}{ccc} \langle Y, Z\rangle & \xrightarrow{L^{\langle\rangle X}_{Y,Z}} & [\langle X, Y\rangle, \langle X, Z\rangle] \\ {\scriptstyle L^{\langle\rangle W}_{Y,Z}}\downarrow & & \downarrow{\scriptstyle [\mathrm{id}, L^{\langle\rangle W}_{X,Z}]} \\ [\langle W, Y\rangle, \langle W, Z\rangle] & & [\langle X, Y\rangle, [\langle W, X\rangle, \langle W, Z\rangle]] \\ {\scriptstyle L^{[]\langle W,X\rangle}_{\langle W,Y\rangle,\langle W,Z\rangle}}\searrow & & \nearrow{\scriptstyle [L^{\langle\rangle W}_{X,Y}, \mathrm{id}]} \\ & [[\langle W, X\rangle, \langle W, Y\rangle], [\langle W, X\rangle, \langle W, Z\rangle]] & \end{array} \qquad (L\langle\rangle)$$

A *right $\mathcal{V}^{[]}$-action* is a bifunctor

$$[-,= \rangle \colon \mathcal{V}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{C}$$

and structure transformations

$$i^{[]}_Y \colon [I, Y\rangle \to Y$$
$$L^{[]\rangle A}_{B,Z} \colon [B, Z\rangle \to [[A, B], [A, Z\rangle\rangle$$

satisfying the laws

$$\begin{array}{ccc} [A, X\rangle & = & [A, X\rangle \\ {\scriptstyle L^{[]\rangle A}_{A,X}}\downarrow & & \uparrow{\scriptstyle i^{[\rangle}_{[A,X\rangle}} \\ [[A, A], [A, X\rangle\rangle & \xrightarrow[[j^{[]}_A, \mathrm{id}\rangle]{} & [I, [A, X\rangle\rangle \end{array} \qquad (j[\rangle)$$

$$\begin{array}{ccc} [B, Y\rangle & \xrightarrow{L^{[]\rangle I}_{B,Y}} & [[I, B], [I, Y\rangle\rangle \\ & {\scriptstyle [i^{[]}_B, \mathrm{id}\rangle}\searrow \quad \swarrow{\scriptstyle [\mathrm{id}, i^{[\rangle}_Y\rangle}] & \\ & [[I, B], Y\rangle & \end{array} \qquad (i[\rangle)$$

$$\begin{array}{ccc} [C, Z\rangle & \xrightarrow{L^{[]\rangle B}_{C,Z}} & [[B, C], [B, Z\rangle\rangle \\ {\scriptstyle L^{[]\rangle A}_{C,Z}}\downarrow & & \downarrow{\scriptstyle [\mathrm{id}, L^{[]\rangle A}_{B,Z}\rangle} \\ [[A, C], [A, Z\rangle\rangle & & [[B, C], [[A, B], [A, Z\rangle\rangle\rangle \\ {\scriptstyle L^{[A,B]}_{[A,C],[A,Z\rangle}}\searrow & & \nearrow{\scriptstyle [L^{A}_{B,C}, \mathrm{id}\rangle} \\ & [[[A, B], [A, C]], [[A, B], [A, Z\rangle\rangle] & \end{array} \qquad (L[\rangle)$$

⌟

Definition 5.1.8 A *skew-closed $(\mathcal{V}, \mathcal{W})$-bimodular category* $\mathcal{C}$ has a left $\mathcal{V}$-action $\langle -,= \rangle$ and a right $\mathcal{W}$-action $[-,= \rangle$, together with a mixed compositor satisfying compatibility axioms:

$$L^{A}_{X,Y} \colon \langle X, Y\rangle \to \langle [A, X\rangle, [A, Y\rangle\rangle \colon \mathcal{W} \times \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$$

⌟

Example 5.1.9. The category of indexed sets $\widetilde{\mathbb{N}}$ is a left skew-closed action over the closed category of presheaves:

$$\langle X, Y\rangle[m] \triangleq \prod_{n \in \mathbb{N}} ([m] \Rightarrow X_n) \to Y_n$$

The presheaf action, given any function $f \colon [m] \to [m']$, maps $h \in \langle X, Y\rangle[m]$ to

$$n \in \mathbb{N}, s \colon [m'] \Rightarrow X_n \;\mapsto\; h_n\big(k \in [m] \mapsto s(f\,k)\big)$$

The transformation $j_X \colon V \to \langle X, X\rangle$ is the mapping $k \in [m] \mapsto n \in \mathbb{N}, s \colon [m] \Rightarrow X_n \mapsto s\,k$ and $L^{X}_{Y,Z} \colon \langle Y, Z\rangle \to [\langle X, Y\rangle, \langle X, Z\rangle]$ is the following, natural in l:

$$h \in \langle Y, Z\rangle[k] \;\mapsto\; l \in \mathbb{N}, f\colon [k] \Rightarrow \langle X, Y\rangle[l] \;\mapsto\; m \in \mathbb{N}, s\colon [l] \Rightarrow X_m$$
$$\mapsto\; h_m\big(i \in [k] \mapsto (f\,i)_m\,(j \in [l] \mapsto s\,j)\big)$$ ⌟

Functors between modular categories generalise strong functors with tensorial strength. Unfortunately, "strong" can be an ambiguous term, referring both to a functor with strength, or a functor that preserves some structure up to isomorphism. Ratkovic (2012) uses the term *very strong monad* for monads with isomorphic strengths; instead, we prefer to use a consistent terminology of modular functors, which can themselves be lax, oplax, strong, or strict.

Definition 5.1.9 A *lax left/right skew $\mathcal{V}^{\otimes}$-modular functor* between left/right skew $\mathcal{V}^{\otimes}$-modules $\mathcal{C}$ and $\mathcal{D}$ is a functor $F\colon \mathcal{C} \to \mathcal{D}$ with a left/right skew strength.

A *lax skew $\mathcal{V}^{\triangleleft}$-strength* is a natural transformation with compatibility laws:

$$s_{A,Y}\colon A \triangleleft FY \to F(A \otimes Y)\colon \mathcal{V} \times \mathcal{C} \to \mathcal{D}$$

$$\begin{array}{ccc} I \triangleleft FY & \xrightarrow{s_{I,Y}} & F(I \otimes Y) \\ & {\scriptstyle \lambda^{\triangleleft}_{FY}} \searrow \quad \swarrow {\scriptstyle F\lambda^{\otimes}_{Y}} & \\ & FY & \end{array} \qquad (s\lambda\otimes)$$

$$\begin{array}{ccc} A \triangleleft (B \triangleleft FZ) & & \\ {\scriptstyle \alpha^{\triangleleft}_{A,B,FZ}} \nearrow & & \searrow {\scriptstyle A \otimes s_{B,Z}} \\ ((A \otimes B) \triangleleft FZ) & & A \otimes F(B \otimes Z) \\ {\scriptstyle s_{A\otimes B,Z}} \downarrow & & \downarrow {\scriptstyle s_{A,B\otimes Z}} \\ F((A \otimes B) \otimes Z) & \xrightarrow[F\alpha^{\otimes}_{A,B,Z}]{} & F(A \otimes (B \otimes Z)) \end{array} \qquad (s\alpha\otimes)$$

A *lax skew $\mathcal{V}^{\triangleright}$-strength* is a natural transformation with compatibility laws:

$$s_{X,B}\colon FX \triangleright B \to F(X \otimes B)\colon \mathcal{C} \times \mathcal{V} \to \mathcal{D}$$

$$\begin{array}{ccc} & FX & \\ & {\scriptstyle \rho^{\triangleright}_{FX}} \swarrow \quad \searrow {\scriptstyle F\rho^{\otimes}_{X}} & \\ FX \triangleright I & \xrightarrow[s_{X,I}]{} & F(X \otimes I) \end{array} \qquad (s\rho\otimes)$$

$$\begin{array}{ccc} F(X \otimes B) \triangleright C & & \\ {\scriptstyle s_{X,B}\otimes C} \nearrow & & \searrow {\scriptstyle s_{X\otimes B,C}} \\ (FX \triangleright B) \triangleright C & & F((X \otimes B) \otimes C) \\ {\scriptstyle \alpha^{\triangleright}_{FX,B,C}} \downarrow & & \downarrow {\scriptstyle F\alpha^{\otimes}_{X,B,C}} \\ FX \triangleright (B \otimes C) & \xrightarrow[s_{X,B\otimes C}]{} & F(X \otimes (B \otimes C)) \end{array} \qquad (s\alpha\otimes)$$

⌟

Definition 5.1.10 A *lax $(\mathcal{V}, \mathcal{W})$-modular functor* between bimodules $\mathcal{C}$ and $\mathcal{D}$ is equipped with

$$s^{\triangleleft}_{A,Y}\colon A \triangleleft FY \to F(A \otimes Y) \qquad s^{\triangleright}_{X,A}\colon FX \triangleright A \to F(X \otimes A)$$

satisfying two unit laws and the associativity law

$$\begin{array}{ccccc} (A \triangleleft FY) \triangleright B & \xrightarrow{s^{\triangleleft}_{A,Y}\otimes B} & F(A \otimes Y) \triangleright B & \xrightarrow{s^{\triangleright}_{A\otimes Y,B}} & F((A \otimes Y) \otimes B) \\ {\scriptstyle \alpha^{\mathcal{D}}_{A,FY,B}} \downarrow & & & & \downarrow {\scriptstyle F\alpha^{e}_{A,Y,B}} \\ A \triangleleft (FY \triangleright B) & \xrightarrow[A\otimes s^{\triangleright}_{Y,B}]{} & A \triangleleft F(Y \otimes B) & \xrightarrow[s^{\triangleleft}_{A,Y\otimes B}]{} & F(A \otimes (Y \otimes B)) \end{array} \qquad (s\alpha\otimes\otimes)$$

⌟

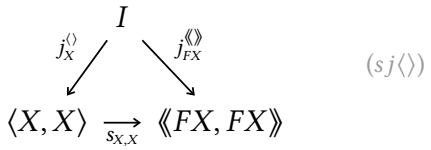
As usual, a modular functor is *oplax* if it is between the opposite categories (so s and the axioms are reversed), *strong* if s is an isomorphism, and *strict* if s is the identity. Natural transformations between modular functors preserve the strength in the obvious way.

The appropriate notion of functors between closed modular categories coincides with enrichment- and power-preserving functors, with axioms expressed via the closed structure.

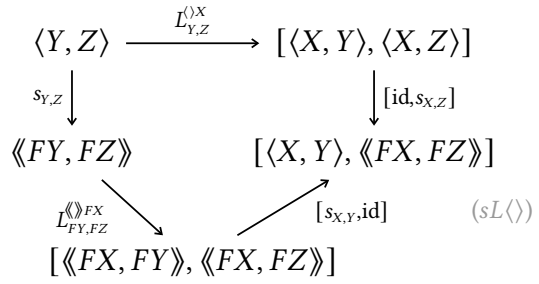
Definition 5.1.11 A *lax left/right skew $\mathcal{V}^{[]}$-modular functor* between left/right skew-closed $\mathcal{V}$-modular $\mathcal{C}$ and $\mathcal{D}$ is a functor $F\colon \mathcal{C} \to \mathcal{D}$ with a left/right skew strength.

A *lax skew $\mathcal{V}^{\langle\rangle}$-strength* is a natural transformation with compatibility laws:

$$s_{X,Y}\colon \langle X, Y\rangle \to \langle\!\langle FX, FY\rangle\!\rangle\colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$$



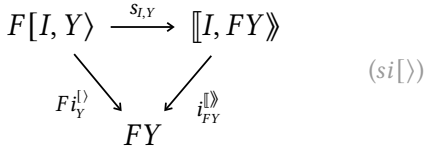
$(sj\langle\rangle)$



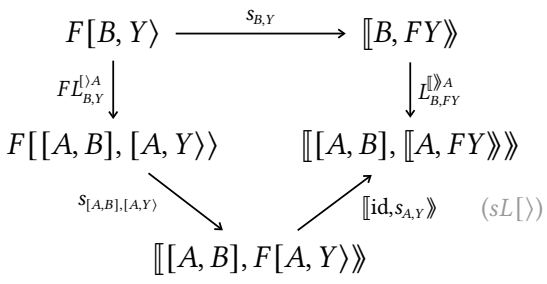
$(sL\langle\rangle)$

A *lax skew $\mathcal{V}^{[\rangle}$-strength* is a natural transformation with compatibility laws:

$$s_{A,Y}\colon F[A, Y\rangle \to [\![A, FY]\!\rangle\colon \mathcal{V}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$$



$(si[\rangle)$



$(sL[\rangle)$

⌟

Example 5.1.10. These notions of functorial strength have appeared throughout the literature, often under different names and assumptions, and typically shown to be equivalent under suitable structural conditions. The resulting terminological inconsistency – encompassing terms such as *left/right strength*, *costrength*, *cotensorial strength*, *opstrength*, *strong functor*, *enriched functor*, and *powered functor* – motivates the unified naming scheme adopted here. By casting these transformations as instances of natural structure between modular categories, we clarify the distinctions and structural requirements involved.

A. Kock (1970[a], 1972) constructs the canonical right transformation (left $\mathcal{V}^{\otimes}$-strength) $A \otimes FB \to F(A \otimes B)$ and canonical left transformation (right $\mathcal{V}^{\otimes}$-strength) $FA \otimes B \to F(A \otimes B)$ from a strong endofunctor (left $\mathcal{V}^{[]}$-strength) $[A, B] \to [FA, FB]$, extending the notions and equivalences to (commutative) monads. A. Kock (1971[b]) gives the equivalence between functorial strength and cotensorial strength (right $\mathcal{V}^{[]}$-strength) $F[A, B] \to [A, FB]$. ⌟

It may not be surprising that many of the structures introduced above are equivalent – an important fact that enables seamless translation between skew-monoidal and skew-closed settings as needed. Eilenberg and Kelly (1966) established that monoidal and closed categories are equivalent in the presence of an adjunction $(-) \otimes B \dashv [B, =]$ internalising to the isomorphism $[A \otimes B, C] \cong [A, [B, C]]$, a principle that also underlies the correspondence between enrichment and powering (McDermott and Uustalu, 2022). This equivalence was extended to the skew setting by Street (2013), who dropped the need for an internalised adjunction. Uustalu et al. (2020) offer a thorough exploration of this relationship in the skew context, and Campbell (2018) demonstrates the equivalence for skew modular categories via skew proactegories.

Definition 5.1.12 The 2-categories of left/right skew-monoidal/closed $\mathcal{V}$-modular categories, modular functors and natural transformations are denoted, respectively, as

$$\mathcal{V}^{\otimes}\textbf{-LMod} \qquad \mathcal{V}^{[]}\textbf{-RMod} \qquad \mathcal{V}^{\otimes}\textbf{-LMod} \qquad \mathcal{V}^{[]}\textbf{-RMod} \qquad \lrcorner$$

Theorem 5.1.1

We have the following implications:

$$\begin{aligned}
&\xi\colon (-)\otimes B \dashv [B,=] \ : \mathcal{V}\to\mathcal{V} &&\implies && \mathcal{V}^{\otimes} \cong \mathcal{V}^{[]}\\
&\kappa\colon (-)\oslash Y \dashv \langle Y,=\rangle \ : \mathcal{V}\to\mathcal{C} &&\implies && \mathcal{V}^{\otimes}\textbf{-LMod} \cong \mathcal{V}^{[]}\textbf{-LMod}\\
&\varkappa\colon (-)\oslash B \dashv [B,=\rangle \ : \mathcal{C}\to\mathcal{C} &&\implies && \mathcal{V}^{\otimes}\textbf{-RMod} \cong \mathcal{V}^{[]}\textbf{-RMod}
\end{aligned}$$

Proof sketch The hom-set natural isomorphisms arising from the adjunctions

$$\xi\colon \mathcal{V}(A\otimes B, C)\cong\mathcal{V}(A,[B,C]) \quad \kappa\colon \mathcal{C}(A\oslash Y, Z)\cong\mathcal{V}(A,\langle Y,Z\rangle) \quad \varkappa\colon \mathcal{C}(X\oslash B, Y)\cong\mathcal{C}(X,[B,Y\rangle)$$

are used to bijectively transpose the structural transformations and axioms from the monoidal to the closed setting. For the left unitors, the transposition is direct:

$$\frac{\lambda^{\otimes}_B\colon I\otimes B\to B}{j^{\otimes}_B\colon I\to[B,B]}\ (\xi) \qquad \frac{\lambda^{\langle\rangle}_Y\colon I\oslash Y\to Y}{j^{[\rangle}_Y\colon I\to\langle Y,Y\rangle}\ (\kappa)$$

For right unitors, it goes via the Yoneda embedding:

$$\begin{array}{ccc}
\mathcal{V}(A\otimes I,B) & \xrightarrow{\mathcal{V}(\rho^{\otimes}_A,B)} & \\
{\scriptstyle\xi}\downarrow & & \mathcal{V}(A,B)\\
\mathcal{V}(A,[I,B]) & \xrightarrow{\mathcal{V}(A,i^{\otimes}_B)} &
\end{array}
\qquad\qquad
\begin{array}{ccc}
\mathcal{C}(X\oslash I,Y) & \xrightarrow{\mathcal{C}(\rho^{\oslash}_X,Y)} & \\
{\scriptstyle\varkappa}\downarrow & & \mathcal{C}(X,Y)\\
\mathcal{C}(X,[I,Y\rangle) & \xrightarrow{\mathcal{C}(X,i^{[\rangle}_Y)} &
\end{array}$$

To connect the associators and compositors, we again apply the Yoneda embedding to transpose the associator to an internal currying operation, then show that the compositor L is its mate under some adjunction (see Kelly and Street (1974, Proposition 2.1)). For example, $\alpha_{X,B,C}\colon (X\oslash B)\oslash C\to X\oslash(B\otimes C)$ transposes by Yoneda to the currying $[A\otimes B,Y\rangle\to[A,[B,Y\rangle\rangle$, and taking its mate under

$$\begin{array}{ccc}
\mathcal{V} & \xrightarrow{[-,[B,Y\rangle\rangle} & \mathcal{C}^{\mathrm{op}}\\
{\scriptstyle(-)\otimes B}\ \circlearrowleft\ \dashv\ \circlearrowright\ {\scriptstyle[B,=]} & & \|\\
\mathcal{V} & \xrightarrow[[-,Y\rangle]{}{\qquad} & \mathcal{C}^{\mathrm{op}}
\end{array}$$

exhibits a bijection between $[A\otimes B,Y\rangle\to[A,[B,Y\rangle\rangle$ and $[B,Z\rangle\to[[A,B],[A,Z\rangle\rangle$. The other equivalences are as follows:

$$\frac{\alpha^{\otimes}_{A,B,C}\colon (A\otimes B)\otimes C\to A\otimes(B\otimes C)}{c^{\otimes}_{A,B,C}\colon [A\otimes B,C]\to[A,[B,C]]}\qquad \frac{\alpha^{\oslash}_{A,B,Z}\colon (A\otimes B)\oslash Z\to A\oslash(B\oslash Z)}{c^{\oslash}_{A,Y,Z}\colon \langle A\oslash Y,Z\rangle\to[A,\langle Y,Z\rangle]}$$

$$\frac{c^{\otimes}_{A,B,C}\colon [A\otimes B,C]\to[A,[B,C]]}{L^{\otimes A}_{B,C}\colon [B,C]\to[[A,B],[A,C]]}\qquad \frac{c^{\oslash}_{A,Y,Z}\colon \langle A\oslash Y,Z\rangle\to[A,\langle Y,Z\rangle]}{L^{\langle\rangle X}_{Y,Z}\colon \langle Y,Z\rangle\to[\langle X,Y\rangle,\langle X,Z\rangle]}$$

The axioms of skew-monoidal and skew-closed categories and left/right actions are shown to be equivalent – in most cases – by transposing the monoidal axiom to an equivalent one concerning the currying operator c, then using the expansion rules and transposition schemas to show that the c-axioms and L-axioms imply each other. We do not include the full derivations here, but see Theorem 7.1.3 for a detailed proof of a similar nature.

For functors between skew-monoidal/closed categories, and left/right actions thereof, the procedure is similar. The units $u\colon J \to FI$ of skew-monoidal and skew-closed functors are identical, while the multiplications $m_{A,B}\colon FA \oplus FB \to F(A \otimes B)$ and $h_{B,C}\colon F[B,C] \to [\![FB, FC]\!]$ are mates under the adjunction

$$\begin{array}{ccc} \mathcal{V} & \xrightarrow{\quad F \quad} & \mathcal{W} \\ (-)\otimes B \;\dashv\; [B,=] & & (-)\oplus FB \;\dashv\; [\![FB,=]\!] \\ \mathcal{V} & \xrightarrow[\quad F \quad]{} & \mathcal{W} \end{array}$$

Similarly, the strengths

$$\frac{s^{\otimes}_{A,Y}\colon A \mathbin{⦶} FY \to F(A \otimes Y)}{s^{\langle\rangle}_{X,Y}\colon \langle X, Y\rangle \to \langle\!\langle FX, FY\rangle\!\rangle} \quad \text{and} \quad \frac{s^{\oslash}_{X,B}\colon FX \mathbin{⊖} B \to F(X \oslash B)}{s^{[\rangle}_{A,Y}\colon F[A, Y\rangle \to [\![A, FY\rangle\!]}$$

are mates under

$$\begin{array}{ccc} \mathcal{V} & = & \mathcal{V} \\ (-)\otimes X \;\dashv\; \langle X,=\rangle & & (-)\mathbin{⦶} FX \;\dashv\; \langle\!\langle FX,=\rangle\!\rangle \\ \mathcal{C} & \xrightarrow[\quad F \quad]{} & \mathcal{D} \end{array} \qquad \begin{array}{ccc} \mathcal{C} & \xrightarrow{\quad F \quad} & \mathcal{D} \\ (-)\oslash A \;\dashv\; [A,=\rangle & & (-)\mathbin{⊖} A \;\dashv\; [\![A,=\rangle\!] \\ \mathcal{C} & \xrightarrow[\quad F \quad]{} & \mathcal{D} \end{array}$$

The correspondence between the respective axioms is also a series of transpositions. □

The following lemma is a useful tool for transposing adjoint functors across monoidal and closed actions, with the internal transposition as the intermediate step.

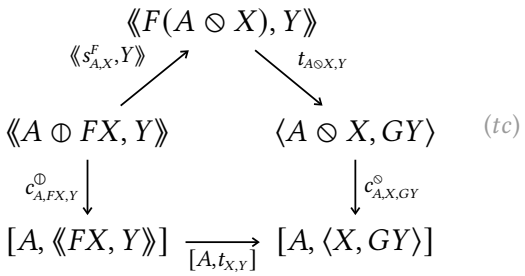
Lemma 5.1.1 *Given an adjunction $\tau\colon F \dashv G\colon \mathcal{C} \to \mathcal{D}$ between left skew-monoidal closed $\mathcal{V}$-modular categories, the following conditions are equivalent:*

1. *F is an oplax left skew $\mathcal{V}^{\otimes}$-modular functor with strength $s^{F}_{A,Y}\colon F(A \otimes Y) \to A \mathbin{⦶} FY$*
2. *G is a lax left skew $\mathcal{V}^{[\,]}$-functor with strength $s^{G}_{Y,Z}\colon \langle\!\langle Y, Z\rangle\!\rangle \to \langle GY, GZ\rangle$;*
3. *τ has a* lax internal transpose

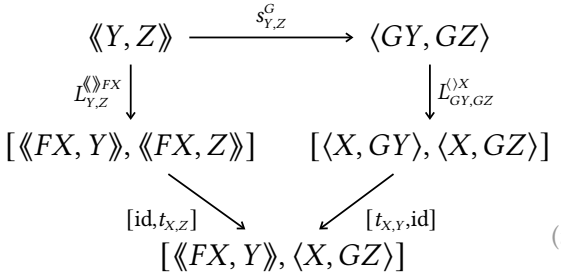
$$t_{X,Y}\colon \langle\!\langle FX, Y\rangle\!\rangle \to \langle X, GY\rangle\colon \mathcal{C} \times \mathcal{D} \to \mathcal{V}$$

that satisfies the unit and associativity axioms

$$\begin{array}{ccc} I & \xrightarrow{\;j^{\langle\rangle}_{X}\;} & \langle X, X\rangle \\ {\scriptstyle j^{\langle\!\langle\rangle\!\rangle}_{FX}}\downarrow & & \downarrow{\scriptstyle \langle X, \tau_X\rangle} \\ \langle\!\langle FX, FX\rangle\!\rangle & \xrightarrow[\;t_{X,FX}\;]{} & \langle X, GFX\rangle \end{array} \tag{tj}$$



(tc)



(tL)

Proof See the Appendix on page 321. □

Corollary 5.1.1 *A strong $\mathcal{V}^{\otimes}$-modular functor F with a right adjoint $\tau\colon F \dashv G\colon \mathcal{C} \to \mathcal{D}$ induces an isomorphism $\langle\!\langle FX, Y\rangle\!\rangle \cong \langle X, GY\rangle$.*

Proof The equivalences of $A \oplus FX \cong F(A \otimes X)$ and $\langle\!\langle FX, Y\rangle\!\rangle \cong \langle X, GY\rangle$ can be calculated via the Yoneda embedding.

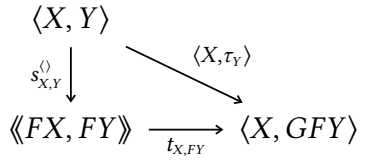
$$\begin{array}{ccc} \mathcal{D}\big(A \oplus FX, Y\big) & \xrightarrow[\cong]{\mathcal{D}(s_{A,X},Y)} & \mathcal{D}(F(A \otimes X), Y) \\ \Big\downarrow{\scriptstyle \kappa^{\mathcal{D}}} & & \Big\downarrow{\scriptstyle \tau} \\ & & \mathcal{C}(A \otimes X, GY) \\ & & \Big\downarrow{\scriptstyle \kappa^{\mathcal{C}}} \\ \mathcal{V}\big(A, \langle\!\langle FX, Y\rangle\!\rangle\big) & \xrightarrow[\mathcal{V}(A,t_{X,Y})]{\cong} & \mathcal{V}\big(A, \langle X, GY\rangle\big) \end{array}$$

□

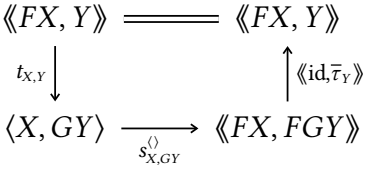
Example 5.1.11. The equivalence of α, c and L in Theorem 5.1.1 is an instance of Lemma 5.1.1, since $\alpha_{A,B,C}\colon (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ is an oplax $\mathcal{V}^{\otimes}$-strength for $F \triangleq (-) \otimes C$ and $L^{A}_{B,C}\colon [B, C] \to [[A, B], [A, C]]$ is a lax $\mathcal{V}^{[]}$-strength for $G \triangleq [A, -]$. The internal transpose $t_{A,B,C}\colon [A \otimes B, C] \to [A, [B, C]]$ is then the currying operator, and the associativity axioms of t reduce to the intermediate axioms of currying required to connect the associativity laws of α and L; see Eilenberg and Kelly (1966, Section II.2), axioms MCC3 and MCC3'. In a strong monoidal category, the associator is an isomorphism, and by Corollary 5.1.1, so is the currying transformation – this is familiar from Eilenberg and Kelly's (1966) proof of equivalence of monoidal categories, and closed categories with an internalised tensor-hom adjunction $[A \otimes B, C] \cong [A, [B, C]]$, and explains the discrepancy highlighted by Uustalu et al. (2020). ⌟

In the context of the corollary above, the directions of the isomorphism $s^{\otimes}_{A,X}\colon A \oplus FX \cong F(A \otimes X)$ give rise to two transformations: a $\mathcal{V}^{\langle\rangle}$-strength for F, and an internal transpose for F and G. The next lemma relates these two formally.

Lemma 5.1.2 *Given a strong $\mathcal{V}^{\otimes}$-modular functor F with right adjoint $\tau\colon F \dashv G\colon \mathcal{C} \to \mathcal{D}$, the induced $\mathcal{V}^{\langle\rangle}$-strength $s_{X,Y}\colon \langle X, Y\rangle \to \langle\!\langle FX, FX\rangle\!\rangle$ and internal transpose $t_{X,Y}\colon \langle\!\langle FX, Y\rangle\!\rangle \to \langle X, GY\rangle$ are compatible in the sense of the following diagrams:*



(τts)



(τ̄ts)

Proof See the Appendix on page 323. □

In the same situation, we can prove similar laws about the interaction between the induced inverse transpose $\langle X, GY\rangle \to \langle\!\langle FX, Y\rangle\!\rangle$ and the $\mathcal{V}^{\langle\rangle}$-strength $\langle\!\langle X, Y\rangle\!\rangle \to \langle GX, GY\rangle$.

Example 5.1.12. In a (strong) monoidal category, the functor $(-) \otimes C \colon \mathcal{V} \to \mathcal{V}$ is a strong modular functor with the associator $(A \otimes B) \otimes C \cong A \otimes (B \otimes C)$. By the above theorem, we have the induced isomorphism $[A \otimes B, C] \cong [A, [B, C]]$, and transformations $K^C_{A,B} \colon [A, B] \to [A \otimes C, B \otimes C]$ and $L^C_{A,B} \colon [A, B] \to [[C, A], [C, B]]$. The K transformation is the subject of Eilenberg and Kelly (1966, Section II.7), and the axioms above become the schemes that allow us to express K in terms of $\bar{t}$ and vice versa. In a skew setting, where α is not invertible, not all transformations are available at the same time: in a left-skew category, we have

$$(A \otimes B) \otimes C \to A \otimes (B \otimes C) \qquad [B, C] \to [[A, B], [A, C]] \qquad [A \otimes B, C] \to [A, [B, C]]$$

while in a right-skew category, we have

$$A \otimes (B \otimes C) \to (A \otimes B) \otimes C \qquad [A, B] \to [A \otimes C, B \otimes C] \qquad [A, [B, C]] \to [A \otimes B, C]$$

As Uustalu et al. (2020) show in Definition 4.5 and the corresponding footnote, there is no "elegant" way to define right-skew closed categories as L is not invertible – this is supported by the fact that the reversed transpose operation $\langle X, GY\rangle \to \langle\!\langle FX, Y\rangle\!\rangle$ does not fit into the framework of mates, so we do not get an inverse left strength $\langle GY, GZ\rangle \to \langle Y, Z\rangle$ that could be a right-skew variant of L. ⌟

The situation for right modules is somewhat simpler.

Lemma 5.1.3 *Assuming an adjunction $\pi \colon F \dashv G \colon \mathcal{C} \to \mathcal{D}$ between right skew-monoidal closed $\mathcal{V}$-modular categories, the following conditions are equivalent:*

1. *F is an oplax right $\mathcal{V}^{\otimes}$-modular functor with $s^{\oslash}_{X,A} \colon FX \ominus A \to F(X \oslash A)$*
2. *G is a lax right $\mathcal{V}^{[]}$-module morphism with $s^{[\rangle}_{A,X} \colon G[\![A, X]\!\rangle \to [A, GX\rangle$.*

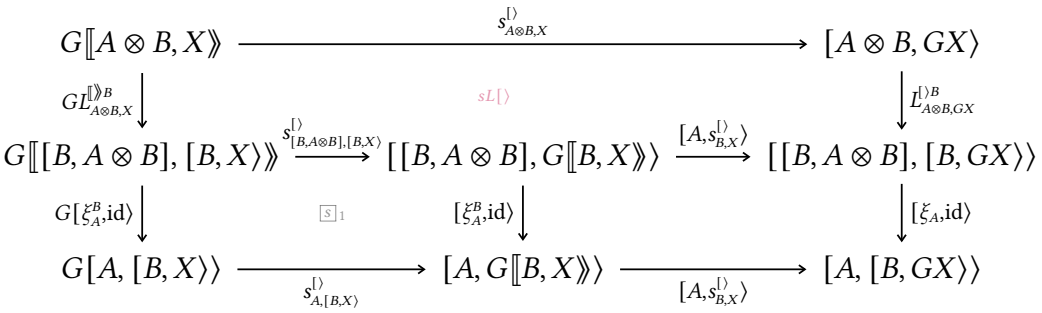
Proof The bijection between the strengths is given in the following diagram:

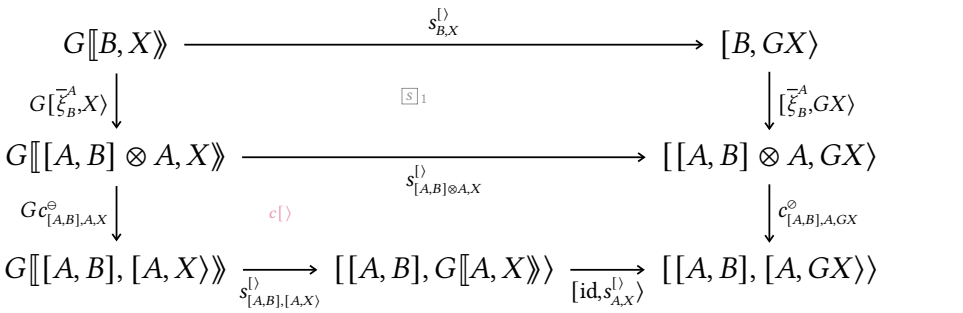
$$\begin{array}{ccc}
\mathcal{D}(FX \ominus B, Y) & \xrightarrow{\mathcal{D}(s^{\oslash}_{X,B}, Y)} & \mathcal{D}(F(X \oslash B), Y) \\
{\scriptstyle \zeta^{\mathcal{D}}}\big\downarrow & & \big\downarrow{\scriptstyle \tau} \\
\mathcal{D}(FX, [\![B, Y]\!\rangle) & & \mathcal{D}(X \oslash B, GY) \\
{\scriptstyle \tau}\big\downarrow & & \big\downarrow{\scriptstyle \zeta^{\mathcal{C}}} \\
\mathcal{C}(X, G[\![B, Y]\!\rangle) & \xrightarrow[\mathcal{C}(X, s^{[\rangle}_{B,Y})]{} & \mathcal{C}(X, [B, GY\rangle)
\end{array}$$

The unit diagrams are transposes of each other. The associativity law of $s^{\oslash}$ transposes to:

$$\begin{array}{ccccc}
G[\![A \otimes B, X]\!\rangle & & \xrightarrow{\quad s^{[\rangle}_{A\otimes B,X} \quad} & & [A \otimes B, GX\rangle \\
{\scriptstyle Gc^{\oslash}_{A,B,X}}\big\downarrow & & & & \big\downarrow{\scriptstyle c^{\ominus}_{A,B,GX}} \\
G[\![A, [\![B, X]\!\rangle]\!\rangle & \xrightarrow[s^{[\rangle}_{A,[B,X\rangle}]{} & [A, G[\![C, X]\!\rangle\rangle & \xrightarrow[[A, s^{[\rangle}_{B,X}\rangle]{} & [A, [B, GX\rangle\rangle
\end{array} \qquad (c[\,])$$

which is equivalent to the associativity law of $s^{[\rangle}$:





□

Example 5.1.13. The forgetful functor $\star\colon \widetilde{\mathbb{F}} \to \widetilde{\mathbb{N}}$ has a right adjoint cofree presheaf functor $\blacksquare\colon \widetilde{\mathbb{N}} \to \widetilde{\mathbb{F}}$, defined as

$$(\blacksquare X)[m] \triangleq \prod_{n\in\mathbb{N}} ([n]^m \Rightarrow X_n)$$

$\star$ is also a strong modular functor from the trivial $\widetilde{\mathbb{F}}$-module $(\widetilde{\mathbb{F}}, \otimes)$ to the left module $(\widetilde{\mathbb{N}}, \oslash)$ of Example 5.1.8: $\star(P \otimes Q) \cong P \oslash \star Q$. Thus, by above, we have the induced transformations $[P, Q] \to \langle \star P, \star Q\rangle$, $\langle X, Y\rangle \to [\blacksquare X, \blacksquare Y]$, and isomorphism $\langle \star P, Y\rangle \cong [P, \blacksquare Y]$. ⌟

Having defined skew-monoidal closed categories and modules, we can turn to the analysis of distinguished objects: monoids, and module objects over the monoid. In the theory of syntax, these will encapsulate the substitution structure of presheaves and families.

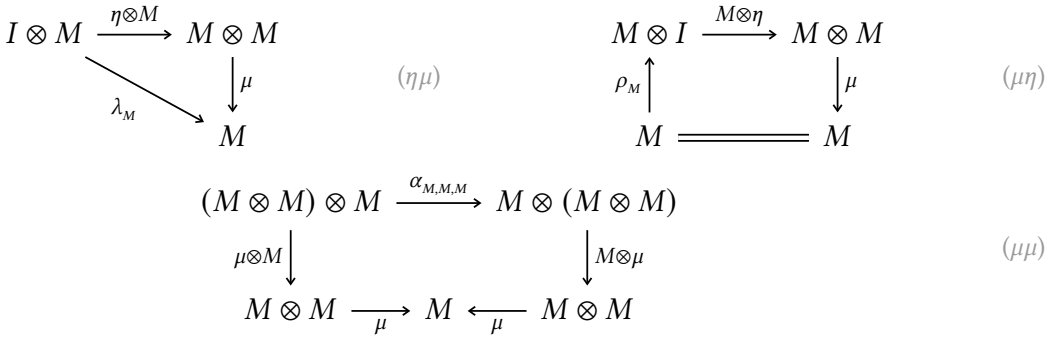
5.2 Monoids and modules

The notion of a monoid in a monoidal category extends naturally to a skew-monoidal setting; in fact, there is no need to call it "skew monoid", as the structure morphisms are identical to the monoidal case, and the laws follow the directionality of the skew structure in the obvious way. One natural variation we consider is the axiomatisation of monoids in a closed category, which will be more convenient to use in the context of initiality.

Definition 5.2.1 A *monoid* (M, η, μ) in a skew-monoidal category $\mathcal{V}$ is given by an object $M \in \mathcal{V}$ with the unit and multiplication maps

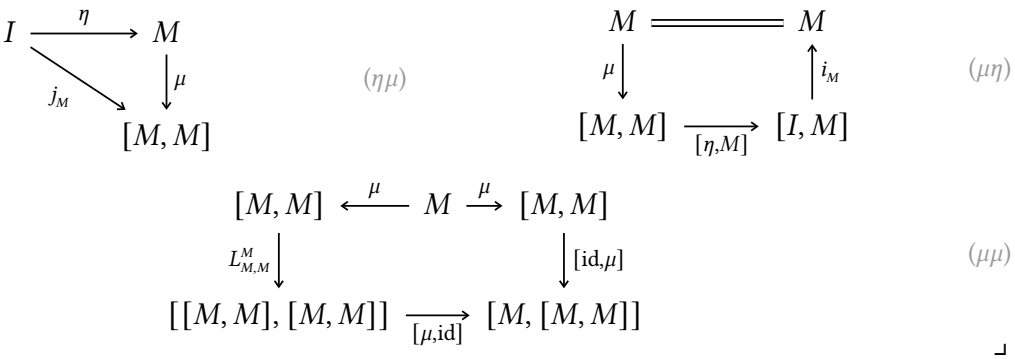
$$\eta\colon I \to M \qquad \mu\colon M \otimes M \to M$$

satisfying unit and associativity laws



The equivalent data and laws in a skew-closed category are maps and axioms

$$\eta : I \to M \qquad \mu : M \to [M, M]$$



⌟

Definition 5.2.2 A *monoid homomorphism* $f : (M, \eta^M, \mu^M) \to (N, \eta^N, \mu^N)$ is a morphism $f : M \to N \in \mathcal{V}$ that preserves the unit and multiplication:

For a skew-monoidal/closed category $\mathcal{V}$, we write **Mon**($\mathcal{V}$) for the *category of monoids* and monoid homomorphisms. ⌟

Example 5.2.1 (Altenkirch et al. (2010, Theorem 5)). Given the skew-monoidal category $([\mathcal{A}, \mathcal{C}], J, \otimes^J)$ of Example 5.1.2, a monoid $M \in [\mathcal{A}, \mathcal{C}]$ comes with unit natural transformation $\eta : J \Longrightarrow M$, and a multiplication $M \otimes^J M \Longrightarrow M$, which, with the calculus properties of left Kan extensions and copowers (Section 8.1.1), expands as follows:

$$\int_{B\in\mathcal{A}} \Big(\int^{A\in\mathcal{A}} \mathcal{C}(JA, MB) \cdot MA \Big) \to MB \;\cong\; \int_{A,B\in\mathcal{A}} \mathcal{C}(JA, MB) \to \mathcal{C}(MA, MB)$$

We recognise the RHS as a relative Kleisli extension, making M a *J-relative monad.* ⌟

Example 5.2.2. The endo-hom $[A, A]$ is a monoid for all objects A in a closed category, with unit $j_A : I \to [A, A]$ and multiplication is $L^A_{A,A} : [A, A] \to [[A, A], [A, A]]$. This generalises to the left $\mathcal{V}^{[]}$-endo-action $\langle X, X\rangle$ for any X in a left modular category $\mathcal{C}$. ⌟

Example 5.2.3 (Fiore et al. (1999, Proposition 3.4)). A monoid in the category $\widetilde{\mathbb{N}}$ is an *abstract clone*: a $\mathbb{N}$-indexed set M with unit as a set of distinguished elements $\eta_n \colon [n] \to M_n = \{u_i \in M_n \mid i \in [n]\}$ and multiplication as an indexed family of functions

$$\left\{m_{i,j} \colon M_i \times (M_j)^i \to M_j\right\}_{i,j\in\mathbb{N}}$$

satisfying unit and associativity axioms that correspond to monoid laws. ⌟

Proposition 5.2.1 *Monoidal/closed functors preserve monoids in skew categories.*

Proof Given a monoid $(M, \eta, \mu) \in \mathcal{V}$ and a monoidal/closed functor $(F \colon \mathcal{V} \to \mathcal{W}, u, m)$, FM is a monoid in $\mathcal{W}$ with unit $J \xrightarrow{u} FI \xrightarrow{F\eta} FM$ and multiplication maps

$$FM \oplus FM \xrightarrow{m_{M,M}} F(M \otimes M) \xrightarrow{F\mu} FM \qquad FM \xrightarrow{F\mu} F[M,M] \xrightarrow{m} [\![FM, FM]\!]$$

and monoid laws constructed from those of M and the monoidal functor laws. □

Corollary 5.2.1 *A monoidal functor $F \colon \mathcal{V} \to \mathcal{W}$ lifts to* $\mathbf{Mon}(\mathcal{V}) \to \mathbf{Mon}(\mathcal{W})$*, making* **Mon** *a functor* **SkMonCat** $\to$ **Cat** *from the category of skew-monoidal categories and monoidal functors.*

5.2.1 Modules

Generalising the concept of a monoid object in a monoidal category leads naturally to the notion of a module in a modular category. By the *microcosm principle* (Baez and Dolan, 1998, Section 2.2), modular categories serve as the canonical setting for defining module objects, just as monoids are most generally formulated within monoidal categories. For the remainder of this chapter, fix a skew-monoidal closed category $(\mathcal{V}, I, \otimes, [-,=])$, a left $\mathcal{V}$-modular category $(\mathcal{C}, \triangleright, \langle -,= \rangle)$ and a right $\mathcal{V}$-modular category $(\mathcal{D}, \triangleleft, [-,=\rangle)$.

Definition 5.2.3 Let $(M, \eta \colon I \to M, \mu \colon M \otimes M \to M)$ be a monoid in $\mathcal{V}$.

A *left M-module* in $\mathcal{C}$ is an object $Z \in \mathcal{C}$ with a *left action* over M, or *left M-action*

$$z \colon M \triangleright Z \to Z$$

that respects the unit and multiplication of the monoid:

$$\begin{array}{ccc} I \triangleright Z & \xrightarrow{\eta \triangleright Z} & M \triangleright Z \\ & {\scriptstyle \lambda_Z}\searrow \quad \swarrow{\scriptstyle z} & \\ & Z & \end{array} \qquad (\eta\triangleright)$$

$$\begin{array}{ccccc} (M \otimes M) \triangleright Z & \xrightarrow{\alpha_{M,M,Z}} & M \triangleright (M \triangleright Z) & & \\ {\scriptstyle \mu\triangleright M}\downarrow & & & \searrow{\scriptstyle M\triangleright z} & \\ M \triangleright Z & \xrightarrow[z]{} & Z & \xleftarrow[z]{} & M \triangleright Z \end{array} \qquad (\mu\triangleright)$$

A *right M-module* in $\mathcal{D}$ is an object $X \in \mathcal{D}$ with a *right action* over M, or *right M-action*

$$x \colon X \triangleleft M \to X$$

that respects the unit and multiplication of the monoid:

$$\begin{array}{ccc} X \triangleleft I & \xrightarrow{X\triangleleft\eta} & X \triangleleft M \\ {\scriptstyle \rho_X}\uparrow & & \downarrow{\scriptstyle x} \\ X & = & X \end{array} \qquad (\triangleleft\eta)$$

$$\begin{array}{ccccc} (X \triangleleft M) \triangleleft M & \xrightarrow{\alpha_{X,M,M}} & X \triangleleft (M \otimes M) & & \\ {\scriptstyle x\triangleleft M}\downarrow & & & \searrow{\scriptstyle X\triangleleft\mu} & \\ X \triangleleft M & \xrightarrow[x]{} & X & \xleftarrow[x]{} & X \triangleleft M \end{array} \qquad (\triangleleft\mu)$$

⌟

If $\mathcal{C}$ is a $(\mathcal{V}, \mathcal{W})$-bimodular category, the two notions can be combined into into a $\mathcal{C}$-object that can be acted on from both sides by two monoids $M \in \mathcal{V}$ and $N \in \mathcal{W}$ in a coherent way.

Definition 5.2.4 A *(M, N)-bimodule* (Y, y_l, y_r) has both a left M-action $y_l \colon M \otimes Y \to Y$ and a right N-action $y_r \colon Y \oslash N \to Y$ that are compatible with each other:

$$\begin{array}{ccccc} (M \otimes Y) \oslash N & & \xrightarrow{\alpha_{M,Y,N}} & & M \otimes (Y \oslash N) \\ {\scriptstyle y_l \oslash N}\downarrow & & & & \downarrow{\scriptstyle M \otimes y_r} \\ Y \oslash N & & \xrightarrow[y_r]{} & Y & \xleftarrow[y_l]{} & M \otimes Y \end{array} \tag{$\otimes\oslash$}$$

Definition 5.2.5 Let $(M, \eta \colon I \to M, \mu \colon M \to [M, M])$ be a monoid in $\mathcal{V}$.

A *left M-module* in $\mathcal{C}$ is an object $Z \in \mathcal{C}$ with a *left action* over M, or *left M-action*

$$z \colon M \to \langle Z, Z \rangle$$

that respects the unit and multiplication of the monoid:

$$\begin{array}{ccc} I & & \\ {\scriptstyle \eta}\swarrow & \searrow{\scriptstyle j_Z} & \\ M & \xrightarrow[z]{} & \langle Z, Z \rangle \end{array} \tag{$\langle\rangle\eta$}$$

$$\begin{array}{ccccc} \langle Z, Z \rangle & \xleftarrow{z} & M & \xrightarrow{\mu} & [M, M] \\ {\scriptstyle L^Z_{Z,Z}}\downarrow & & & & \downarrow{\scriptstyle \langle M, z \rangle} \\ [\langle Z, Z \rangle, \langle Z, Z \rangle] & & \xrightarrow[[z, \mathrm{id}]]{} & & [M, \langle Z, Z \rangle] \end{array} \tag{$\langle\rangle\mu$}$$

A *right M-module* in $\mathcal{D}$ is an object $X \in \mathcal{D}$ with a *right action* over M, or *right M-action*

$$x \colon X \to [M, X\rangle$$

that respects the unit and multiplication of the monoid:

$$\begin{array}{ccc} X & = & X \\ {\scriptstyle x}\downarrow & & \uparrow{\scriptstyle i_X} \\ [M, X\rangle & \xrightarrow[[\eta, x\rangle]{} & [I, X\rangle \end{array} \tag{$[]\eta$}$$

$$\begin{array}{ccccc} [M, X\rangle & \xleftarrow{x} & X & \xrightarrow{x} & [M, X\rangle \\ {\scriptstyle L^M_{X,X}}\downarrow & & & & \downarrow{\scriptstyle [M, x\rangle} \\ [[M, M], [M, X\rangle\rangle & & \xrightarrow[[\mu, \mathrm{id}\rangle]{} & & [M, [M, X\rangle\rangle \end{array} \tag{$[]\mu$}$$

Definition 5.2.6 A *(M, N)-bimodule* (Y, y_l, y_r) has both a left M-action $y_l \colon M \to \langle Y, Y \rangle$ and a right N-action $y_r \colon Y \to [N, Y\rangle$ that are compatible with each other:

$$\begin{array}{ccccc} \langle Y, Y \rangle & \xleftarrow{y_l} & M & \xrightarrow{y_l} & \langle Y, Y \rangle \\ {\scriptstyle L^N_{Y,Y}}\downarrow & & & & \downarrow{\scriptstyle \langle Y, y_r \rangle} \\ \langle [N, Y\rangle, [N, Y\rangle\rangle & & \xrightarrow[\langle y_r, \mathrm{id}\rangle]{} & & \langle Y, [N, Y\rangle\rangle \end{array} \tag{$\langle\rangle[]$}$$

Homomorphisms of modules are maps between the carrier objects that commute with the action in the obvious way, giving rise to a category of modules for a monoid. We write M-**LMod**$(\mathcal{C})$ for the category of left M-modules, N-**RMod**$(\mathcal{D})$ for the category of right N-modules, and (M, N)-**BMod**$(\mathcal{C})$ for bimodules. Of course, a trivial example of an M-bimodule is the monoid M itself, with the multiplication μ as either action, but the flexibility over the object and its category lets us state far more general properties.

Example 5.2.4. Every object X of a left $\mathcal{V}^{[]}$-modular category $\mathcal{C}$ is a module over its endo-hom $\langle X, X \rangle$, with the evaluation morphism $\overline{\kappa}_X \colon \langle X, X \rangle \otimes X \to X$.

Example 5.2.5. In a skew category, every object is canonically a left I-module, with actions $\lambda_Y : I \mathbin{⦸} Y \to Y$ and $j_Y : I \to \langle Y, Y\rangle$. Every right I-module is automatically an I-bimodule. ⌟

Example 5.2.6. Module objects in skew-closed modular categories are the appropriate notions of distinguished objects in enriched and powered categories (see Chapter 4): a $\mathcal{C}$-object X with a left N-action $N \to \langle X, X\rangle \in \mathcal{V}$ or right N-action $X \to [N, X\rangle \in \mathcal{C}$ compatible with the unit and composition or currying (which can be equivalently expressed in terms of L):

$$\begin{array}{ccc} N \otimes N & \xrightarrow{x \otimes x} & \langle X, X\rangle \otimes \langle X, X\rangle \\ {\scriptstyle \mu}\downarrow & & \downarrow{\scriptstyle N^X_{X,X}} \\ N & \xrightarrow[x]{} & \langle X, X\rangle \end{array} \qquad \begin{array}{ccc} [N, X\rangle & \xleftarrow{x} & X & \xrightarrow{x} & [N, X\rangle \\ {\scriptstyle [\mu, X\rangle}\downarrow & & & & \downarrow{\scriptstyle [N, x\rangle} \\ [N \otimes N, X\rangle & & \xrightarrow[c^{N,N}_X]{} & & [N, [N, X\rangle\rangle \end{array}$$

⌟

Example 5.2.7. Since $\langle Z, Z\rangle \in \mathcal{V}$ is a monoid for all $Z \in \mathcal{C}$, a left M-module in a skew-closed modular category $\mathcal{C}$ is equivalently an object $Z \in \mathcal{C}$ with a monoid morphism $z : N \to \langle Z, Z\rangle$. A categorification of this fact is precisely the equivalence of skew left $\mathcal{V}$-modular categories $(\mathcal{C}, \mathbin{⦸} : \mathcal{V} \times \mathcal{C} \to \mathcal{C})$ and oplax monoidal functors $\mathcal{V} \to [\mathcal{C}, \mathcal{C}]$: if $(\mathcal{V}, \hat{I} : \mathbf{1} \to \mathcal{V}, \otimes : \mathcal{V} \times \mathcal{V} \to \mathcal{V})$ is considered as a skew pseudomonoid in **Cat** (Lack and Street, 2012[b]), and the endofunctor category $([\mathcal{C}, \mathcal{C}], \mathrm{Id} : \mathbf{1} \to [\mathcal{C}, \mathcal{C}], \circ : [\mathcal{C}, \mathcal{C}] \times [\mathcal{C}, \mathcal{C}] \to [\mathcal{C}, \mathcal{C}])$ is a (pseudo)monoid for any $\mathcal{C}$, an oplax morphism of pseudomonoids $\mathbin{⦸} : \mathcal{V} \to [\mathcal{C}, \mathcal{C}]$ – or equivalently, a skew left pseudomodule over the closed pseudomonoid $\mathcal{V}$, or a module object in the **MonCat**-enriched **Cat** – is equipped with natural transformations

$$\begin{array}{ccc} & \mathbf{1} & \\ {\scriptstyle \hat{I}}\swarrow & & \searrow{\scriptstyle \mathrm{Id}} \\ \mathcal{V} & \xrightarrow[\mathbin{⦸}]{\lambda^{⦸}} & [\mathcal{C}, \mathcal{C}] \end{array} \qquad \begin{array}{ccc} \mathcal{V} \times \mathcal{V} & \xrightarrow{\mathbin{⦸} \times \mathbin{⦸}} & [\mathcal{C}, \mathcal{C}] \times [\mathcal{C}, \mathcal{C}] \\ {\scriptstyle \otimes}\downarrow & \nearrow_{\alpha^{⦸}} & \downarrow{\scriptstyle \circ} \\ \mathcal{V} & \xrightarrow[\mathbin{⦸}]{} & [\mathcal{C}, \mathcal{C}] \end{array}$$

that satisfy appropriate coherence conditions that reduce to axioms of a modular category.

This analysis also answers a question we posed earlier: if a skew left $\mathcal{V}$-modular category is equivalently an oplax monoidal functor $\mathcal{V} \to [\mathcal{C}, \mathcal{C}]$ which is a categorification of closed module object, a skew right $\mathcal{V}$-modular category $(\mathcal{C}, \oslash : \mathcal{D} \times \mathcal{V} \to \mathcal{D})$ is equivalently the categorification of a skew right module object $X \to [M, X\rangle$. Namely, defining the closed action $[-, =\rangle : \mathbf{MonCat}^{\mathrm{op}} \times \mathbf{Cat} \to \mathbf{Cat}$ by taking the functor category of the underlying categories $[\mathcal{V}, \mathcal{D}\rangle \triangleq [\underline{\mathcal{V}}, \mathcal{D}]$, we have that a right $\mathcal{V}$-modular category $(\mathcal{D}, \oslash : \mathcal{D} \times \mathcal{V} \to \mathcal{D})$ is equivalently a skew-closed right $\mathcal{V}$-pseudomodule in **Cat**, with a right $\mathcal{V}$-pseudoaction $\oslash : \mathcal{D} \to [\mathcal{V}, \mathcal{D}]$ equipped with coherent natural transformations

$$\begin{array}{ccc} \mathcal{D} & = & \mathcal{D} \\ {\scriptstyle \oslash}\downarrow & \Downarrow \rho^{\oslash} & \uparrow{\scriptstyle i} \\ [\mathcal{V}, \mathcal{D}] & \xrightarrow[[\hat{I}, \mathcal{D}]]{} & [\mathbf{1}, \mathcal{D}] \end{array} \qquad \begin{array}{ccccc} [\mathcal{V}, \mathcal{D}] & \xleftarrow{\oslash} & \mathcal{D} & \xrightarrow{\oslash} & [\mathcal{V}, \mathcal{D}] \\ {\scriptstyle [\otimes, \mathcal{D}]}\downarrow & & \swarrow_{\alpha^{\oslash}} & & \downarrow{\scriptstyle [\mathcal{V}, \oslash]} \\ [\mathcal{V} \times \mathcal{V}, \mathcal{D}] & & \xrightarrow[c]{} & & [\mathcal{V}, [\mathcal{V}, \mathcal{D}]] \end{array}$$

that have components $X \to i([\hat{I}, \mathcal{D}](\oslash X))$ and $[\mathcal{V}, \oslash](\oslash X)AB \to c([\otimes, \mathcal{D}](\oslash X))AB$, which simplify (with infix notation) to $\rho^{\oslash}_X : X \to X \oslash I$ and $\alpha^{\oslash}_{X,A,B} : (X \oslash A) \oslash B \to X \oslash (A \otimes B)$. What this tells us is that despite the usual presentation of left modular categories as monoidal

functors $\mathcal{V} \to [\mathcal{C}, \mathcal{C}]$, a more fundamental characterisation is as a left skew $\mathcal{V}$-pseudomodule (over the pseudomonoid $\mathcal{V} \in \mathbf{Cat}$), either in the monoidal form $\mathcal{V} \times \mathcal{C} \to \mathcal{C}$ or the closed form $\mathcal{V} \to [\mathcal{C}, \mathcal{C}]$, with right modular categories defined the other way around. The fact that the former definition aligns with the monoidal functor characterisation simply stems from the special case that the endo-action $\langle X, X\rangle$ in any left modular category is a monoid. ⌟

Example 5.2.8. In the presheaf model, the right unitor $P \otimes V \to P$ is a right V-module action. In $\widetilde{\mathbb{N}}$, the transformation $X \oplus I \to X$ isn't available as part of the structure, since it expresses the reindexing of the $\mathbb{N}$-indexed set:

$$(\sum_{m\in\mathbb{N}} X_m \times [n]^m) \to X_n = \{X_m \times [n]^m \to X_n\}_{m,n\in\mathbb{N}}$$

We can prove that a I-module structure on an indexed set is equivalent to a presheaf action: the free presheaf monad $\diamond X$ is isomorphic to $X \oplus I$, and module actions $X \oplus I \to X$ correspond to $\diamond$-algebra structure maps $\diamond X \to X$. Dually, the cofree presheaf comonad $\square X$ is isomorphic to $[I, X]$, and coalgebras $X \to \square X$ are module actions $X \to [I, X]$. ⌟

Right modules can be described categorically using the following standard result.

Proposition 5.2.2 *For a monoid M, the functor ${}^{\oslash}M \triangleq (-) \oslash M \colon \mathcal{C} \to \mathcal{C}$ is a monad, with unit and multiplication given by the following composites:*

$$X \xrightarrow{\rho_X} X \oslash I \xrightarrow{X\oslash\eta} X \oslash M \qquad (X \oslash M) \oslash M \xrightarrow{\alpha_{X,M,M}} X \oslash (M \otimes M) \xrightarrow{X\oslash\mu} X \oslash M$$

For a monoid M, the functor $M^{[\rangle} \triangleq [M, =\rangle \colon \mathcal{C} \to \mathcal{C}$ is a comonad, with counit and comultiplication

$$[M, Y\rangle \xrightarrow{[\eta,Y\rangle} [I, Y\rangle \xrightarrow{i_Y} Y \qquad [M, Y\rangle \xrightarrow{L^M_{M,Y}} [[M, M], [M, Y\rangle\rangle \xrightarrow{[\mu,\mathrm{id}\rangle} [M, [M, Y\rangle\rangle$$

The co/monad laws stem from axioms of skew modular categories and monoids, with all transformations directed in just the right way to make the appropriate diagrams commute. The axioms of modules are precisely the compatibility laws of co/algebras for the co/monads.

Proposition 5.2.3 *The categories of M-modules M-**RMod**$(\mathcal{C})$, algebras ${}^{\oslash}M$-**Alg**$(\mathcal{C})$, and coalgebras $M^{[\rangle}$-**Coalg**$(\mathcal{C})$ are isomorphic.*

The functor ${}^{\oslash}M \triangleq (-) \oslash M \colon \mathcal{C} \to \mathcal{C}$ then lifts to the free M-module functor $\mathcal{C} \to M$-**RMod**$(\mathcal{C})$, mapping objects $X \in \mathcal{C}$ to M-modules $(X \oslash M, (X \oslash M) \oslash M \to X \oslash M)$, and for all M-modules $(Y, y \colon Y \oslash M \to Y)$, we have a right M-module homomorphism ${}^{\oslash}M(Y) \to Y \in M$-**RMod**$(\mathcal{C})$ exhibited by Diagram ($\oslash\mu$).

Remark. $M^{\oslash} \triangleq M \oslash (-)$ is not a monad in a left-skew monoidal category due to the direction of the associator, so left modules are not expressible as algebras for a monad (but they are algebras for the endofunctor). The left action $M \to \langle X, X\rangle$ is not expressible as an algebra structure at all. Consequently, we will be working with right modules as default, specifying the direction in case of ambiguity. ⌟

We have a preservation property of module objects under modular functors, arising from a distributive law induced by the strength.

Proposition 5.2.4 *A right $\mathcal{V}$-modular functor $F\colon \mathcal{C} \to \mathcal{D}$ is an elevator from $(-) \oslash B\colon \mathcal{C} \to \mathcal{C}$ to $(-) \ominus B\colon \mathcal{D} \to \mathcal{D}$, for all $B \in \mathcal{V}$. When $B = M$ a monoid, this is a monad-monad distributive law.*

Proof For all $X \in \mathcal{C}$, the action $FX \ominus B \to F(X \oslash B)$ gives the components of the elevator. For $B = M$ a monoid, the distributive law axioms follow from strength and monoid axioms. □

Corollary 5.2.2 *Right $\mathcal{V}$-modular functors $F\colon \mathcal{C} \to \mathcal{D}$ lift to M-**RMod**$(\mathcal{C}) \to M$-**RMod**$(\mathcal{D})$, and M-**RMod** becomes a functor $\mathcal{V}$-**RMod** $\to$ **Cat**.*

Proof The distributive law of Proposition 5.2.4 induces the lifting to ${}^{\oslash}M$-algebras, or equivalently, M-modules. Given (X, x), FX is equipped with a module structure

$$FX \ominus M \xrightarrow{s_{X,M}} F(X \oslash M) \xrightarrow{Fx} FX \qquad FX \xrightarrow{Fx} F[M, X\rangle \xrightarrow{s_{M,X}} [\![M, FX]\!\rangle$$ □

Example 5.2.9. Let $\mathcal{C}$ be a $(\mathcal{V}, \mathcal{W})$-bimodular category, $M \in \mathcal{V}, N \in \mathcal{W}$ monoids, and $A \in \mathcal{V}$ an object. Then, the functor $A \oslash (-)\colon \mathcal{C} \to \mathcal{C}$ lifts to N-**RMod**$(\mathcal{C}) \to N$-**RMod**$(\mathcal{C})$, using the associator $(A \oslash Y) \oslash N \to A \oslash (Y \oslash N)$ as the distributive law. Similarly, $X \oslash (-)$ lifts to M-**LMod**$(\mathcal{V}) \to M$-**LMod**$(\mathcal{V})$, mapping $(A, a\colon A \otimes M \to A)$ to $(X \oslash A) \oslash M \to X \oslash (A \otimes M) \to X \oslash A$. ⌟

Corollary 5.2.3 *The action $s_{X,M}\colon FX \ominus M \to F(X \oslash M)$ of a right $\mathcal{V}$-modular functor $F\colon \mathcal{C} \to \mathcal{D}$ over a monoid M is a right M-module homomorphism.*

Proof The homomorphism property expands as follows:

$$\begin{array}{ccccccc}
(FX \ominus M) \ominus M & & \xrightarrow{\quad\alpha_{FX,M,M}\quad} & & FX \ominus (M \otimes M) & \xrightarrow{\mathrm{id} \ominus \mu} & FX \ominus M \\
\downarrow{\scriptstyle s_{X,M} \ominus M} & & s\alpha\oslash & & \downarrow{\scriptstyle s_{X,M\otimes M}} & \boxed{s}_2 & \downarrow{\scriptstyle s_{X,M}} \\
F(X \oslash M) \ominus M & \xrightarrow[s_{X\oslash M,M}]{} & F((X \oslash M) \oslash M) & \xrightarrow[F\alpha_{X,M,M}]{} & F(X \oslash (M \otimes M)) & \xrightarrow[F(X\oslash\mu)]{} & F(X \oslash M)
\end{array}$$

The proof is similar for the closed setting. □

Modules will form a very important part of our development, as presheaves are equivalently captured as co/algebras for the co/free presheaf co/monad, or modules for the monoidal unit. We turn to the discussion of modules and homomorphisms with additional structure, and how modular functors interact with this structure.

5.2.2 Parametrised maps

We investigate properties of *parametrised maps* $f\colon X \oslash B \to Y$, or $g\colon X \to [B, Y\rangle$ for $X, Y \in \mathcal{C}$ and $B \in \mathcal{V}$. These intend to represent morphisms from X to Y, while incorporating a parameter $B \in \mathcal{V}$. Such maps may interact with module structure in three ways, depending on where in the object $X \oslash B$ a monoid M acts.

Definition 5.2.7 Let $f\colon X \oslash B \to Y \in \mathcal{C}$ be a morphism in $\mathcal{C}$, and $M \in \mathcal{V}$ a monoid.

1. f is *left-linear* if $\mathcal{C}$ is a $\mathcal{V}$-bimodular category, $x\colon M \oslash X \to X$ and $y\colon M \oslash Y \to Y$ are left modules, and

$$\begin{array}{ccccc} (M \oslash X) \oslash B & \xrightarrow{\alpha_{M,X,B}} & M \oslash (X \oslash B) & \xrightarrow{M \oslash f} & M \oslash Y \\ {\scriptstyle x \oslash B}\downarrow & & & & \downarrow{\scriptstyle y} \\ X \oslash B & & \xrightarrow[f]{} & & Y \end{array} \tag{fL}$$

2. f is *middle-linear* if $x\colon X \oslash M \to X$ is a right module, $b\colon M \otimes B \to B$ is a left module, and

$$\begin{array}{ccccc} (X \oslash M) \oslash B & \xrightarrow{\alpha_{X,M,B}} & X \oslash (M \otimes B) & \xrightarrow{X \oslash b} & X \oslash B \\ {\scriptstyle x \oslash B}\downarrow & & & & \downarrow{\scriptstyle f} \\ X \oslash B & & \xrightarrow[f]{} & & Y \end{array} \tag{fM}$$

3. f is *right-linear* if $b\colon B \otimes M \to B$ and $y\colon Y \oslash M \to Y$ are right modules, and

$$\begin{array}{ccccc} (X \oslash B) \oslash M & \xrightarrow{\alpha_{X,B,M}} & X \oslash (B \otimes M) & \xrightarrow{X \oslash b} & X \oslash B \\ {\scriptstyle f \oslash M}\downarrow & & & & \downarrow{\scriptstyle f} \\ Y \oslash M & & \xrightarrow[y]{} & & Y \end{array} \tag{fR}$$

⌟

Example 5.2.10. The motivation for the terminology comes from abstract algebra: when R is a commutative ring and X, B, Y are R-modules, the diagrams correspond to the following properties of a function $f\colon X \times B \to Y$:

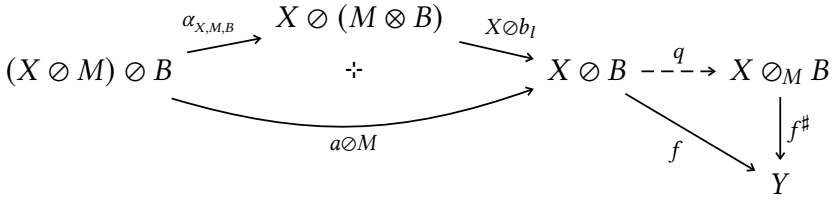
$$\begin{aligned} f(r \cdot_X x, b) &= r \cdot_Y f(x, b) \\ f(x \cdot_X r, b) &= f(x, r \cdot_B b) \\ f(x, b \cdot_B r) &= f(x, b) \cdot_Y r \end{aligned}$$

⌟

Right-linearity is just the notion of homomorphism from the M-module $(X \oslash B, (X \oslash b_l) \circ \alpha_{X,B,M})$ to the M-module (Y, y). Middle linearity is also known as *balancedness* (Jacobson, 2012, Section 3.7): postcomposition by a balanced map $f\colon X \oslash B \to Y$ equates the parallel pair:

$$\begin{array}{ccccc} & & X \oslash (M \otimes B) & & \\ & {\scriptstyle \alpha_{X,M,B}}\nearrow & & \searrow{\scriptstyle X \oslash b_l} & \\ (X \oslash M) \oslash B & & \dashv & & X \oslash B \\ & {\scriptstyle a \oslash M}\searrow & & \nearrow & \end{array}$$

By taking the coequaliser $X \oslash B \xrightarrow{q} Q$, we find the initial balanced map, through which all other balanced maps factor via q. The construction therefore allows us to define the *right action of a left M-module on a right M-module* $(-) \oslash_M (=)\colon M\text{-}\mathbf{RMod}(\mathcal{C}) \times M\text{-}\mathbf{LMod}(\mathcal{V}) \to \mathcal{C}$, given, for all $(X, x) \in M\text{-}\mathbf{RMod}(\mathcal{C})$ and $(B, a_r)\colon M\text{-}\mathbf{LMod}(\mathcal{V})$, as the coequaliser



The construction is a generalisation of the tensor product of R-modules for a commutative ring R (Mac Lane and Birkhoff, 1967, Section IX.8). If R is a commutative ring and (A, a), (B, b) are R-modules, their tensor product over R, $A \otimes_R B$, is defined as the tensor product of the underlying commutative group $A \otimes B$ quotiented by an invariance property with respect to the module actions:

$$A \otimes_R B \triangleq A \otimes B \;/\; (a \cdot_A r, b) \sim (x, r \cdot_B b)$$

For a commutative ring R, this tensor is itself an R-module, exhibiting the category $(R\text{-}\mathbf{Mod}([\rangle), R, \otimes_R)$ as monoidal. In case R is not commutative, the construction still works as long as A is a right R-module and B a left R-module, though the resulting object will not in general be a module. The universal property of the tensor product identifies every bilinear map $A \times B \to C$ as a linear map $A \otimes_R B \to C$. This balancedness property and its relation to quotienting is used to address the main discrepancy between the presheaf and familial models.

Example 5.2.11. In $\widetilde{\mathbb{F}}$, a transformation $f : P \otimes Q \to R$ is both natural and dinatural: it is equivalently a family of maps $\{f : P[m] \times ([m] \Rightarrow Q[n]) \to R[n]\}_{m,n \in \mathbb{F}}$ satisfying, for all $\rho : [m] \to [n]$ and $\varrho : [n] \to [p]$, the dinaturality and naturality conditions

$$f([m], t, \sigma \circ \rho) = f([n], P(\rho)t, \sigma) \qquad f([m], t, Q(\varrho) \circ \sigma) = R(\varrho)(f([m], t, \sigma))$$

In $\widetilde{\mathbb{N}}$, a map $f : X \oplus Y \to Z$ satisfies the same axioms exactly when all three objects are right I-modules, and the map is middle- and right-linear respectively. ⌟

Linearity properties compose with homomorphism conditions to give the following.

Proposition 5.2.5 *Let $f : X \oslash B \to Y \in \mathcal{C}$ be a parametrised map, and $i : X' \to X$, $j : Y \to Y' \in \mathcal{C}$ and $h : B' \to B \in \mathcal{V}$ morphisms. Then, the composite $X' \oslash B' \xrightarrow{i \oslash h} X \oslash B \xrightarrow{f} Y \xrightarrow{j} Y'$ is:*

1. *left-linear, if f is left-linear, $x' : M \odot X' \to X'$ and $y' : M \odot Y' \to Y'$ are left modules, and i, j are left module homomorphisms;*
2. *middle-linear, if f is middle-linear, $x' : X' \oslash M \to X'$ is a right module, $b' : M \otimes B' \to B'$ is a left module, i is a right module homomorphism, and b' is a left module homomorphism;*
3. *right-linear, if f is right-linear, $b' : B' \otimes M \to B'$ and $y' : Y' \oslash M \to Y'$ are right modules, and h, j are right module homomorphisms.*

The homomorphism properties give rise to the functors

$$\mathbf{LeftLin}(-,-;-) : M\text{-}\mathbf{LMod}(\mathcal{C})^{\mathrm{op}} \times \mathcal{V}^{\mathrm{op}} \times M\text{-}\mathbf{LMod}(\mathcal{C}) \to \mathbf{Set}$$
$$\mathbf{MiddleLin}(-,-;-) : M\text{-}\mathbf{RMod}(\mathcal{C})^{\mathrm{op}} \times M\text{-}\mathbf{LMod}(\mathcal{V})^{\mathrm{op}} \times \mathcal{C} \to \mathbf{Set}$$
$$\mathbf{RightLin}(-,-;-) : \mathcal{C}^{\mathrm{op}} \times M\text{-}\mathbf{RMod}(\mathcal{V})^{\mathrm{op}} \times M\text{-}\mathbf{RMod}(\mathcal{C}) \to \mathbf{Set}$$

that map objects to sets of linear maps of the right form.

Proposition 5.2.6 *If $X \oslash B \longrightarrow Y$ is a right/middle/left-linear map in $\mathcal{C}$ and $F\colon \mathcal{C} \longrightarrow \mathcal{D}$ is a modular functor, $FX \ominus B \xrightarrow{s^F_{X,B}} F(X \ominus B) \xrightarrow{Ff} FY$ is right/middle/left-linear in $\mathcal{D}$.*

Proof Assuming the appropriate structure on the objects and the functor, the linearity laws hold by the laws assumed for f, strength properties, and naturality.

$$\begin{array}{ccccccc}
(M \oplus FX) \ominus B & \xrightarrow{\alpha_{M,FX,B}} & M \oplus (FX \ominus B) & \xrightarrow{M \otimes s^{\oslash}_{X,B}} & M \oplus F(X \oslash_{\mathcal{C}} B) & \xrightarrow{M \otimes Ff} & M \oplus FY \\
{\scriptstyle s^{\otimes}_{M,X}}\downarrow & & s\alpha\otimes\oslash & & {\scriptstyle s^{\otimes}_{M,X\oslash B}}\downarrow & \boxed{s}_2 & \downarrow{\scriptstyle s^{\otimes}_{M,Y}} \\
F(M \otimes_{\mathcal{C}} X) \ominus B & \xrightarrow[s^{\oslash}_{M\otimes X,B}]{} & F((M \otimes_{\mathcal{C}} X) \oslash_{\mathcal{C}} B) & \xrightarrow{F\alpha_{M,X,B}} & F(M \otimes_{\mathcal{C}} (X \oslash_{\mathcal{C}} B)) & \xrightarrow{F(M\oslash f)} & F(M \otimes_{\mathcal{C}} Y) \\
{\scriptstyle Fx\oslash B}\downarrow & \boxed{s}_1 & \downarrow{\scriptstyle F(x\oslash B)} & & f\mathrm{L} & & \downarrow{\scriptstyle Fy} \\
FX \ominus B & \xrightarrow[s^{\oslash}_{X,B}]{} & F(X \oslash_{\mathcal{C}} B) & \xrightarrow[Ff]{} & & & FY
\end{array}$$

$$\begin{array}{ccccccc}
(FX \oslash M) \oslash B & \xrightarrow{\alpha_{FX,M,B}} & FX \oslash (M \otimes B) & \xrightarrow{\mathrm{id}\oslash Fb} & FX \oslash B \\
{\scriptstyle s_{X,M}\oslash B}\downarrow & s\alpha\oslash & \downarrow{\scriptstyle s_{X,M\otimes B}} & \boxed{s}_2 & \downarrow{\scriptstyle s_{X,B}} \\
F(X \otimes M) \oslash B & \xrightarrow{s_{X\otimes M,B}} F((X \otimes M) \oslash B) & \xrightarrow{F\alpha_{X,M,B}} F(X \otimes (M \otimes B)) & \xrightarrow{F(X\oslash b)} & F(X \otimes B) \\
{\scriptstyle Fx\oslash B}\downarrow & \boxed{s}_1 \quad {\scriptstyle F(x\oslash B)}\downarrow & f\mathrm{M} & & \downarrow{\scriptstyle Ff} \\
FX \otimes B & \xrightarrow[s_{X,B}]{} F(X \otimes B) & \xrightarrow[Ff]{} & & FY
\end{array}$$

$$\begin{array}{ccccccc}
(FX \oslash B) \oslash M & \xrightarrow{\alpha_{FX,B,M}} & FX \oslash (B \otimes M) & \xrightarrow{\mathrm{id}\oslash Fb} & FX \oslash B \\
{\scriptstyle s_{X,B}\oslash \mathrm{id}}\downarrow & s\alpha\oslash & \downarrow{\scriptstyle s_{X,B\otimes M}} & \boxed{s}_2 & \downarrow{\scriptstyle s_{X,B}} \\
F(X \otimes B) \oslash M & \xrightarrow{s_{X\otimes B,M}} F((X \otimes B) \oslash M) & \xrightarrow{F\alpha_{X,B,M}} F(X \otimes (B \otimes M)) & \xrightarrow{F(X\oslash b)} & F(X \otimes B) \\
{\scriptstyle Ff\oslash M}\downarrow & \boxed{s}_1 \quad \downarrow{\scriptstyle F(f\oslash M)} & f\mathrm{R} & & \downarrow{\scriptstyle Ff} \\
FY \otimes M & \xrightarrow[s_{C,M}]{} F(Y \otimes M) & \xrightarrow[Fy]{} & & FY
\end{array}$$

□

The definition above showcases the symmetry of the operations, but in practice we will use a more general definition of left-linearity, changing from $X \otimes (-)\colon \mathcal{C} \longrightarrow \mathcal{C}$ to an arbitrary module endofunctor F – this also rids us of the requirement that $\mathcal{C}$ is a modular category. Other names for the idea are *1-linearity* (A. Kock, 1971[a]), *left-linearity* (Fiore and Saville, 2018) or *left homomorphism* (Fiore and Saville, 2017).

Definition 5.2.8 If $F\colon \mathcal{C} \longrightarrow \mathcal{C}$ is a right $\mathcal{V}$-modular functor, and (X, x), (Y, y) are F-algebras, a parametrised map $X \oslash A \longrightarrow Y$ is *F-linear* if

$$\begin{array}{ccccc}
FX \oslash B & \xrightarrow{s_{X,B}} & F(X \oslash B) & \xrightarrow{Ff} & FY \\
{\scriptstyle x\oslash B}\downarrow & & & & \downarrow{\scriptstyle y} \\
X \oslash B & & \xrightarrow[f]{} & & Y
\end{array} \qquad (f\mathrm{F})$$

F-linearity is closed under homomorphisms (Fiore and Saville, 2021), so we have a functor

$$F\text{-}\mathbf{Lin}(-,-;-)\colon F\text{-}\mathbf{Alg}(\mathcal{C})^{\mathrm{op}} \times \mathcal{V}^{\mathrm{op}} \times F\text{-}\mathbf{Alg}(\mathcal{C}) \to \mathbf{Set}$$

that maps $(((X,x),B),(Y,y))$ to F-linear maps $X \oslash B \to Y$. ⌟

Definition 5.2.9 For a right $\mathcal{V}$-modular functor $F\colon \mathcal{C} \to \mathcal{C}$, an *$F$-module over M* is an object $X \in \mathcal{C}$ with F-algebra structure $x\colon FX \to X$ and an F-linear module structure $f\colon X \oslash M \to X$:

$$\begin{array}{ccccc} FX \oslash M & \xrightarrow{s_{X,M}} & F(X \oslash M) & \xrightarrow{Ff} & FX \\ {\scriptstyle x\oslash M}\downarrow & & & & \downarrow{\scriptstyle x} \\ X \oslash M & & \xrightarrow[f]{} & & X \end{array}$$

An *F-monoid* for $F\colon \mathcal{V} \to \mathcal{V}$ is a monoid M which is an F-module over M. We will generally talk about *algebraic modules* and *algebraic monoids* when the functor F is not named. ⌟

Example 5.2.12. For a right modular monad T, the strength $TX \oslash M \to T(X \oslash M)$ is T-linear:

$$\begin{array}{ccccc} TTX \oslash M & \xrightarrow{s_{TX,M}} & T(TX \oslash M) & \xrightarrow{Ts_{X,M}} & TT(X \oslash M) \\ {\scriptstyle \mu_X\oslash M}\downarrow & & & & \downarrow{\scriptstyle \mu_{X\oslash M}} \\ TX \oslash M & & \xrightarrow[s_{X,M}]{} & & T(X \oslash M) \end{array}$$

⌟

Example 5.2.13. Defining $(-) \oslash (=)\colon \mathcal{C} \times [\mathcal{C},\mathcal{C}] \to \mathcal{C}$ as $X \oslash G \triangleq GX$, an F-module over G is a natural transformation $\varphi\colon GF \Longrightarrow FG$ and object X with compatible algebra structures:

$$\begin{array}{ccccc} GFX & \xrightarrow{\varphi_X} & FGX & \xrightarrow{Fg} & FX \\ {\scriptstyle Gf}\downarrow & & & & \downarrow{\scriptstyle f} \\ GX & & \xrightarrow[g]{} & & X \end{array}$$

⌟

Example 5.2.14. In $\widetilde{\mathbb{F}}$ with substitution monoidal structure, for a signature endofunctor Ω, a Ω-monoid is a presheaf with compatible substitution and syntax structure. For the initial Ω-monoid, the compatibility law unfolds into the structurally recursive specification of the substitution operation. ⌟

The above definitions were presented in their most general form, in order to exhibit the symmetry of the definitions; in practice, we will work most often work with modules over the monoidal unit I.

5.2.3 Modules over the unit

The monoidal unit I is a monoid, with unit $\mathrm{id}_I\colon I \to I$ and multiplication $\lambda_I\colon I \otimes I \to I$. Modules over the unit feature prominently in our work, as they capture the renaming of presheaves as an extra algebraic structure on families. The general theory from the previous section also simplifies: every object of $\mathcal{V}$ is canonically a left I-module with action $\lambda_B\colon I \otimes B \to B$, and thus every bimodule is just a right module. In this section we discuss some derived definitions.

Pointed and invariant modules Modules over the unit may interact with compatible ways with monoids in $\mathcal{V}$. If the module resides in $\mathcal{V}$ itself, it may also have a point that is compatible with the module structure.

Definition 5.2.10 An *pointed module* (A, p, a) is an I-module (A, a) with a module homomorphism $p\colon I \to A$. Pointed modules are the objects in the slice category $I/I\text{-}\mathbf{RMod}(\mathcal{V})$.

$$\begin{array}{ccc} I \otimes I & \xrightarrow{p\otimes I} & A \otimes I \\ {\scriptstyle \lambda_I}\downarrow & & \downarrow{\scriptstyle a} \\ I & \xrightarrow[p]{} & A \end{array} \qquad (\otimes p)$$

⌟

Remark. The unit I is pointed, as is tensor of pointed objects $(A, p\colon I \to A)$ and $(B, q\colon I \to B)$:

$$I \xrightarrow{\rho_I} I \otimes I \xrightarrow{p\otimes q} A \otimes B$$

However, the tensor is *not* necessarily a pointed module – see discussion later. ⌟

Any M-module can be turned into an N-module by restricting along a monoid homomorphism $f\colon N \to M$:

$$X \oslash N \xrightarrow{X\oslash f} X \oslash M \xrightarrow{x} X$$

One such homomorphism is the unit $\eta\colon I \to N$, which is a monoid homomorphism; this can turn any N-module into an I-module. If a module has both structures present independently, we ask for them to be compatible.

Definition 5.2.11 A module $(X, m\colon X \otimes M \to X)$ is *invariant* if it is also an I-module with $a\colon X \otimes I \to X$, and the structures are compatible:

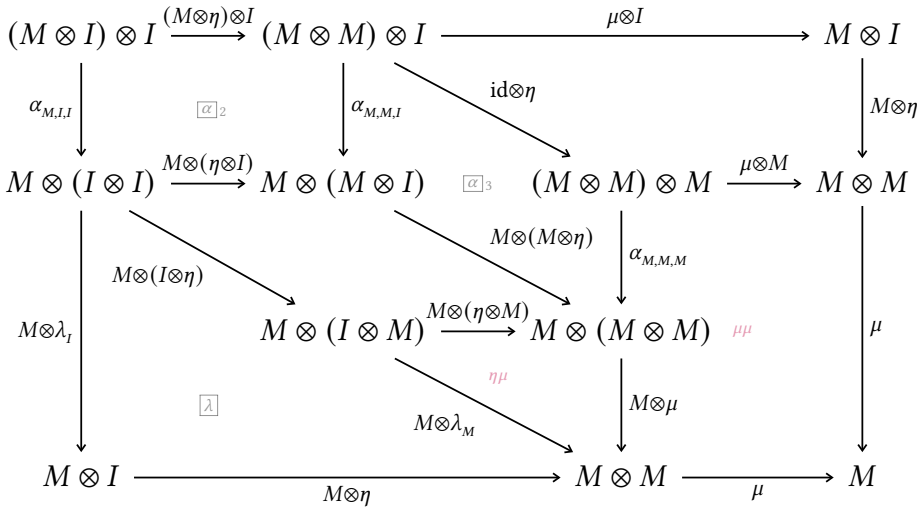
$$\begin{array}{ccc} X \oslash I & & \\ {\scriptstyle X\oslash\eta}\downarrow & \searrow^{a} & \\ X \oslash M & \xrightarrow[m]{} & X \end{array} \qquad (\oslash\eta)$$

⌟

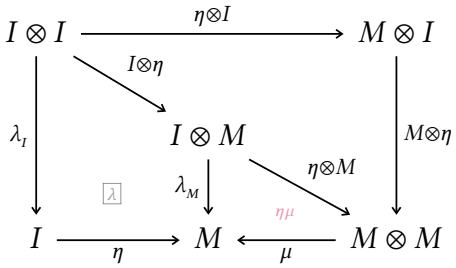
The notion of invariance is more valuable when X has both M-module and I-module structures a priori. In a strong monoidal modular category, every M-module $(X, m\colon X \oslash M \to X)$ has the canonical I-module action $\rho_X\colon X \oslash I \to X$ which are compatible by the right unit law of the modular category. In a skew-monoidal modular category, the right unitor $X \oslash I \to X$ does not exist by default, so if it is constructed by alternative means, the invariance condition may not necessarily hold.

Proposition 5.2.7 *Every monoid is a pointed invariant I-module, with the unit $\eta\colon I \to M$ a module homomorphism.*

Proof Given a monoid $(M, \eta\colon I \to M, \mu\colon M \otimes M \to M)$, the I-module structure is given by Diagram $(\oslash\eta)$: $a\colon M \otimes I \xrightarrow{M\otimes\eta} M \otimes M \xrightarrow{\mu} M$. The module unit law is the right monoid law of M; the associativity law is



The compatibility Diagram ($\otimes p$) (which establishes η as a module homomorphism) is:

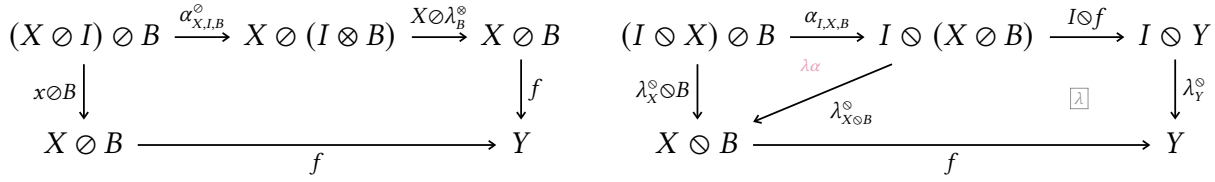


□

We will call monoids seen as invariant I-modules *invariant monoids*, and write **IMod**($\mathcal{V}$) for the category of invariant monoids in $\mathcal{V}$: its objects are monoids with a compatible monoid and module structures, and morphisms are monoid and module homomorphisms.

Example 5.2.15. In the familial model, the proposition above demonstrates that any family equipped with a substitution structure can also be endowed with a renaming structure, simply by interpreting variable renaming as substitution of variable terms. However, in the case of the initial model of a syntax – that is, the inductive datatype generated by constructors and variables – substitution depends on renaming, particularly because substituting under binders necessitates weakening. Thus, the module structure must be established first. Once the monoid structure is subsequently induced, we can prove that the resulting monoid is invariant – that is, substitution of variables for variables is extensionally equal to renaming. This invariant property is a recurring requirement in mechanised formalisations – see Allais et al. (2021, Figure 86) or the *Substitution* chapter of the PLFA textbook. ⌟

Parametrised maps When working with modules over I, properties of parametrised maps are also simplified. Without the need for bimodules, middle-linearity for $X \oslash B \longrightarrow C \in \mathcal{C}$ reduces to the diagram on the left, and left-linearity for $A \otimes B \longrightarrow C$ is automatic:



Middle- and right-linearity requires all three objects in the map to be I-modules, and expresses preservation of the module structure in two ways. If the objects are also pointed, we can ask for the map to preserve all points – this is a stronger condition than simply being a homomorphism $A \otimes B \to C \in I/\mathcal{V}$, as the points of the input tensor are specified separately.

Definition 5.2.12 A parametrised map $f\colon A \otimes B \to C \in \mathcal{V}$ is *pointed* if A, B, C are pointed objects, and the following diagram commutes:

$$\begin{array}{ccc} I \otimes & \xrightarrow{\lambda_I} & I \\ {\scriptstyle p^A \otimes p^B}\downarrow & & \downarrow{\scriptstyle p^C} \\ A \otimes B & \xrightarrow[f]{} & C \end{array} \tag{fP}$$

⌟

Example 5.2.16. Linearity and pointedness for maps with one parameter capture a form of renaming-invariance, but it is limited: the incompatibility due to the lack of quotienting can still be triggered by maps with more parameters. This, again, is no issue in the presheaf model, where a tensor of multiple components $((P \otimes Q) \otimes R)$ bakes in two quotienting conditions and two renaming-invariance properties at different elements of the tuple: for $t \in P[i]$, $\rho \in [j]^i$, $\sigma\colon [j] \Rightarrow Q[k]$, $\varrho \in [l]^k$, $\varsigma\colon [l] \Rightarrow R[m]$ we have identification of tuples in $((P \otimes Q) \otimes R)[m]$:

$$([l], ([j], P(\rho)t, Q(\varrho) \circ \sigma), \varsigma) \sim ([k], ([i], t, \sigma \circ \rho), \varsigma \circ \varrho)$$

A middle-linear map of families $(W \oplus X) \oplus Y \to Z$ could only capture the transposition of ϱ and would not go "deep enough" to account for ρ. ⌟

This motivates the following generalisation, inspired by n-left-linear maps in skew-monoidal categories by Fiore and Saville (2018).

Definition 5.2.13 For $1 \le n$, an *n-parametrised map* in $\mathcal{C}$, denoted $(X \oslash A_{1 \le i \le n})^n \to Y$ for $1 \le i \le n$, is a map of the form $(\cdots((X \oslash A_1) \oslash A_2)\cdots) \oslash A_n \to Y$. The bounds on i are shown if they differ from 1 and the exponent n, otherwise we write $(X \oslash A_i)^n \to Y$. ⌟

Note that any n-parametrised map can be seen as a k-parametrised one for $1 \le k \le n$, grouping together the first $n - k$ parameters with X into $X' \triangleq (X \oslash A_i)^{n-k}$ and $(X' \oslash A_{n-k<i})^k \to Y$.

Definition 5.2.14 An n-parametrised map $f\colon (X \oslash A_i)^n \to Y$ is *n-middle-linear* if (X, x) is an I-module, and the following diagram commutes:

$$\begin{array}{ccccc} ((X \oslash I) \oslash A_i)^n & \xrightarrow{(\alpha_{X,I,A_1} \oslash A_{2\le i})^n} & ((X \oslash (I \otimes A_1)) \oslash A_{2\le i})^n & \xrightarrow{((X \oslash \lambda_{A_1}) \oslash A_{2\le i})^n} & ((X \oslash A_1) \oslash A_{2\le i})^n \\ {\scriptstyle (x \oslash A_i)^n}\downarrow & & & & \downarrow{\scriptstyle f} \\ (X \oslash A_i)^n & & \xrightarrow[f]{} & & Y \end{array}$$

An n-parametrised map is *k-linear* for $1 \le k \le n$ if the k-parametrised map $(X' \oslash A_{n-k<i})^k \to Y$ for $X' = (X \oslash A_i)^{n-k}$ is k-middle-linear, provided (A_{n-k}, a_{n-k}) is an I-module so that $(X \oslash A_i)^{n-k}$ is an I-module with structure map

$$(X \oslash A_i)^{n-k} \oslash I \xrightarrow{\alpha_{(X \oslash A_i)^{n-k-1}, A_{n-k}, I}} (X \oslash A_i)^{n-k-1} \oslash (A_{n-k} \otimes I) \xrightarrow{\mathrm{id} \oslash a_{n-k}} (X \oslash A_i)^{n-k-1} \oslash A_{n-k} = (X \oslash A_i)^{n-k}$$

⌟

We extend the arity of parametrisation to $n = 0$ by taking a 0-parametrised map to simply be $X \to Y$, and 0-middle-linearity to be right-linearity in the sense before (namely, that it is a module homomorphism, assuming X and Y are modules). Middle linearity as defined above then corresponds to 1-middle-linearity. We define multilinear maps to be linear in all available parameters.

Definition 5.2.15 An n-parametrised map $f\colon (X \oslash A_i)^n \to Y$ is called *multilinear* if X, Y and A_i are all I-modules, and it is k-middle-linear for all $0 \leq k \leq n$. If the map $(A \otimes B_i)^n \to C$ resides in $\mathcal{V}$, it is *pointed multilinear* if all its objects are pointed I-modules, and the n-ary point-preservation condition holds, where λ_I^n is the obvious sequence of nested left unitors collapsing the left-biased tensor of units into the unit:

$$\begin{array}{ccc} (I \otimes I_i)^n & \xrightarrow{\lambda_I^n} & I \\ {\scriptstyle (p \otimes p_i)^n}\downarrow & & \downarrow{\scriptstyle q} \\ (A \otimes B_i)^n & \xrightarrow[f]{} & C \end{array}$$

⌟

The hom-set functors from before can now be generalised to

$$\mathbf{MLin}(-, -, \ldots -; -)\colon (\mathbf{Mod}(\mathcal{C}) \times \mathbf{Mod}(\mathcal{V})^n)^{\mathrm{op}} \times \mathbf{Mod}(\mathcal{C}) \to \mathbf{Set}$$
$$\mathbf{PMLin}(-, -, \ldots, -; -)\colon (I/\mathbf{Mod}(\mathcal{V}) \times (I/\mathbf{Mod}(\mathcal{V}))^n)^{\mathrm{op}} \times I/\mathbf{Mod}(\mathcal{V}) \to \mathbf{Set}$$

Example 5.2.17. A parametrised map $(P \otimes Q_i)^n \to R$ in $\widetilde{\mathbb{F}}$ is natural and dinatural in all components. It is equivalent to a multilinear map $(A \oplus B_i)^n \to C$ in $\widetilde{\mathbb{N}}$, which equates all tuples that are identified by the quotienting condition of the coend in the presheaf model. Writing $A\rho t$ for $a(t, \rho)$ with $a\colon A \oplus I \to A$, a multilinear map $f\colon (A \oplus B_i)^n \to C$ satisfies, in C_m:

$$f(\ldots((t, \sigma_1 \circ \rho_1), \sigma_2 \circ \rho_2), \ldots, \sigma_n \circ \rho_n) = f(\ldots((A(\rho_1)t, B_1\rho_2 \circ \sigma_1), B_2\rho_3 \circ \sigma_2), \ldots, \sigma_n)$$

for $t \in A_l$, $\rho_1\colon [k_1]^l$, $\sigma_n\colon (B_n)_m^{k_n}$ for $1 < n$, and $\rho_i\colon [k_{i-2}]^{k_i}$ and $\sigma_j\colon (B_j)_{k_{j+1}}^{k_j}$ for $1 < i, j < n$. ⌟

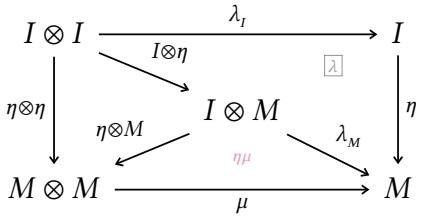
We show some examples of multilinear transformations next.

Lemma 5.2.1 *The following morphisms and natural transformations are multilinear. When the modular category is $\mathcal{V}$, they are furthermore pointed.*

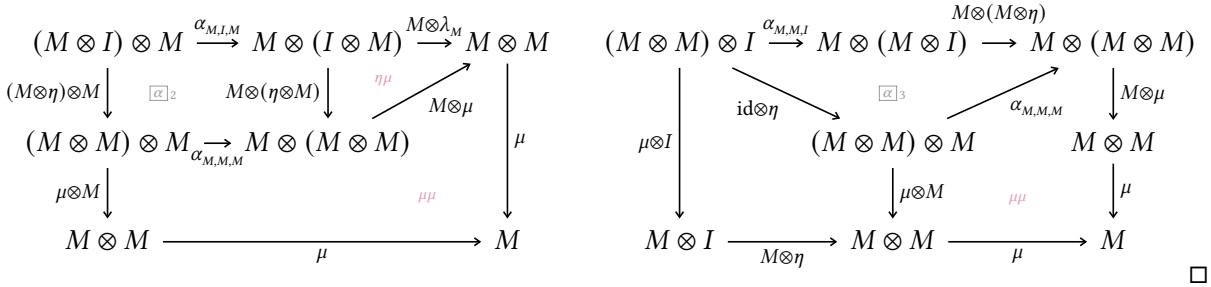
1. *The left unitor $\lambda_B\colon I \otimes B \to B$ for a module B.*
2. *The module structure $x\colon X \oslash I \to X$ for a module X.*
3. *The multiplication $\mu\colon M \otimes M \to M$ for an invariant monoid M.*

Proof The properties for the left unitor follow from naturality, skew-monoidal axioms and module laws. For $b\colon B \otimes I \to B$, point-preservation is Diagram ($\otimes p$), middle-linearity and left-linearity both reduce to the module associativity law Diagram ($\oslash\mu$).

Let (M, η, μ) be an invariant monoid, with its I-module structure given by the pullback module $M \otimes I \xrightarrow{M \otimes \eta} M \otimes M \xrightarrow{\mu} M$. We show that $\mu\colon M \otimes M \to M$ is a multilinear map. The pointedness condition is



The multilinearity conditions expand as

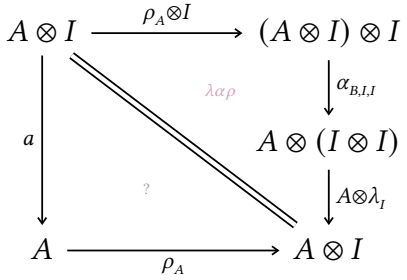


□

Tensoring of modules Pointed multilinear maps will play an essential role in the characterisation of pointed strengths, allowing us to collapse tensors that could not be identified otherwise. Their necessity arises from a more fundamental problem that stands at the heart of our work: skew-monoidal structure on a category does not lift to categories of algebras or modules without the use of coequalisers (Fiore and Saville, 2018, 2021). Families form a skew-monoidal category, but coequalisers correspond to quotienting, which is not supported by a dependently-typed formalisation. Thus, the skew substitution structure does not lift to the category of I-modules: even though the tensor product has an I-module structure if B does

$$(A, a\colon A \otimes I \to B) \otimes (B, b\colon B \otimes I \to B) \triangleq (A \otimes B, (A \otimes B) \otimes I \xrightarrow{\alpha_{A,B,I}} A \otimes (B \otimes I) \xrightarrow{A \otimes b} A \otimes B) \quad (\dagger)$$

this only extends to a left (I-**Mod**($\mathcal{V}$))-action $(-) \oslash (-)\colon \mathcal{V} \times I\text{-}\mathbf{Mod}(\mathcal{V}) \to I\text{-}\mathbf{Mod}(\mathcal{V})$, with transformations λ and α. The right unitor $\rho_{(A,a)}\colon (A, a) \to A \oslash (I, \mathrm{id})$ is not necessarily an I-module homomorphism, as a is only a left inverse of ρ:



Even worse, the point $I \to A \otimes B$ of the tensor product of $(A, p_A), (B, p_B) \in I/\mathcal{V}$ defined as $I \xrightarrow{\rho_I} I \otimes I \xrightarrow{p_A \otimes p_B} A \otimes B$ is not an I-module homomorphism either, so $A \otimes B$ has no pointed module structure.

These limitations are not merely due to our abstraction level: for the substitution tensor product $\oplus$ in $\widetilde{\mathbb{N}}$, the unitor homomorphism diagram above reduces to the equality of tuples $([n], a([m], t, \rho), \mathrm{id})$ and $([m], t, \rho)$ in $(X \oplus Y)_n$, which are not identifiable without quotient-

ing. As a result, we cannot claim that pointed I-modules form a skew-monoidal category. This poses a challenge for the familial model: skew-monoidal structure on pointed modules is essential for a coherent axiomatization of the pointed strength

$$FX \oslash B \longrightarrow F(X \oslash B) : \widetilde{\mathbb{N}} \times I/I\text{-}\mathbf{Mod} \longrightarrow \widetilde{\mathbb{N}}$$

as a right I/I-**Mod**-modular functor. This strength operation is central to several key constructions, including the specification of syntax with substitution, the definition of models for a second-order signature, and more.

The crux of the problem lies in the associativity law for strength: it requires tensoring elements of I/I-**Mod**, a process that – even when performed naïvely as in (†) – results in unprovable equations for signature endofunctors that involve variable binding, such as context extension $\delta(X)_n = X_{n+1}$. Since associativity underpins numerous critical results involving renaming, substitution, and lifting (see Section 2.1.3), its omission is not an option.

To address this, we propose a generalisation of strength via multilinear maps. Specifically, a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ between skew right $(I/I\text{-}\mathbf{Mod})$-modular categories is said to be "multilinear-strong" if its strength

$$s_{X,A} : FX \ominus A \longrightarrow F(X \ominus A)$$

satisfies the standard unit law (Diagram ($s\rho\oslash$)), and the following associativity law formulated for all pointed multilinear maps $f : A \oplus B \longrightarrow C$ in $\mathbf{PMLin}(A, B; C)$:

$$\begin{array}{ccccc}
(FX \ominus A) \ominus B & & \xrightarrow{\quad\alpha_{FX,A,B}\quad} & & FX \ominus (A \otimes B) \\
\downarrow{\scriptstyle s_{X,A}\ominus B} & & & & \downarrow{\scriptstyle FX\ominus f} \\
F(X \oslash A) \ominus B & & & & FX \ominus C \\
\downarrow{\scriptstyle s_{X\oplus A,B}} & & & & \downarrow{\scriptstyle s_{X,C}} \\
F((X \oslash A) \oslash B) & \xrightarrow[F\alpha_{X,A,B}]{} & F(X \oslash (A \otimes B)) & \xrightarrow[F(X\oslash f)]{} & F(X \oslash C)
\end{array} \qquad (s\alpha\oslash L)$$

For the context extension endofunctor δ, the associativity law for strength *is* provable: the problematic tuples in the original diagram, which resist identification, are collapsed by the action of the pointed multilinear map f, and the pointed multiilinearity axioms are sufficient to carry through the calculation. Furthermore, the pointed multilinearity properties of f enable a complete and coherent derivation (see Theorem 10.3.1). This adapted notion of pointed strength is familiar from the literature – for instance, the "coalgebraic strength" in our Agda formalisation (Fiore and Szamozvancev, 2022), or the "structural strength" and Lemma A.5 in Borthelle et al. (2020) – yet a satisfying theoretical justification has so far remained elusive.

In the next section, we begin to address this gap, outlining initial steps toward a generalised theory of strength that accounts for these phenomena in a principled and modular fashion.

CHAPTER 6

Synthetic constructions

A central challenge in reformulating the presheaf model in the category of indexed sets lies in the absence – or inadequacy – of several core constructions necessary for the formal theory. In the previous chapter, we traced the origin of these inconsistencies to the abstract setting of skew-monoidal closed categories and their modules. Chief among these issues is the observation that the category of modules over the unit in a skew-monoidal category does not inherit a tensor product whose underlying object corresponds to the skew-monoidal tensor. This failure makes the standard axiomatisation of pointed-strong functors in this setting impossible.

The guiding intuition toward a resolution is as follows: the category of (pointed) I-modules for a skew-monoidal category often appears monoidal – until one inspects the coherence conditions closely. That is, while we may act as if the category possesses a monoidal structure, the purported unit and tensor product generally fail to satisfy the axioms needed to make them actual I-modules. More broadly, for a skew-monoidal category $\mathcal{V}$, a category $\mathcal{A}$ of $\mathcal{V}$-objects equipped with some additional structure may not itself be monoidal, as the tensor of underlying $\mathcal{V}$-objects may fail to preserve that extra structure. Nevertheless, we can behave as though $\mathcal{A}$ is monoidal insofar as we avoid explicitly tensoring its objects. Functors between such categories may similarly simulate monoidality by interacting with the monoidal structure of $\mathcal{V}$ in regular, predictable ways. Naturally, in cases where $\mathcal{A}$ is genuinely monoidal, the real and simulated structures should coincide.

To formally capture this phenomenon, we introduce the notion of *synthetic monoidal categories*. Synthetic modular categories and synthetic functors are the topic of Section 6.2, and Section 6.3 discusses elevators and liftings between synthetic monoidal/modular functors.

6.1 Synthetic monoidal categories

A synthetic monoidal category $\mathcal{A}$ consists of objects that cannot be tensored formally, but which can be embedded into a skew-monoidal category $\mathcal{V}$ such that a distinguished class of morphisms in $\mathcal{A}$ correspond to multi-parametrised morphisms in $\mathcal{V}$. This is the situation we encounter with families and modules: although $\widetilde{\mathbb{N}}$ is skew-monoidal, the category $\mathbf{Mod}(\widetilde{\mathbb{N}})$ is not; however, the underlying morphism of a multilinear map $f \in \mathbf{MLin}((A,a),(B,b);(C,c))$

is a morphism $f : A \otimes B \to C$. If a synthetic monoidal category happens to be skew-monoidal in its own right, the embedding functor becomes a formally skew-monoidal functor. Functors between synthetic monoidal categories preserve the distinguished class of morphisms in a manner that respects the embedding. The most important example for our purposes is the equivalence $L : \mathbf{Mod} \to \widetilde{\mathbb{F}}$ (with the latter category actually monoidal), which transforms multilinear maps to equivalent parametrised natural transformations.

The benefit of this generalisation is that the strength transformation for functors between synthetic modular categories (over synthetic monoidal categories) can incorporate parametrised multilinear morphisms in a way that makes the unit and associativity axioms constructively provable. The collapsing of tensor structure, which would otherwise lead to the need for quotienting, is encoded directly into the associativity axiom. This provides the correct axiomatization of signature strength in the familial model – accounting for, in particular, the modified strength associativity condition for the context extension endofunctor in **??** – supported by the fact that this form of strength associativity generalises familiar syntactic interaction and fusion properties.

Remark. Finding the right formal setting for this notion of strength has been one of the main technical challenges of our work. While the axioms can be explicitly stated easily enough (see Diagram (*sα⊘L*)) we sought to find a setting where this modified and slightly ad hoc notion of strength is the canonical axiomatisation of modular functors. The theory presented in this chapter is a first approximation, but a better presentation would be using the language of promonoidal categories (Day, 1970) or multicategories (Hermida, 2000; Leinster, 2004), which exactly describe situations where objects of a category are combined without a formal tensor product being available. We give a brief description of this view in Section 14.2, and leave working out the details to future work. ⌟

Fix skew-monoidal categories $(\mathcal{V}, I, \otimes)$ and $(\mathcal{W}, J, \oplus)$.

Definition 6.1.1 Given a skew-monoidal category $\mathcal{V}$, a *synthetic monoidal category* over $\mathcal{V}$ is an object $(\mathcal{A}, E) \in \mathbf{Cat}/\mathcal{V}$ of the slice category associated with profunctors $\mathcal{U} \in \mathbf{1} \nrightarrow \mathcal{A}$ and $\mathscr{M} : \mathcal{A} \times \mathcal{A} \nrightarrow \mathcal{A}$, and natural families of maps $u = \{\mathcal{U}[C] \to \mathcal{V}(I, EC)\}_{C \in \mathcal{A}}$ and $m = \{\mathscr{M}[A, B; C] \to \mathcal{V}(EA \otimes EB, EC)\}_{A,B,C \in \mathcal{A}}$. $\mathcal{A}$ is *representable* if u and m are isomorphisms. ⌟

Notation. A synthetic monoidal category $\mathcal{A}$ over $\mathcal{V}$ will be denoted $(\mathcal{A}_{|\mathcal{V}}, \mathcal{U}, \mathscr{M})$ or just $\mathcal{A}_{|\mathcal{V}}$. Application of E on objects and u and m on morphisms will generally be written as underlining: for $C \in \mathcal{A}$, $EC \in \mathcal{V}$ is denoted $\underline{C}_{\mathcal{A}}$, and for $f \in \mathcal{U}[C]$ and $g \in \mathscr{M}[A, B; C]$, $\underline{f}_{\mathcal{A}} : I \to \underline{C}_{\mathcal{A}}$ and $\underline{g}_{\mathcal{A}} : \underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}} \to \underline{C}_{\mathcal{A}}$, with the category subscript omitted where clear. ⌟

Definition 6.1.2 For synthetic monoidal categories $\mathcal{A}_{|\mathcal{V}}$ and $\mathcal{B}_{|\mathcal{W}}$, a *synthetic monoidal functor* $F : \mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$ is a functor $F : \mathcal{A} \to \mathcal{B}$ with associated *unit* and *multiplication* operators

$$u[-] : \mathcal{U}_{\mathcal{A}}[C] \to \mathcal{U}_{\mathcal{B}}[LC] \qquad m[-] : \mathscr{M}_{\mathcal{A}}[A, B; C] \to \mathscr{M}_{\mathcal{B}}[LA, LB; LC]$$

satisfying the following axioms:

- If for all $f \in U_{\mathcal{A}}[C]$, $g \in \mathscr{M}_{\mathcal{A}}[A, B; C]$ and $h : C \to B \in \mathcal{A}$ the diagram on the left commutes

in $\mathcal{V}$, so does the diagram on the right commute in $\mathcal{W}$:

$$\begin{array}{ccc} I \otimes \underline{B}_{\mathcal{A}} & \xrightarrow{\underline{f}\otimes\mathrm{id}} & \underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}} \\ {\scriptstyle \lambda^{\otimes}_{\underline{B}}}\downarrow & & \downarrow{\scriptstyle \underline{g}} \\ \underline{B}_{\mathcal{A}} & \xleftarrow[\underline{h}]{} & \underline{C}_{\mathcal{A}} \end{array} \qquad \begin{array}{ccc} J \oplus \underline{LB}_{\mathcal{B}} & \xrightarrow{\underline{u[f]}\oplus\mathrm{id}} & \underline{LA}_{\mathcal{B}} \oplus \underline{LB}_{\mathcal{B}} \\ {\scriptstyle \lambda^{\oplus}_{\underline{LB}}}\downarrow & & \downarrow{\scriptstyle \underline{m[g]}} \\ \underline{LB}_{\mathcal{B}} & \xleftarrow[\underline{Lh}]{} & \underline{LC}_{\mathcal{B}} \end{array}$$

- If for all $f \in \mathcal{U}_{\mathcal{A}}[B]$, $g \in \mathscr{M}_{\mathcal{A}}[A, B; C]$ and $h\colon A \to C \in \mathcal{A}$ the diagram on the left commutes in $\mathcal{V}$, so does the diagram on the right commute in $\mathcal{W}$:

$$\begin{array}{ccc} \underline{A}_{\mathcal{A}} & \xrightarrow{\underline{h}} & \underline{C}_{\mathcal{A}} \\ {\scriptstyle \rho^{\otimes}_{\underline{A}}}\downarrow & & \uparrow{\scriptstyle \underline{g}} \\ \underline{A}_{\mathcal{A}} \otimes I & \xrightarrow[\mathrm{id}\otimes\underline{f}]{} & \underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}} \end{array} \qquad \begin{array}{ccc} \underline{LA}_{\mathcal{B}} & \xrightarrow{\underline{Lh}} & \underline{LC}_{\mathcal{B}} \\ {\scriptstyle \rho^{\oplus}_{\underline{LA}}}\downarrow & & \uparrow{\scriptstyle \underline{m[g]}} \\ \underline{LA}_{\mathcal{B}} \oplus J & \xrightarrow[\mathrm{id}\oplus\underline{u[f]}]{} & \underline{LA}_{\mathcal{B}} \oplus \underline{LB}_{\mathcal{B}} \end{array}$$

- If for all $e \in \mathscr{M}_{\mathcal{A}}[A, B; D]$, $f \in \mathscr{M}_{\mathcal{A}}[B, C; E]$, $g \in \mathscr{M}_{\mathcal{A}}[D, C; F]$ and $h \in \mathscr{M}_{\mathcal{A}}[A, E; F]$ the left diagram commutes in $\mathcal{V}$, so does the right diagram commute in $\mathcal{W}$:

$$\begin{array}{ccccc} (\underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}}) \otimes \underline{C}_{\mathcal{A}} & \xrightarrow{\alpha^{\otimes}_{\underline{A},\underline{B},\underline{C}}} & \underline{A}_{\mathcal{A}} \otimes (\underline{B}_{\mathcal{A}} \otimes \underline{C}_{\mathcal{A}}) \\ {\scriptstyle \underline{e}\otimes\mathrm{id}}\downarrow & & \downarrow{\scriptstyle \mathrm{id}\otimes\underline{f}} \\ \underline{D}_{\mathcal{A}} \otimes \underline{C}_{\mathcal{A}} & \xrightarrow{\underline{g}} & \underline{F}_{\mathcal{A}} & \xleftarrow{\underline{h}} & \underline{A}_{\mathcal{A}} \otimes \underline{E}_{\mathcal{A}} \end{array} \qquad \begin{array}{ccccc} (\underline{LA}_{\mathcal{B}} \oplus \underline{LB}_{\mathcal{B}}) \oplus \underline{LC}_{\mathcal{B}} & \xrightarrow{\alpha^{\oplus}_{\underline{LA},\underline{LB},\underline{LC}}} & \underline{LA}_{\mathcal{B}} \oplus (\underline{LB}_{\mathcal{B}} \oplus \underline{LC}_{\mathcal{B}}) \\ {\scriptstyle \underline{m[e]}\oplus\mathrm{id}}\downarrow & & \downarrow{\scriptstyle \mathrm{id}\oplus\underline{m[f]}} \\ \underline{LD}_{\mathcal{B}} \oplus \underline{LC}_{\mathcal{B}} & \xrightarrow{\underline{m[g]}} & \underline{LF}_{\mathcal{B}} & \xleftarrow{\underline{m[h]}} & \underline{LA}_{\mathcal{B}} \oplus \underline{LE}_{\mathcal{B}} \end{array}$$

A functor between representable synthetic monoidal categories is itself *representable* when the unit and multiplication operators restrict to (di)natural transformations

$$u[-]\colon \mathcal{V}\big(I, \underline{C}_{\mathcal{A}}\big) \to \mathcal{W}\big(J, \underline{LC}_{\mathcal{B}}\big) \qquad m[-]\colon \mathcal{V}\big(\underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}}, \underline{C}_{\mathcal{A}}\big) \to \mathcal{W}\big(\underline{LA}_{\mathcal{B}} \oplus \underline{LB}_{\mathcal{B}}, \underline{LC}_{\mathcal{B}}\big)$$

⌟

Definition 6.1.3 A *synthetic monoidal natural transformation* $\kappa\colon L \Longrightarrow N\colon \mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$ preserves the unit and multiplication transformations: for all $f\colon \mathcal{U}_{\mathcal{A}}[C]$ and $g\colon \mathscr{M}_{\mathcal{A}}[A, B; C]$,

$$\begin{array}{ccc} I & \xrightarrow{\underline{u^L[f]}} & \underline{LC}_{\mathcal{B}} \\ & {\scriptstyle \underline{u^N[f]}}\searrow & \downarrow{\scriptstyle \underline{\kappa_C}} \\ & & \underline{NC}_{\mathcal{B}} \end{array} \qquad \begin{array}{ccc} \underline{LA} \oplus \underline{LB}_{\mathcal{B}} & \xrightarrow{\underline{m^L[g]}} & \underline{LC}_{\mathcal{B}} \\ {\scriptstyle \underline{\kappa_A}\oplus\underline{\kappa_B}}\downarrow & & \downarrow{\scriptstyle \underline{\kappa_C}} \\ \underline{NA} \oplus \underline{NB}_{\mathcal{B}} & \xrightarrow{\underline{m^N[g]}} & \underline{NC}_{\mathcal{B}} \end{array}$$

⌟

Example 6.1.1. If $(\mathcal{A}, \widehat{I}, \widehat{\otimes})$ *is* skew-monoidal and $E\colon \mathcal{A} \to \mathcal{V}$ is a skew-monoidal functor, $\mathcal{A}$ is in particular synthetic monoidal with $\mathcal{U}[C] \triangleq \mathcal{A}(\widehat{I}, C)$ and $\mathscr{M}[A, B; C] \triangleq \mathcal{A}(A \mathbin{\widehat{\otimes}} B, C)$. Consequently, every skew-monoidal category is synthetic monoidal over itself. ⌟

Example 6.1.2. A representable synthetic monoidal functor $\mathcal{V}_{|\mathcal{V}} \to \mathcal{W}_{|\mathcal{W}}$ between skew-monoidal categories (presented as synthetic categories as above) is skew-monoidal, with unit and multiplication given by $u \triangleq u[\mathrm{id}_I]\colon J \to LI$ and $m_{A,B} \triangleq m[\mathrm{id}_{A\otimes B}]\colon LA \oplus LB \to L(A \otimes B)$. ⌟

Example 6.1.3. The category of I-modules is synthetic monoidal over $\widetilde{\mathbb{N}}$ via the forgetful functor $U\colon \mathbf{Mod} \to \widetilde{\mathbb{N}}$, with $\mathscr{U}[Z] \triangleq \mathbf{Mod}(I, Z)$ given by module-homomorphic points, and $\mathscr{M}[X, Y; Z] \triangleq \mathbf{MLin}(X, Y; Z)$ given by multilinear maps, with u and m extracting the underlying family maps from the homomorphisms. Notably, while modules cannot be tensored, the family of indices is an I-module with $\lambda_I\colon I \oplus I \to I$, so $(I, \lambda_I) \in \mathbf{Mod}$ and the embedding U strictly preserves this unit: $U(I, \lambda_I) = I$.

The category of presheaves embeds into $\widetilde{\mathbb{N}}$ with a monoidal forgetful functor $\star\colon \widetilde{\mathbb{F}} \to \widetilde{\mathbb{N}}$, making $\widetilde{\mathbb{F}}$ a synthetic monoidal category with $\mathscr{U}[R] \triangleq \widetilde{\mathbb{F}}(V, R)$ and $\mathscr{M}[P, Q; R] \triangleq \widetilde{\mathbb{F}}(P \otimes Q, R)$. It is also synthetic monoidal over itself.

The equivalence $L\colon I\text{-}\mathbf{Mod} \to \widetilde{\mathbb{F}}$ is a synthetic monoidal functor $L\colon \mathbf{Mod}_{|\widetilde{\mathbb{N}}} \to \widetilde{\mathbb{F}}_{|\widetilde{\mathbb{F}}}$, mapping a module homomorphism $(I, \mathrm{id}) \to (Z, z) \in \mathbf{Mod}$ to a natural transformation $V \Longrightarrow L(Z, z) \in \widetilde{\mathbb{F}}$, and a multilinear map $f\colon X \oplus Y \to Z$ to the corresponding natural transformation $L(X, x) \otimes L(Y, y) \Longrightarrow L(Z, z)$ with components given by f, and the naturality and dinaturality conditions by the multilinearity properties. In addition, we have $L(I, \lambda_I) \cong V$ with $I \cong \star V \cong \star L(I, \lambda_I)$. ⌟

Skew-monoidal functors between base categories lift to representable synthetic monoidal categories along the embeddings.

Proposition 6.1.1 *If $\mathcal{A}_{|\mathcal{V}}$ and $\mathcal{B}_{|\mathcal{W}}$ are representable synthetic monoidal categories, and a monoidal functor $K\colon \mathcal{V} \to \mathcal{W}$ lifts to a $L\colon \mathcal{A} \to \mathcal{B}$ with $K\underline{A}_{\mathcal{A}} = \underline{LA}_{\mathcal{B}}$, then L is representable synthetic monoidal $\mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$.*

Proof See the Appendix on page 324. □

Monoids and the monoid-preservation result of lax monoidal functors have an analogue in the synthetic monoidal setting as well.

Definition 6.1.4 A *synthetic monoid* in a synthetic monoidal category $\mathcal{A}_{|\mathcal{V}}$ is an object $M \in \mathcal{A}$, and morphisms $\eta \in \mathscr{U}[M]$ and $\mu \in \mathscr{M}[M, M; M]$ such that $(\underline{M}, \underline{\eta}\colon I \to \underline{M}, \underline{\mu}\colon \underline{M} \otimes \underline{M} \to \underline{M})$ is a monoid in $\mathcal{V}$. ⌟

Proposition 6.1.2 *Synthetic monoidal functors preserve synthetic monoids.*

Proof Let $L\colon \mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$ be a synthetic monoidal functor and (M, η^M, μ^M) a synthetic monoid in $\mathcal{A}$. Then, LM is a synthetic monoid in $\mathcal{B}$ with operations $\eta^{LM} \triangleq u[\eta^M] \in \mathscr{U}[LM]$ and $\mu^{LM} \triangleq m[\mu^M] \in \mathscr{M}[LM, LM; LM]$, because $(\underline{LM} \in \mathcal{W}, \underline{u[\eta^M]}\colon J \to \underline{LM}, \underline{m[\mu^M]}\colon \underline{LM} \oplus \underline{LM} \to \underline{LM})$ is a monoid in $\mathcal{W}$. The monoid laws in $\mathcal{W}$ follow by instantiating the $\mathcal{V}$-diagrams in the structural laws of L with the monoid laws of $\underline{M}$ in $\mathcal{V}$. □

The definition of synthetic modular categories and functors follows a similar pattern.

6.2 Synthetic modular categories

A modular category over a monoidal category $\mathcal{V}$ can be weakened to a synthetic modular category over a synthetic monoidal category $\mathcal{A}_{|\mathcal{V}}$ using a similar approach to the previous section. Identifying when such modular categories are representable is more subtle, as an action bifunctor may exist even if the acting category is only synthetic monoidal.

We give the most general definition, where $\mathcal{A}_{|\mathcal{V}}$ is a synthetic monoidal category and and $\mathcal{C}$ has no actions.

Definition 6.2.1 A *synthetic $\mathcal{A}_{|\mathcal{V}}$-modular category* $(\mathcal{C}_{|\mathcal{E}}, \mathcal{S})$ consists of a $\mathcal{V}$-modular category $(\mathcal{E}, \oslash\colon \mathcal{E} \times \mathcal{V} \to \mathcal{E})$, a category $\mathcal{C}$, a functor $E\colon \mathcal{C} \to \mathcal{E}$, and a profunctor $\mathcal{S}\colon \mathcal{C} \times \mathcal{A} \nrightarrow \mathcal{C}$ with natural family of maps $\mathfrak{z} = \{\mathcal{S}[X, B; Y] \to \mathcal{E}(\underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}}, \underline{Y}_{\mathcal{C}})\}_{X,Y \in \mathcal{C}, B \in \mathcal{A}}$. The category is *representable* if the natural transformation $\mathfrak{z}$ is an isomorphism. As before, application of E and $\mathfrak{z}$ will be denoted by underlining. ⌟

Our definition of synthetic modular functor will also incorporate a change of the acting synthetic monoidal category via restricting the action along a monoidal functor. In a non-synthetic setting, the total category of modular categories can be defined as the Grothendieck construction of the functor $(-)$-**Mod**, consisting of pairs $(\mathcal{V}, \mathcal{C} \in \mathcal{V}\text{-}\mathbf{Mod})$ as objects and pairs $(K, F)\colon (\mathcal{V}, \mathcal{C}) \to (\mathcal{W}, \mathcal{D})$ as morphisms, comprising a skew-monoidal functor $K\colon \mathcal{V} \to \mathcal{W}$ and a functor $F\colon \mathcal{C} \to \mathcal{D}$ with K-relative strength $s^{F,K}_{X,B}\colon FX \ominus KB \to F(X \oslash B)\colon \mathcal{C} \times \mathcal{V} \to \mathcal{D}$. This generalises the notion of a *strong relative monad* by Uustalu (2010) and coincides with the *morphism of actions* defined by Zsido (2010, Section 4.8).

Definition 6.2.2 A *morphism of synthetic modular categories* $(L, F)\colon (\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}) \to (\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}})$ consists of a pair of a relative monoidal functor $L\colon \mathcal{A} \to \mathcal{B}$ and a functor $F\colon \mathcal{C} \to \mathcal{D}$ with an associated *L-relative strength* operator

$$s[-]\colon \mathcal{S}_{\mathcal{C}}(X, B; Y) \to \mathcal{S}_{\mathcal{D}}(FX, LB; FY)$$

satisfying the following axioms:

- If for all $f \in \mathcal{U}_{\mathcal{A}}[B]$, $g \in \mathcal{S}_{\mathcal{C}}[X, B; Y]$ and $h\colon X \to Y \in \mathcal{C}$ the diagram on the left commutes in $\mathcal{E}$, so does the diagram on the right commute in $\mathcal{F}$:

$$\begin{array}{ccc} \underline{X}_{\mathcal{C}} & \xrightarrow{\underline{h}} & \underline{Y}_{\mathcal{C}} \\ {\scriptstyle \rho^{\mathcal{E}}_{\underline{X}}}\downarrow & & \uparrow{\scriptstyle \underline{g}} \\ \underline{X}_{\mathcal{C}} \oslash I & \xrightarrow[\mathrm{id} \oslash \underline{f}]{} & \underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}} \end{array} \qquad \begin{array}{ccc} \underline{FX}_{\mathcal{D}} & \xrightarrow{\underline{Fh}} & \underline{FY}_{\mathcal{D}} \\ {\scriptstyle \rho^{\mathcal{F}}_{\underline{FX}}}\downarrow & & \uparrow{\scriptstyle \underline{s[g]}} \\ \underline{FX}_{\mathcal{D}} \ominus J & \xrightarrow[\mathrm{id} \ominus \underline{u[f]}]{} & \underline{FX}_{\mathcal{D}} \ominus \underline{LB}_{\mathcal{B}} \end{array}$$

- If for all $e \in \mathcal{S}_{\mathcal{C}}[X, B; Y]$, $f \in \mathcal{S}_{\mathcal{C}}[Y, C; Z]$, $g \in \mathscr{M}_{\mathcal{A}}[B, C; D]$ and $h \in \mathcal{S}_{\mathcal{C}}[X, D; Z]$ the diagram on the left commutes in $\mathcal{E}$, then so does the diagram on the right commute in $\mathcal{F}$:

$$\begin{array}{ccccc} (\underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}}) \oslash \underline{C}_{\mathcal{A}} & & \xrightarrow{\alpha^{\mathcal{E}}_{\underline{X},B,C}} & & \underline{X}_{\mathcal{C}} \oslash (\underline{B}_{\mathcal{A}} \otimes \underline{C}_{\mathcal{A}}) \\ {\scriptstyle \underline{e} \oslash \mathrm{id}}\downarrow & & & & \downarrow{\scriptstyle \mathrm{id} \oslash \underline{g}} \\ \underline{Y}_{\mathcal{C}} \oslash \underline{C}_{\mathcal{A}} & \xrightarrow[\underline{f}]{} & \underline{Z}_{\mathcal{C}} & \xleftarrow[\underline{h}]{} & \underline{X}_{\mathcal{C}} \oslash \underline{D}_{\mathcal{A}} \end{array} \qquad \begin{array}{ccccc} (\underline{FX}_{\mathcal{D}} \ominus \underline{LB}_{\mathcal{B}}) \ominus \underline{LC}_{\mathcal{B}} & & \xrightarrow{\alpha^{\mathcal{F}}_{\underline{FX},\underline{LB},\underline{LC}}} & & \underline{FX}_{\mathcal{D}} \ominus (\underline{LB}_{\mathcal{B}} \oplus \underline{LC}_{\mathcal{B}}) \\ {\scriptstyle \underline{s[e]} \ominus \mathrm{id}}\downarrow & & & & \downarrow{\scriptstyle \mathrm{id} \ominus \underline{m[g]}} \\ \underline{FY}_{\mathcal{D}} \ominus \underline{LC}_{\mathcal{B}} & \xrightarrow[\underline{s[f]}]{} & \underline{FZ}_{\mathcal{D}} & \xleftarrow[\underline{s[h]}]{} & \underline{FX}_{\mathcal{D}} \ominus \underline{LD}_{\mathcal{B}} \end{array}$$

The strength is *representable* when the operator restricts to a (di)natural transformation

$$s[-]\colon \mathcal{E}(\underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}}, \underline{Y}_{\mathcal{C}}) \to \mathcal{F}(\underline{FX}_{\mathcal{D}} \ominus \underline{LB}_{\mathcal{B}}, \underline{FY}_{\mathcal{D}})$$

⌟

Definition 6.2.3 A *natural transformation of synthetic modular functors* $(\kappa, \varphi) \colon (L, F) \Longrightarrow (N, G) \colon (\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}) \to (\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}})$ is a pair of a relative monoidal natural transformation $\kappa \colon L \Longrightarrow N \colon \mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$ and a natural transformation $\varphi \colon F \Longrightarrow G \colon \mathcal{C}_{|\mathcal{E}} \to \mathcal{D}_{|\mathcal{F}}$ that preserves the strength operator: for all $h \in \mathcal{S}_{\mathcal{C}}[X, B; Y]$, we have

$$\begin{array}{ccc} \underline{FX}_{\mathcal{D}} \ominus \underline{LB}_{\mathcal{A}} & \xrightarrow{s^{F,L}[h]} & \underline{FY}_{\mathcal{D}} \\ \downarrow{\scriptstyle \underline{\varphi_X} \ominus \underline{\kappa_B}} & & \downarrow{\scriptstyle \underline{\varphi_Y}} \\ \underline{GX}_{\mathcal{D}} \ominus \underline{NB}_{\mathcal{A}} & \xrightarrow[\underline{s^{G,N}[h]}]{} & \underline{GY}_{\mathcal{D}} \end{array} \qquad (\varphi\kappa\lfloor s \rceil)$$

⌟

Synthetic modular categories, functors and natural transformations form the total 2-category **SynMod**. If $(\mathcal{A}, \widehat{I}, \widehat{\otimes})$ is skew-monoidal and $(\mathcal{C}, \widehat{\oslash} \colon \mathcal{C} \times \mathcal{A} \to \mathcal{C})$ is a $\mathcal{A}$-modular category, $\mathcal{C}$ can be equipped with synthetic modular category structure with $\mathcal{S}[X, B; Y] \triangleq \mathcal{C}(X \widehat{\oslash} B, Y)$. More generally, one can talk about representable modular categories over synthetic monoidal categories, which amounts to a category $\mathcal{C}$ with an action $\oslash \colon \mathcal{C} \times \mathcal{A} \to \mathcal{C}$ and representable synthetic unitor $\rho[-]_X \colon \mathscr{U}[C] \to \mathcal{C}(X, X \oslash C)$ and $\alpha[-]_X \colon \mathscr{M}[A, B; C] \to \mathcal{C}((X \oslash A) \oslash B, X \oslash C)$ satisfying axioms. While this view is appropriate in our setting – where pointed strength for a signature endofunctor is on the modular category $\widetilde{\mathbb{N}}$ over the synthetic monoidal category $I/\mathbf{Mod}$, with an action $X \oslash (B, p, b) \triangleq X \otimes B$ – using the language of fully synthetic modular categories will be sufficient. The example below demonstrates the value of the generalisation: it encapsulates the correct notion of pointed strength in the familial model.

Example 6.2.1. $I/\mathbf{Mod}$ is a synthetic monoidal category over $\widetilde{\mathbb{N}}$ with $\mathscr{U}$ and $\mathscr{M}$ given by point-preserving module homomorphisms and pointed multilinear maps, respectively. Then $\widetilde{\mathbb{N}}$ is a synthetic $I/\mathbf{Mod}$-modular category with $\mathcal{S}[X, \boldsymbol{Y}; Z] \triangleq \widetilde{\mathbb{N}}(X \otimes Y, Z)$, where the pointed module $\boldsymbol{Y} = (Y, p \colon J \to Y, y \colon Y \oplus J \to Y)$ satisfies $\underline{\boldsymbol{Y}} = Y$. An Id-relative $I/\mathbf{Mod}$-modular endofunctor $\Sigma \colon \widetilde{\mathbb{N}} \to \widetilde{\mathbb{N}}$ satisfies the unit and associativity laws for the synthetic strength transformation $s[-] \colon \mathcal{S}(X, \boldsymbol{Y}; Z) \to \mathcal{S}(\Sigma X, \boldsymbol{Y}; \Sigma Z)$. Instantiating the hypothesis of the synthetic strength associativity law with the diagram

$$\begin{array}{ccccc} (W \oplus X) \oplus Y & & \xrightarrow{\quad\alpha_{W,X,Y}\quad} & & W \oplus (X \oplus Y) \\ \downarrow{\scriptstyle \mathrm{id}} & & & & \downarrow{\scriptstyle W \oplus g} \\ (W \oplus X) \oplus Y & \xrightarrow[(W \oplus g) \circ \alpha_{W,X,Y}]{} & W \oplus Z & \xleftarrow[\mathrm{id}]{} & W \oplus Z \end{array}$$

where $g \colon X \oplus Y \to Z \in \mathbf{PMLin}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{Z})$ is a pointed multilinear map, we get the diagram

$$\begin{array}{ccccc} (FW \oplus X) \oplus Y & & \xrightarrow{\quad\alpha_{FW,X,Y}\quad} & & FW \oplus (X \oplus Y) \\ \downarrow{\scriptstyle s_{W,X} \oplus Y} & & & & \downarrow{\scriptstyle FW \oplus g} \\ F(W \oplus X) \oplus Y & & & & FW \oplus Z \\ \downarrow{\scriptstyle s_{W \oplus X, Y}} & & & & \downarrow{\scriptstyle s_{W,Z}} \\ F((W \oplus X) \oplus Y) & \xrightarrow[F\alpha_{W,X,Y}]{} & F(W \oplus (X \oplus Y)) & \xrightarrow[F(W \oplus g)]{} & F(W \oplus Z) \end{array}$$

which is exactly the associativity law in Diagram ($s\alpha\oslash L$) we wished to capture. For $F = \delta$, the

law becomes provable, as the postcomposition with the pointed multilinear map collapses tensor products that would otherwise only be equal up to quotienting. The standard associativity law Diagram (*sα⊘*) of modular functors is not derivable for δ, as the identity $X \oplus Y \longrightarrow X \oplus Y$ is not a pointed multilinear map. ⌟

Example 6.2.2. The canonical map $\langle \pi_1 \oplus Y, \pi_2 \oplus Y \rangle \colon (X \times X) \oplus Y \longrightarrow (X \oplus Y) \times (X \oplus Y)$ exhibits the functor $F(X) \triangleq X \times X \colon \widetilde{\mathbb{N}} \longrightarrow \widetilde{\mathbb{N}}$ as skew $\widetilde{\mathbb{N}}$-modular, or more generally, representable Id-relative synthetic $\widetilde{\mathbb{N}}_{|\widetilde{\mathbb{N}}}$-modular. The synthetic strength associativity law can now be instantiated with

$$\begin{array}{ccccc}
(W \oplus X) \oplus Y & & \xrightarrow{\quad\alpha_{W,X,Y}\quad} & & W \oplus (X \oplus Y) \\
{\scriptstyle \mathrm{id} \oplus Y}\downarrow & & & & \downarrow{\scriptstyle W \oplus \mathrm{id}} \\
(W \oplus X) \oplus Y & \xrightarrow[\alpha_{W,X,Y}]{} & W \oplus (X \oplus Y) & \xleftarrow[\mathrm{id}]{} & W \oplus (X \oplus Y)
\end{array}$$

where all parametrised morphisms are just the identities, as we do not ask for any constraints (e.g. multilinearity) on any of e, f, g or h. Upon application of F, the law expands to

$$\begin{array}{ccccc}
((W \times W) \oplus X) \oplus Y & & \xrightarrow{\quad\alpha_{W\times W,X,Y}\quad} & & (W \times W) \oplus (X \oplus Y) \\
{\scriptstyle s_{W,X} \oplus Y}\downarrow & & & & \downarrow{\scriptstyle s_{W,X\oplus Y}} \\
((W \oplus X) \times (W \oplus X)) \oplus Y & & & & (W \oplus (X \oplus Y)) \times (W \oplus (X \oplus Y)) \\
& {\scriptstyle s_{W\oplus X,Y}}\searrow & & \nearrow{\scriptstyle \alpha_{W,X,Y} \times \alpha_{W,X,Y}} & \\
& & ((W \oplus X) \oplus Y) \times ((W \oplus X) \oplus Y) & &
\end{array}$$

which expresses the mapping of $(((t_1, t_2), \sigma), \varsigma)$ to $\big(((t_1, \sigma), \varsigma), ((t_2, \sigma), \varsigma)\big)$. ⌟

An analogue of Proposition 6.1.1 is stated as follows.

Proposition 6.2.1 *Let $(K, H) \colon (\mathcal{V}, (\mathcal{E}, \oslash)) \longrightarrow (\mathcal{W}, (\mathcal{F}, \ominus))$ be a modular functor with K-relative strength $s^{H,K}_{X,B} \colon HX \ominus KB \longrightarrow H(X \oslash B) \colon \mathcal{E} \times \mathcal{V} \longrightarrow \mathcal{F}$ and suppose $L \colon \mathcal{A} \longrightarrow \mathcal{B}$ lifts K with $\underline{LA}_{\mathcal{B}} = K\underline{A}_{\mathcal{A}}$ and $N \colon \mathcal{C} \longrightarrow \mathcal{D}$ lifts H with $\underline{NX}_{\mathcal{D}} = H\underline{X}_{\mathcal{C}}$. Also suppose $\mathcal{A}_{|\mathcal{V}}, \mathcal{B}_{|\mathcal{W}}$ are representable synthetic monoidal and $\mathcal{C}_{|\mathcal{E}}, \mathcal{D}_{|\mathcal{F}}$ are representable synthetic modular categories. Then (N, L) is a representable synthetic modular functor from $(\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}})$ to $(\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}})$.*

Proof The strength transformation for (N, L) is as follows:

$$s[h \colon \underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}} \longrightarrow \underline{Y}_{\mathcal{C}}] \triangleq \underline{NX}_{\mathcal{D}} \ominus \underline{LB}_{\mathcal{B}} = H\underline{X}_{\mathcal{C}} \ominus K\underline{B}_{\mathcal{A}} \xrightarrow{s^{H,K}_{X,B}} H(\underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}}) \xrightarrow{Hh} H\underline{Y}_{\mathcal{C}} = \underline{NY}_{\mathcal{D}}$$

The synthetic strength laws are proved similarly to Proposition 6.1.1, by applying the lifting equalities to the target diagrams and using naturality and skew monoidal axioms. □

Finally, let us consider module objects in synthetic modular categories, and what parametrised and multilinear maps look like in this setting.

Definition 6.2.4 Let $M \in \mathcal{A}$ be a synthetic monoid in $\mathcal{A}_{|\mathcal{V}}$. A *synthetic M-module* in a synthetic $\mathcal{A}_{|\mathcal{V}}$-modular category $\mathcal{C}_{|\mathcal{E}}$ is an object $X \in \mathcal{C}$ with map $x \colon \mathcal{S}(X, M; X)$ such that $(\underline{X}, \underline{x})$ is an $\underline{M}$-module in $\mathcal{E}$. ⌟

Proposition 6.2.2 *For a synthetic monoidal $K\colon \mathcal{A} \to \mathcal{B}$, a K-relative synthetic modular functor $F\colon \mathcal{C} \to \mathcal{D}$ maps a synthetic M-module X to a synthetic KM-module FX.*

Proof The action $x \in \mathcal{S}_{\mathcal{C}}[X, M; X]$ is mapped to $s[x] \in \mathcal{S}_{\mathcal{D}}[FX, KM; FX]$, and the module axioms for FX follow by instantiating the laws for a synthetic modular functor with module axioms for X. □

We can adapt the notion of linear maps to the setting of synthetic modular categories.

Definition 6.2.5 If $F\colon \mathcal{C} \to \mathcal{C}$ is a Id-relative synthetic $\mathcal{A}_{|\mathcal{V}}$-modular functor, $B \in \mathcal{A}$ is an object, and $(X, x), (Y, y)$ are F-algebras, a map $f\colon \mathcal{S}[X, B; Y]$ is *synthetic F-linear* if the following diagram commutes in $\mathcal{E}$:

$$\begin{array}{ccc} \underline{FX}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}} & \xrightarrow{\underline{s[f]}} & \underline{FY}_{\mathcal{C}} \\ {\scriptstyle \underline{x} \oslash B}\downarrow & & \downarrow{\scriptstyle \underline{y}} \\ \underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}} & \xrightarrow[\underline{f}]{} & \underline{Y}_{\mathcal{C}} \end{array} \tag{$Fs[f]$}$$

An *synthetic F-module over M* for a synthetic monoid $M \in \mathcal{A}$ is an F-algebra $(X, x) \in \mathcal{C}$ such that X is a synthetic M-module and the action $x \in \mathcal{S}[X, M; X]$ is synthetic F-linear. A *synthetic F-monoid* for $F\colon \mathcal{A} \to \mathcal{A}$ is a synthetic monoid $M \in \mathcal{A}$ such that M is a synthetic F-module over M. ⌟

Definition 6.2.6 Given a synthetic $\mathcal{A}_{|\mathcal{V}}$-modular category $\mathcal{C}_{|\mathcal{E}}$, a map $f \in \mathcal{S}(X, B; Y)$ for $X, Y \in \mathcal{C}$ and $B \in \mathcal{A}$ is *synthetic multilinear* if $\underline{f}\colon \underline{X}_{\mathcal{C}} \oslash \underline{B}_{\mathcal{A}} \to \underline{Y}_{\mathcal{C}}$ is multilinear in $\mathcal{E}$. ⌟

Proposition 6.2.3 *Given a multilinear map $f \in \mathcal{S}_{\mathcal{C}}[X, B; Y]$ and an* Id*-relative synthetic modular functor $F\colon \mathcal{C} \to \mathcal{D}$, the map $s[f]\colon \mathcal{S}_{\mathcal{D}}[FX, B; FY]$ is also multilinear.*

Proof Follows by instantiating the associativity axiom of synthetic modular functors with the multilinearity laws. □

The last two sections introduced synthetic monoidal categories and associated notions, giving a general enough definition of strength to encompass context extension and therefore signature endofunctors to be discussed in **??**. We next investigate liftings and elevators in the categories discussed.

6.3 Synthetic liftings

The 2-category **SynMod** is quite intricate, so it is worth spelling out some important constructions explicitly: namely, liftings and distributive laws, as introduced in Chapter 3. Since the familial model employs a variety of functors and strengths, some synthetic, some representable, some lifted, etc., having a clean framework to compare and relate them will be useful.

Given two endo-1-cells in the category **SynMod** of synthetic modules and strong functors

$$(K, F)\colon \left(\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}\right) \to \left(\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}\right) \qquad (L, G)\colon \left(\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}}\right) \to \left(\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}}\right)$$

an elevator between them is a 1-cell $(N,H)\colon (\mathcal{A}_{|\mathcal{V}},\mathcal{C}_{|\mathcal{E}}) \to (\mathcal{B}_{|\mathcal{W}},\mathcal{D}_{|\mathcal{F}})$ and a 2-cell $(\kappa,\varphi)\colon (L,G)\circ(N,H)\Longrightarrow(N,H)\circ(K,F)$. Explicitly, we have functors with operators

$$\begin{aligned} s^{F,K}[-]&\colon \mathcal{S}_{\mathcal{C}}[X,B;Y]\to\mathcal{S}_{\mathcal{C}}[FX,KB;FY]\\ s^{G,L}[-]&\colon \mathcal{S}_{\mathcal{D}}[X,B;Y]\to\mathcal{S}_{\mathcal{D}}[GX,LB;GY]\\ s^{H,N}[-]&\colon \mathcal{S}_{\mathcal{C}}[X,B;Y]\to\mathcal{S}_{\mathcal{D}}[HX,NB;HY] \end{aligned}$$

a synthetic monoidal natural transformation $\kappa\colon LN\Longrightarrow NK\colon \mathcal{A}_{|\mathcal{V}}\to\mathcal{B}_{|\mathcal{W}}$, and a natural transformation $\varphi\colon GH\Longrightarrow HF\colon \mathcal{C}\to\mathcal{D}$ satisfying, for all $h\in\mathcal{S}_{\mathcal{C}}[X,B;Y]$,

$$\begin{array}{ccc} \underline{GHX}_{\mathcal{D}}\ominus\underline{LNB}_{\mathcal{B}} & \xrightarrow{\underline{s^{G,L}[s^{H,N}[h]]}} & \underline{GHY}_{\mathcal{D}} \\ {\scriptstyle \underline{\varphi_X}\ominus\underline{\kappa_B}}\downarrow & & \downarrow{\scriptstyle \underline{\varphi_Y}} \\ \underline{HFX}_{\mathcal{D}}\ominus\underline{NKB}_{\mathcal{B}} & \xrightarrow[\underline{s^{H,N}[s^{F,K}[h]]}]{} & \underline{HFY}_{\mathcal{D}} \end{array}$$

Restriction of modular functors along monoidal transformations has its synthetic analogue: given $(L,F)\colon(\mathcal{A}_{|\mathcal{V}},\mathcal{C}_{|\mathcal{E}})\to(\mathcal{B}_{|\mathcal{W}},\mathcal{D}_{|\mathcal{F}})$ and a synthetic monoidal transformation $\varphi\colon K\Longrightarrow L$, we can construct a 1-cell $(K,\varphi^*(L,F))$ with $F\colon\mathcal{C}\to\mathcal{D}$ and an K-relative strength operator

$$s^{F,K}[-]\colon \mathcal{S}_{\mathcal{C}}[X,B;Y]\xrightarrow{s^{F,L}[-]}\mathcal{S}_{\mathcal{D}}[FX,LB;FY]\xrightarrow{\mathcal{S}[FX,\varphi_B;FY]}\mathcal{S}_{\mathcal{D}}[FX,KB;FY]$$

The following lemma concerns the interaction of elevators in **SynMod** and restrictions of 1-cells along natural transformations.

Lemma 6.3.1 *For synthetic monoidal transformations $\alpha\colon K'\Longrightarrow K$ and $\beta\colon L'\Longrightarrow L$, an elevator $\big((N,H),(\varphi,\varphi')\big)$ from (K,F) to (L,G) in* **SynMod** *extends to one from $\alpha^*(K,F)$ to $\beta^*(L,G)$ provided $\varphi\colon LN\Longrightarrow NK$ lifts to $\varphi'\colon L'N\Longrightarrow NK'$ with $\varphi\circ\beta N = N\alpha\circ\varphi'$.*

Proof Take an elevator in **SynMod**, consisting of a synthetic monoidal transformation $\varphi\colon LM\Longrightarrow MK$ and a natural transformation $\varphi\colon GH\Longrightarrow HF$ satisfying the square above. Assume further that there is a $\varphi'\colon L'N\Longrightarrow NK'$ such that

$$\begin{array}{ccc} L'N & \xrightarrow{\beta N} & LN \\ {\scriptstyle \varphi'}\downarrow & & \downarrow{\scriptstyle \varphi} \\ NK' & \xrightarrow[N\alpha]{} & NK \end{array} \qquad (\dagger)$$

We construct the elevator $\beta^*(L,G)\circ(N,H)\Longrightarrow(N,H)\circ\alpha^*(K,F)$ with the synthetic monoidal transformation $\varphi'\colon L'N\Longrightarrow NK'\colon\mathcal{A}_{|\mathcal{V}}\to\mathcal{B}_{|\mathcal{W}}$, and natural transformation $\varphi\colon GH\Longrightarrow HF$ satisfying the strength-preservation condition

$$\begin{array}{ccccc} \underline{GHX}_{\mathcal{D}}\ominus\underline{L'NB}_{\mathcal{B}} & \xrightarrow{\mathrm{id}\oslash\beta_{NB}} & \underline{GHX}_{\mathcal{D}}\ominus\underline{LNB}_{\mathcal{B}} & \xrightarrow{s^{G,L}[s^{H,L}[h]]} & GHY \\ {\scriptstyle \underline{\varphi_X}\oslash\underline{\varphi'_B}}\downarrow & \dagger & {\scriptstyle \underline{\varphi_X}\oslash\underline{\varphi_B}}\downarrow & \varphi\kappa\lfloor s\rceil & \downarrow{\scriptstyle \varphi_Y} \\ \underline{HFX}_{\mathcal{D}}\ominus\underline{NK'B}_{\mathcal{B}} & \xrightarrow[\mathrm{id}\oslash\underline{N\alpha_B}]{} & \underline{HFX}_{\mathcal{D}}\ominus\underline{NKB}_{\mathcal{B}} & \xrightarrow[s^{H,L}[s^{F,K}[h]]]{} & HFY \end{array}$$

□

Example 6.3.1. Given a synthetic monoidal transformation $\eta\colon \mathrm{Id} \Longrightarrow K$, a K-relative synthetic strength $s^{F,K}\colon \mathcal{S}_{\mathcal{C}}[X, B; Y] \to \mathcal{S}_{\mathcal{D}}[FX, KB; FY]$ can be restricted to an Id-relative strength from $(\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}})$ to $(\mathcal{A}_{|\mathcal{V}}, \mathcal{D}_{|\mathcal{F}})$:

$$\eta^*(s^{F,K}_{X,B})\colon \mathcal{S}_{\mathcal{C}}[X, B; Y] \to \mathcal{S}_{\mathcal{D}}[FX, B; FY]$$

⌟

With the framework of synthetic modular categories and liftings now set up, the main theorem of this section is straightforward.

Theorem 6.3.1

Given Id*-relative synthetic module endofunctors* $F\colon \mathcal{C}_{|\mathcal{E}} \to \mathcal{C}_{|\mathcal{E}}$ *and* $G\colon \mathcal{D}_{|\mathcal{F}} \to \mathcal{D}_{|\mathcal{F}}$*, if there is a lifting* $(N, H)\colon (\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}) \to (\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}})$ *from* F *to* G*, then every synthetic* F*-linear map* $h \in \mathcal{S}_{\mathcal{C}}[X, B; Y]$ *becomes a synthetic* G*-linear map* $s^{H,N}[h] \in \mathcal{S}_{\mathcal{D}}[HX, NB; HY]$*.*

Proof Take endofunctors $F\colon \mathcal{C} \to \mathcal{C}$ and $G\colon \mathcal{D} \to \mathcal{D}$ with strength operators

$$s^F[-]\colon \mathcal{S}_{\mathcal{C}}[X, B; Y] \to \mathcal{S}_{\mathcal{C}}[FX, B; FY] \qquad s^G[-]\colon \mathcal{S}_{\mathcal{D}}[X, B; Y] \to \mathcal{S}_{\mathcal{D}}[GX, B; GY]$$

and a lifting $(N, H)\colon (\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}) \to (\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}})$ with a synthetic monoidal functor $N\colon \mathcal{A} \to \mathcal{B}$, a functor $H\colon \mathcal{C} \to \mathcal{D}$ with strength operator

$$s^{H,N}[-]\colon \mathcal{S}_{\mathcal{C}}[X, B; Y] \to \mathcal{S}_{\mathcal{D}}[HX, NB; HY]$$

and a natural transformation $\varphi\colon GH \Longrightarrow HF\colon \mathcal{C} \to \mathcal{D}$. Let $(X, x), (Y, y) \in \mathcal{C}$ be F-algebras and $h \in \mathcal{S}_{\mathcal{C}}[X, B; Y]$ an F-linear map. We show that $s^{H,N}[h] \in \mathcal{S}_{\mathcal{D}}[HX, NB; FY]$ is a G-linear map. The G-algebra structure on HX and HY is given by the elevator $GH \Longrightarrow HF$, and the linearity condition is stated as follows:

$$\begin{array}{ccc}
\underline{GHX}_{\mathcal{D}} \ominus \underline{NB}_{\mathcal{B}} & \xrightarrow{s^G[s^{H,N}[h]]} & \underline{GHY}_{\mathcal{D}} \\
{\scriptstyle \underline{\varphi_X} \oslash \mathrm{id}} \downarrow & & \downarrow {\scriptstyle \underline{\varphi_Y}} \\
\underline{HFX}_{\mathcal{D}} \ominus \underline{NB}_{\mathcal{B}} & \xrightarrow[s^{H,N}[s^F[h]]]{} & \underline{HFY}_{\mathcal{D}} \\
{\scriptstyle \underline{Hx} \oslash \mathrm{id}} \downarrow & & \downarrow {\scriptstyle \underline{Hy}} \\
\underline{HX}_{\mathcal{D}} \ominus \underline{NB}_{\mathcal{B}} & \xrightarrow[s^{H,N}[h]]{} & \underline{HY}_{\mathcal{D}}
\end{array}$$

The bottom rectangle is the application of $s^{H,N}[-]$ to the F-linearity axiom for h, and using the naturality of the strength operator to extract the algebra applications. The top rectangle is the strength-preservation condition of the elevator φ. □

Corollary 6.3.1 *For* Id*-relative synthetic module endofunctors* $F\colon \mathcal{C}_{|\mathcal{E}} \to \mathcal{C}_{|\mathcal{E}}$ *and* $G\colon \mathcal{D}_{|\mathcal{F}} \to \mathcal{D}_{|\mathcal{F}}$*, if there is a lifting* $(N, H)\colon (\mathcal{A}_{|\mathcal{V}}, \mathcal{C}_{|\mathcal{E}}) \to (\mathcal{B}_{|\mathcal{W}}, \mathcal{D}_{|\mathcal{F}})$ *from* F *to* G*, then* $H\colon \mathcal{C} \to \mathcal{D}$ *maps synthetic* F*-modules over* $M \in \mathcal{A}$ *to synthetic* G*-modules over* $NM \in \mathcal{B}$*.*

Proof From Proposition 6.2.2 we know that (N, H) preserves synthetic modules, mapping $(X, f \in \mathcal{S}_{\mathcal{C}}[X, M; X])$ to $(HX, s^{H,N}[f] \in \mathcal{S}_{\mathcal{D}}[HX, NM; HX])$. The G-algebra structure composed as $GHX \to HFX \to FX$ is compatible with the module structure by Theorem 6.3.1. □

As every synthetic monoidal category $\mathcal{A}_{|\mathcal{V}}$ is a synthetic modular category over itself, the following lifting result of synthetic F-monoids follows from the preceding corollary.

Corollary 6.3.2 *Given* Id*-relative module endofunctors* $F\colon \mathcal{A}_{|\mathcal{V}} \to \mathcal{A}_{|\mathcal{V}}$ *and* $G\colon \mathcal{B}_{|\mathcal{W}} \to \mathcal{B}_{|\mathcal{W}}$ *(modular over* $\mathcal{A}_{|\mathcal{V}}$ *and* $\mathcal{B}_{|\mathcal{W}}$*, respectively), if there is a synthetic monoidal lifting* $H\colon \mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$ *from* F *to* G*, then* $H\colon \mathcal{A} \to \mathcal{B}$ *maps synthetic* F*-monoids* $M \in \mathcal{A}$ *to synthetic* G*-monoids* $HM \in \mathcal{B}$*.*

These lifting results will be used in the next section to show that algebraic monoids in presheaves map to algebraic monoids in families, establishing one half of the equivalence of syntactic models in presheaves and families.

Having singled out our categorical setting – the total category of synthetic modules, functors, and natural transformations – we will turn to an orthogonal categorical tool for constructing and relating skew-monoidal categories.

CHAPTER 7

Warped constructions

Our mathematical theory involves monoidal structures across several categories, and relating them formally is one of our main undertakings. A useful categorical tool for this is a *warping*, introduced by Booker and Street (2013) as the minimal structure needed on a functor $F\colon \mathcal{V} \to \mathcal{V}$ to make $A \oplus B \triangleq FA \otimes B$ a tensor product, and thereby transport monoidal structure across categories. Lack and Street generalised this concept to skew-monoidal categories (Lack and Street, 2012[b]), and then to actions thereof (Lack and Street, 2015). In Section 7.1, we recap the definition and main constructions in the monoidal case, then adapt the notion to the closed setting, proving an equivalence result between warpings along adjoint functors. We finally analyse the liftings of categorical structures across warping functors in Section 7.2, justifying the lifting properties we rely on when adapting properties of presheaves to properties of families and co/algebras.

7.1 Skew warpings

We recall the definition of a skew-monoidal warping from Lack and Street (2012[b]) and define closed warpings, followed by a discussion of adjoint warpings and their equivalence.

7.1.1 Monoidal warpings

In this section, we let $(\mathcal{V}, I, \otimes)$ be a skew-monoidal category and $(\mathcal{C}, \mathbin{⦸}\colon \mathcal{V}\times\mathcal{C} \to \mathcal{C})$ a skew left $\mathcal{V}$-modular category.

Definition 7.1.1 A *skew-monoidal warping over* $(\mathcal{C}, \mathbin{⦸})$ consists of

- a functor $F\colon \mathcal{C} \to \mathcal{V}$
- an object $J \in \mathcal{C}$
- a morphism $v\colon FJ \to I \in \mathcal{V}$
- a natural family of maps $k_X\colon X \to FX \mathbin{⦸} J\colon \mathcal{C} \to \mathcal{C}$
- a natural family of maps $p_{X,Y}\colon F(FX \mathbin{⦸} Y) \to FX \otimes FY\colon \mathcal{C}\times\mathcal{C} \to \mathcal{V}$

satisfying the following axioms:

$$\begin{array}{ccc} FX & \xrightarrow{\rho^{\otimes}_{FX}} & FX \otimes I \\ {\scriptstyle Fk_X}\downarrow & & \uparrow{\scriptstyle \mathrm{id}\otimes v} \\ F(FX \otimes J) & \xrightarrow[p_{X,J}]{} & FX \otimes FJ \end{array} \quad (\rho p) \qquad \begin{array}{ccc} FX \otimes Y & \xrightarrow{\mathrm{id}\otimes k_Y} & FX \otimes (FY \otimes J) \\ {\scriptstyle k_{FX\otimes Y}}\downarrow & & \uparrow{\scriptstyle \alpha^{\otimes}_{FX,FY,J}} \\ F(FX \otimes Y) \otimes J & \xrightarrow[p_{X,Y}\otimes J]{} & (FX \otimes FY) \otimes J \end{array} \quad (\alpha k)$$

$$\begin{array}{ccc} J & = & J \\ {\scriptstyle k_J}\downarrow & & \uparrow{\scriptstyle \lambda^{\otimes}_J} \\ FJ \otimes J & \xrightarrow[v\otimes J]{} & I \otimes J \end{array} \quad (\lambda v) \qquad \begin{array}{ccccc} F(FJ \otimes Y) & \xrightarrow{p_{J,Y}} & FJ \otimes FY & \xrightarrow{v\otimes FY} & I \otimes FY \\ {\scriptstyle F(v\otimes Y)}\downarrow & & & & \downarrow{\scriptstyle \lambda^{\otimes}_{FY}} \\ F(I \otimes Y) & & \xrightarrow[F\lambda^{\otimes}_Y]{} & & FY \end{array} \quad (\lambda p)$$

$$\begin{array}{ccccc} F(F(FX \otimes Y) \otimes Z) & & \xrightarrow{p_{FX\otimes Y,Z}} & & F(FX \otimes Y) \otimes FZ \\ {\scriptstyle F(p_{X,Y}\otimes Z)}\downarrow & & & & \downarrow{\scriptstyle p_{X,Y}\otimes FZ} \\ F((FX \otimes FY) \otimes Z) & & & & (FX \otimes FY) \otimes FZ \\ {\scriptstyle F\alpha^{\otimes}_{FX,FY,Z}}\downarrow & & & & \downarrow{\scriptstyle \alpha^{\otimes}_{FX,FY,FZ}} \\ F(FX \otimes (FY \otimes Z)) & \xrightarrow[p_{FX,FY\otimes Z}]{} & FX \otimes F(FY \otimes Z) & \xrightarrow[FX\otimes p_{Y,Z}]{} & FX \otimes (FY \otimes FZ) \end{array} \quad (\alpha p)$$

The main application is transporting the skew-monoidal structure of $\mathcal{V}$ along F to one on $\mathcal{C}$.

Theorem 7.1.1

For a skew warping $(J \in \mathcal{C}, F\colon \mathcal{C} \to \mathcal{V})$ over $(\mathcal{C}, \otimes)$, $\mathcal{C}$ can be equipped with another skew-monoidal structure with tensor $X \oplus Y \triangleq FX \otimes Y\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ and unit $J \in \mathcal{C}$.

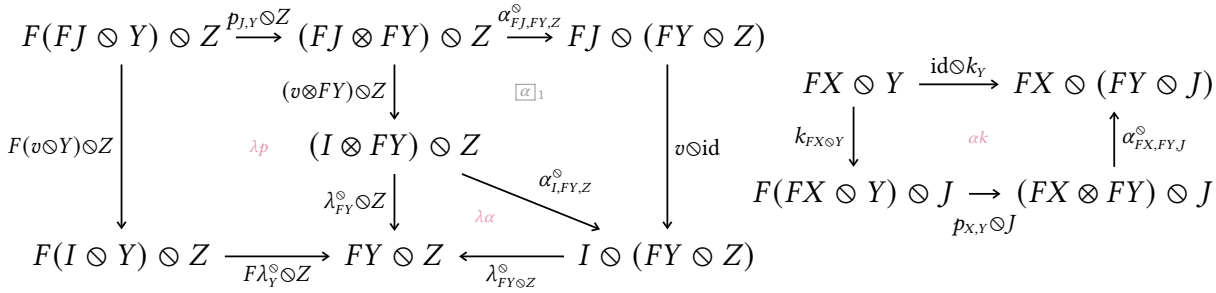
Proof First, the structural transformations:

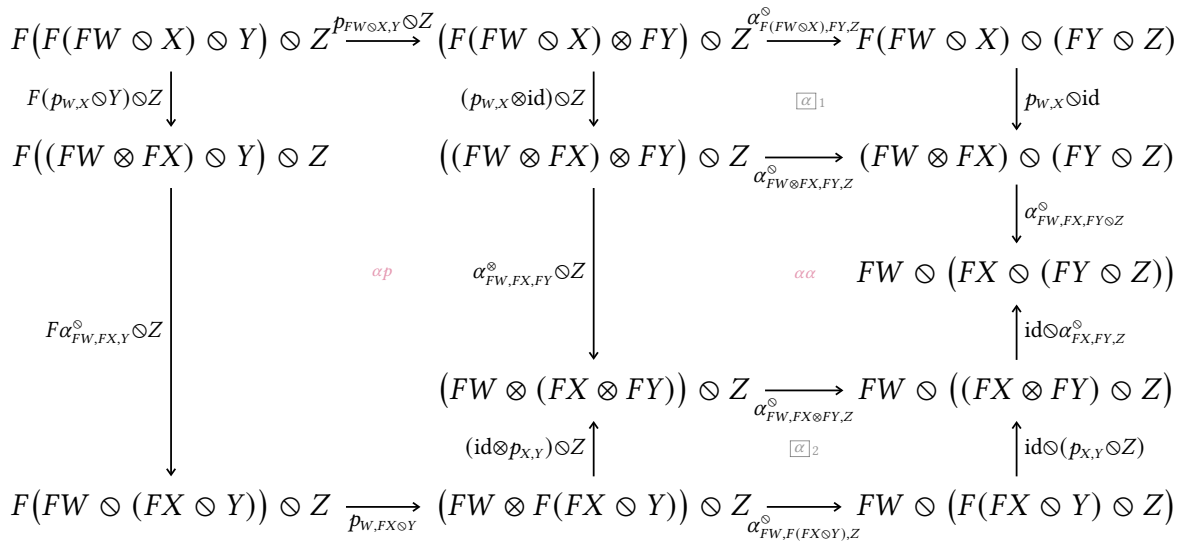
- The left unitor $\lambda^{\oplus}_X\colon J \oplus Y \to Y$ is $FJ \otimes Y \xrightarrow{v\otimes Y} I \otimes Y \xrightarrow{\lambda^{\otimes}_Y} Y$
- The right unitor $\rho^{\oplus}_X\colon X \to X \oplus J$ is $k_X\colon X \to FX \otimes J$;
- The associator $\alpha^{\oplus}_{X,Y,Z}\colon (X \oplus Y) \oplus Z \to X \oplus (Y \oplus Z)$ is

$$F(FX \otimes Y) \otimes Z \xrightarrow{p_{X,Y}\otimes Z} (FX \otimes FY) \otimes Z \xrightarrow{\alpha^{\otimes}_{FX,FY,Z}} FX \otimes (FY \otimes Z)$$

We construct the coherence diagrams for a skew-monoidal category as follows.

$$\begin{array}{ccc} J & = & J \\ {\scriptstyle k_J}\downarrow & \lambda v & \uparrow{\scriptstyle \lambda^{\otimes}_J} \\ FJ \otimes J & \xrightarrow[v\otimes J]{} & I \otimes J \end{array} \qquad \begin{array}{ccccc} FX \otimes Y & = & & = & FX \otimes Y \\ {\scriptstyle Fk_X\otimes Y}\downarrow & \searrow^{\rho^{\otimes}_{FX}\otimes Y} & & \lambda\alpha\rho & \uparrow{\scriptstyle \mathrm{id}\otimes\lambda^{\otimes}_Y} \\ & & (FX \otimes I) \otimes Y & \xrightarrow[\alpha^{\otimes}_{FX,I,Y}]{} & FX \otimes (I \otimes Y) \\ & & \uparrow{\scriptstyle (\mathrm{id}\otimes v)\otimes Y} & \rho p \quad \boxed{\alpha}_2 & \uparrow{\scriptstyle \mathrm{id}\otimes(v\otimes Y)} \\ F(FX \otimes J) \otimes Y & \xrightarrow[p_{X,J}\otimes Y]{} & (FX \otimes FJ) \otimes Y & \xrightarrow[\alpha^{\otimes}_{FX,FJ,Y}]{} & FX \otimes (FJ \otimes Y) \end{array}$$





We furthermore have that $F\colon \mathcal{C} \to \mathcal{V}$ lifts to a skew opmonoidal functor $(\mathcal{C}, J, \oplus) \to (\mathcal{V}, I, \otimes)$ with unit morphism $v\colon FJ \to I$ and multiplication $p_{X,Y}\colon F(X \oplus Y) \to FX \otimes FY$. □

Example 7.1.1. The mixed substitution action $(-) \oslash (=)\colon \widetilde{\mathbb{F}} \times \widetilde{\mathbb{N}} \to \widetilde{\mathbb{N}}$ of the skew-monoidal (in fact, monoidal) $\widetilde{\mathbb{F}}$ on $\widetilde{\mathbb{N}}$ may be turned into a tensor product using the free presheaf functor $\blacklozenge\colon \widetilde{\mathbb{N}} \to \widetilde{\mathbb{F}}$ which – as we will show later – has the structure of a warping. The $\widetilde{\mathbb{N}}$-tensor $X \oplus Y \triangleq \blacklozenge X \oslash Y$ is isomorphic to the definition we have been using thus far by Yoneda:

$$\int^{m \in \mathbb{N}} \Big(\sum_{k \in \mathbb{N}} X_k \times [m]^k\Big) \times (Y_n)^m \cong \sum_{k \in \mathbb{N}} X_k \times (Y_n)^k$$

⌟

7.1.2 Closed warpings

The notion of a warping for skew-closed categories has not appeared in literature before, but it can be easily reverse-engineered from the motivating theorem: given a functor $G\colon \mathcal{V} \to \mathcal{C}$ and object $J \in \mathcal{C}$ for a skew-closed category $(\mathcal{V}, I, [-,=])$ and a left skew $\mathcal{V}$-modular category $(\mathcal{C}, \langle -,= \rangle\colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V})$, what extra structure and properties are required to make $\mathcal{C}$ with unit J and hom $G\langle -,= \rangle\colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{C}$ a skew-closed category?

Definition 7.1.2 A *skew-closed warping over* $(\mathcal{C}, \langle -,= \rangle)$ consists of

- a functor $G\colon \mathcal{V} \to \mathcal{C}$
- an object $J \in \mathcal{C}$

- a morphism $u\colon J \to GI \in \mathcal{C}$
- a natural family of maps $l_X\colon G\langle J, X\rangle \to X\colon \mathcal{C} \to \mathcal{C}$
- a natural family of maps $q_{A,B}\colon G[A,B] \to G\langle GA, GB\rangle\colon \mathcal{V}^{\mathrm{op}} \times \mathcal{V} \to \mathcal{C}$

satisfying the following axioms:

$$\begin{array}{ccc} G[I,B] & \xrightarrow{G i_B^{[]}} & GB \\ {\scriptstyle q_{I,B}}\downarrow & & \uparrow{\scriptstyle l_{GB}} \\ G\langle GI, GB\rangle & \xrightarrow[G\langle u,\mathrm{id}\rangle]{} & G\langle J, GB\rangle \end{array} \qquad (iq)$$

$$\begin{array}{ccc} & GI & \\ {\scriptstyle G j_A^{[]}}\swarrow & & \searrow{\scriptstyle G j_{GA}^{\langle\rangle}} \\ G[A,A] & \xrightarrow[q_{A,A}]{} & G\langle GA, GA\rangle \end{array} \qquad (jq)$$

$$\begin{array}{ccc} J & = & J \\ {\scriptstyle u}\downarrow & & \uparrow{\scriptstyle l_J} \\ GI & \xrightarrow[G j_J^{\langle\rangle}]{} & G\langle J, J\rangle \end{array} \qquad (ju)$$

$$\begin{array}{ccc} G\langle Y, Z\rangle & \xrightarrow{GL_{Y,Z}^{\langle\rangle J}} & G[\langle J, Y\rangle, \langle J, Z\rangle] \\ {\scriptstyle G\langle l_Y, Z\rangle}\downarrow & & \downarrow{\scriptstyle q_{\langle J,Y\rangle,\langle J,Z\rangle}} \\ G\langle G\langle J, Y\rangle, Z\rangle & \xleftarrow[G\langle \mathrm{id}, l_Z\rangle]{} & G\langle G\langle J, Y\rangle, G\langle J, Z\rangle\rangle \end{array} \qquad (Ll)$$

$$\begin{array}{ccccc} G[B,C] & \xrightarrow{q_{B,C}} & G\langle GB, GC\rangle & \xrightarrow{GL_{GB,GC}^{\langle\rangle GA}} & G[\langle GA, GB\rangle, \langle GA, GC\rangle] \\ {\scriptstyle GL_{B,C}^{[]A}}\downarrow & & & & \downarrow{\scriptstyle q_{\langle GA,GB\rangle,\langle GA,GC\rangle}} \\ G[[A,B],[A,C]] & & & & G\langle G\langle GA, GB\rangle, G\langle GA, GC\rangle\rangle \\ {\scriptstyle q_{[A,B],[A,C]}}\downarrow & & & & \downarrow{\scriptstyle G\langle q_{A,B}, \mathrm{id}\rangle} \\ G\langle G[A,B], G[A,C]\rangle & & \xrightarrow[G\langle \mathrm{id}, q_{A,C}\rangle]{} & & G\langle G[A,B], G\langle GA, GC\rangle\rangle \end{array} \qquad (Lq)$$

As intended, the main transport property is an easy consequence of the axioms.

Theorem 7.1.2

For a skew-closed warping $(G\colon \mathcal{V} \to \mathcal{C}, J \in \mathcal{C})$ over $(\mathcal{C}, \otimes)$, $\mathcal{C}$ can be equipped with another skew-closed structure with internal hom $[\![X, Y]\!] \triangleq G\langle X, Y\rangle$ and unit $J \in \mathcal{C}$.

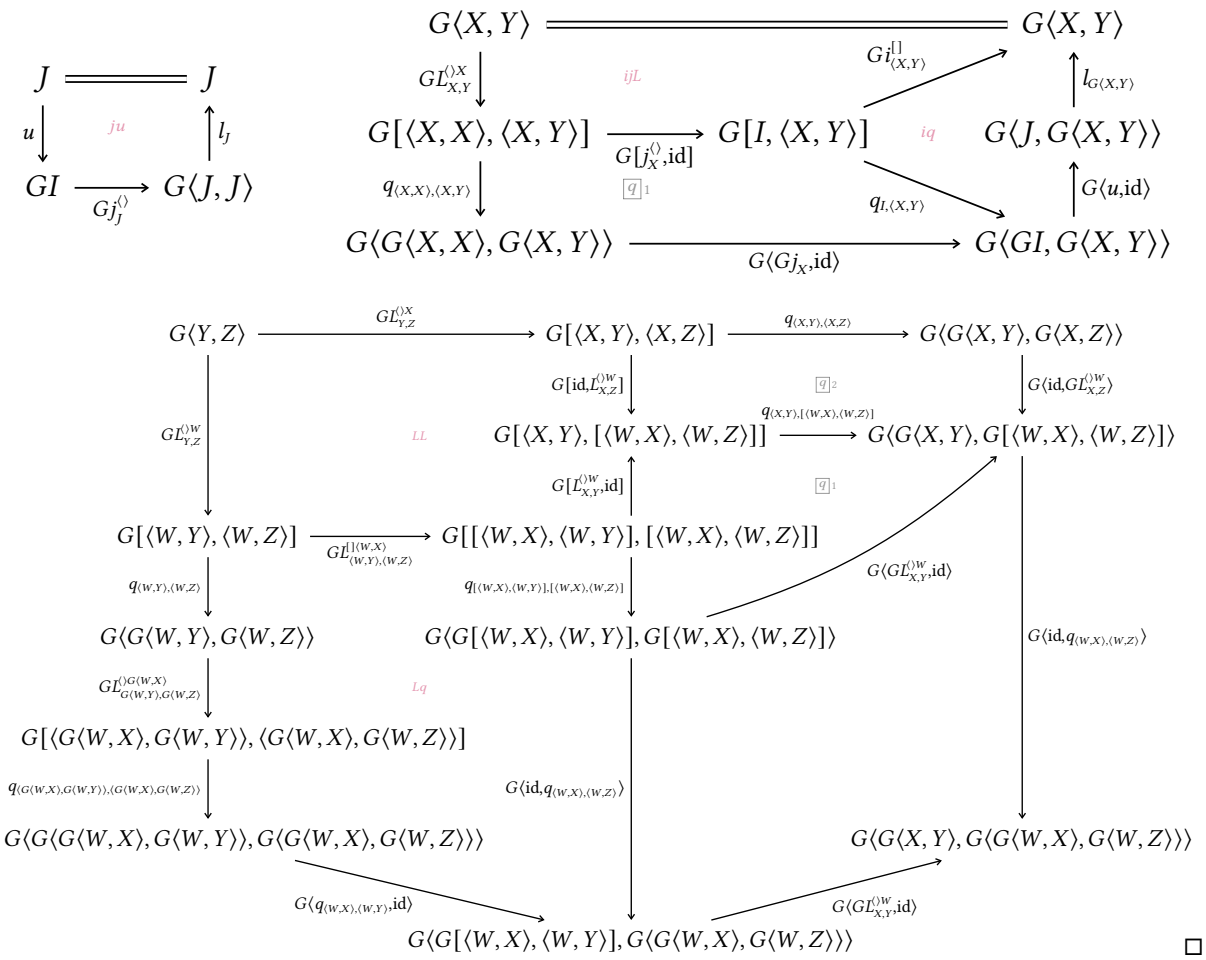
Proof First, the structural transformations:

- The left unitor $j_X^{[[]]}\colon J \to [\![X, X]\!]$ is $J \xrightarrow{u} GI \xrightarrow{G j_X^{\langle\rangle}} G\langle X, X\rangle$;
- The right unitor $i_X^{[[]]}\colon [\![J, X]\!] \to X$ is $l_X\colon G\langle J, X\rangle \to X$;
- The compositor $L_{Y,Z}^{[[]]X}\colon [\![Y, Z]\!] \to [\![[\![X, Y]\!], [\![X, Z]\!]]\!]$ is

$$G\langle Y, Z\rangle \xrightarrow{GL_{Y,Z}^{\langle\rangle X}} G[\langle X, Y\rangle, \langle X, Z\rangle] \xrightarrow{q_{\langle X,Y\rangle,\langle X,Z\rangle}} G\langle G\langle X, Y\rangle, G\langle X, Z\rangle\rangle$$

We construct the coherence diagrams for a skew-closed category as follows:

$$\begin{array}{ccc} G\langle Y, Z\rangle & \xrightarrow{GL_{Y,Z}^{J}} & G[\langle J, Y\rangle, \langle J, Z\rangle] \\ {\scriptstyle G\langle l_Y, Z\rangle}\downarrow & Lq & \downarrow{\scriptstyle q_{\langle J,Y\rangle,\langle J,Z\rangle}} \\ G\langle G\langle J, Y\rangle, Z\rangle & \xleftarrow[G\langle \mathrm{id}, l_Z\rangle]{} & G\langle G\langle J, Y\rangle, G\langle J, Z\rangle\rangle \end{array} \qquad \begin{array}{ccccc} J & \xrightarrow[u]{} & GI & \xrightarrow{G j_{G\langle X,Y\rangle}} & G\langle G\langle X, Y\rangle, G\langle X, Y\rangle\rangle \\ & & {\scriptstyle G j_Y}\downarrow & Lj \quad {\scriptstyle G j_{\langle X,Y\rangle}} \quad jq & \uparrow{\scriptstyle G q_{\langle X,Y\rangle,\langle X,Y\rangle}} \\ & & G\langle Y, Y\rangle & \xrightarrow[GL_{Y,Y}^{X}]{} & G[\langle X, Y\rangle, \langle X, Y\rangle] \end{array}$$



7.1.3 Adjoint warpings

Since the functors for monoidal and closed warpings go in opposite directions, it is worth investigating the interaction between warpings on adjoint functors. Indeed, we find that the monoidal and closed warping structures are equivalent.

Theorem 7.1.3

Suppose we have an adjunction $F \dashv G \colon \mathcal{C} \to \mathcal{V}$ for $(\mathcal{V}, I, \otimes, [-,=])$ skew-monoidal closed and $(\mathcal{C}, \circledcirc, \langle -,= \rangle)$ a left skew-monoidal closed action. Then, F is a skew-monoidal warping riding $\circledcirc$ if and only if G is a skew-closed warping riding $\langle -,= \rangle$.

Proof See the Appendix on page 325. □

Corollary 7.1.1 *Given an adjoint warping $F \dashv G \colon \mathcal{C} \to \mathcal{V}$, the skew left $\mathcal{V}$-modular category $\mathcal{C}$ inherits the skew-monoidal closed structure of $\mathcal{V}$.*

Proof Above we showed that skew-monoidal closed warpings can be used to equip the category $\mathcal{C}$ with a tensor $X \oplus Y \triangleq FX \circledcirc Y$, and a hom $[\![X, Y]\!] \triangleq G\langle X, Y \rangle$, both with unit $J \triangleq GI$. To show that the category is then skew-monoidal closed, it is sufficient to establish the tensor-hom adjunction $(-) \oplus Y \dashv [\![Y, =]\!]$ – but it is merely a composition of the adjunction $(-) \circledcirc Y \dashv \langle Y, = \rangle$ and $F \dashv G$. □

The following theorem gives an efficient way of inducing adjoint warpings and will be applicable in the case of our adjoint modalities in Section 9.3.2.

Theorem 7.1.4

For an adjoint triple $F \dashv G \dashv H \colon \mathcal{C} \to \mathcal{V}$, if $G \colon \mathcal{V} \to \mathcal{C}$ is a strong $\mathcal{V}^{\circledcirc}$-module functor, then F and G form adjoint warpings.

Proof See the Appendix on page 332. □

The lemma above uses an adjoint triple as it offers a streamlined construction of the warping, and is the situation we will encounter in Section 9.3.2; however, the existence of a right adjoint to G is not essential for the result.

Example 7.1.2. The free-forgetful-cofree adjunction $\blacklozenge \dashv \star \dashv \blacksquare$ satisfies $\star(P \otimes Q) \cong P \oslash \star Q$ so induces an adjoint warping between $\blacklozenge$ and $\star$, with the skew-monoidal closed structure on $\widetilde{\mathbb{N}}$ given by $X \oplus Y \triangleq \blacklozenge X \oslash Y$ and $[\![X, Y]\!] \triangleq \star\langle X, Y\rangle$. ⌟

Adjoint warpings can be induced with the minimal assumption of the right/middle adjoint being a strong module functor, allowing us to transport skew-monoidal closed structure on a category to one on its modular category. The skew-monoidal closed substitution structure on families is induced in this way from the analogous (but stronger) structure on presheaves. We next investigate the interaction between co/monads induced by the adjunction and the respective skew-monoidal closed structures, and establish some equivalence results in the case that the warping G is monadic.

7.2 Warped adjoint triples

The adjoint situation discussed in the previous section and assumed throughout this section – namely, an adjoint triple $F \dashv G \dashv H \colon \mathcal{C} \to \mathcal{V}$, with G a strong $\mathcal{V}^{\circledcirc}$-module functor – provides rich opportunities for studying the interaction of the induced skew-monoidal closed structure on $\mathcal{C}$ with functors thereon and objects within.

7.2.1 Warpings and co/monads

The adjoint triple $F \dashv G \dashv H$ with $A \oslash GB \cong G(A \otimes B)$ induces a skew-monoidal closed structure $(\oplus, [\![-,=]\!])$ on $\mathcal{C}$, as well as an adjoint monad-comonad pair $T \dashv C \colon \mathcal{C} \to \mathcal{C}$, with $T \triangleq GF$ and $C \triangleq GH$. In this section we analyse co/algebras for these in $\mathcal{C}$, relating them to the modules over J. To strengthen the results, assume further that $\mathcal{V}$ is right-normal, i.e. $\rho_A \colon A \cong A \otimes I$ and $i_A \colon [I, A] \cong A$ are isomorphisms.

As is the case in any skew-monoidal category, every monoid $M \in \mathcal{C}$ gives rise to the monad $(-) \oplus M \colon \mathcal{C} \to \mathcal{C}$ (Proposition 5.2.2). Modules over the unit J are of particular interest, as they coincide with T-algebras and C-coalgebras.

Proposition 7.2.1 *We have the following natural isomorphisms:*

$$T \cong {}^{\oplus}J \qquad {}^{[\![]\!]}J \cong C$$

Proof The natural isomorphism $T \cong (-) \oplus J$ has components

$$TX = GFX \cong G(FX \otimes I) \cong FX \oslash GI \triangleq X \oplus J$$

Similarly,

$$CX = GHX \cong G[I, HX] \cong G\langle GI, X\rangle = [\![J, X]\!]$$ □

Naturally isomorphic functors have isomorphic categories of co/algebras.

Corollary 7.2.1 *We have the following isomorphisms of categories:*

$$T\text{-}\mathbf{Alg}(\mathcal{C}) \cong \mathcal{C}^{\oplus} \cong \mathcal{C}^{[\![]\!]} \cong C\text{-}\mathbf{Coalg}(\mathcal{C})$$

This allows us to translate between results about categories of co/algebras and J-modules (Section 5.2.1). For example, we know that $X \oplus (-) \colon \mathcal{C} \to \mathcal{C}$ lifts to J-modules, and therefore to T-algebras: for a T-algebra (Y, y), $X \oplus Y$ has T-algebra structure with

$$T(X \oplus Y) \cong (X \oplus Y) \oplus J \xrightarrow{\alpha^{\oplus}_{X,Y,J}} X \oplus (Y \oplus J) \cong X \oplus TY \xrightarrow{X \oplus y} X \oplus Y$$

In the presence of the strength $s_{A,B} \colon G(A \otimes B) \cong A \oslash GB$, this can be expanded in terms of $\oslash$:

$$T(X \oplus Y) = GF(X \oplus Y) \xrightarrow{Gm^F_{X,Y}} G(FX \otimes FY) \xrightarrow{s^G_{FX,FY}} FX \oslash GFY \xrightarrow{\mathrm{id} \oslash y} FX \oslash Y \triangleq X \oplus Y$$

The internal hom $[\![X, Y]\!]$ has a canonical C-coalgebra structure, independent of whether X, Y are coalgebras (that is, it is a skew left $\mathcal{V}^{[]}$-action $[\![-, =]\!] \colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$):

$$[\![X, Y]\!] \triangleq G\langle X, Y\rangle \xrightarrow{G\pi_{\langle X,Y\rangle}} GHG\langle X, Y\rangle \triangleq C[\![X, Y]\!]$$

Of course, as we know from Section 5.2.3, these identities do not claim that T-$\mathbf{Alg}(\mathcal{C})$ is skew-monoidal or C-$\mathbf{Coalg}(\mathcal{C})$ is skew-closed – the same issues with the T-algebra homomorphism property of the right unitor $X \to X \oplus J$ apply, which requires us to show that $TX \to X \to X \oplus J$ is the identity. We must therefore be careful not to assume that T-$\mathbf{Alg}(\mathcal{C})$ is monoidal, even though both $\mathcal{V}$ and $\mathcal{C}$ are.

Example 7.2.1. The monoidal category of presheaves is in particular right-normal, with $P \cong P \otimes V$. Thus, the monad $\Diamond$ and comonad $\Box$ induced by the adjoint triple $\blacklozenge \dashv \star \dashv \blacksquare$ are equivalent to the right I-module structure, with $\Diamond X \cong X \oplus I$ and $\Box X \cong [\![I, X]\!]$. ⌟

We now consider the relationship between the functors that make up the adjoint triple, and the induced warped monoidal closed structure.

Proposition 7.2.2 *In the context of the adjoint warped triple $F \dashv G \dashv H$, $F \colon \mathcal{C} \to \mathcal{V}$ is an oplax monoidal functor, and $G \colon \mathcal{V} \to \mathcal{C}$ is lax monoidal closed.*

Proof The first part follows from the fact that F is a warping; the second part is a standard result about adjoints to oplax monoidal functors, equipping G with monoidal structure

$$u^G \colon J \xrightarrow{=} GI$$

$$m^G_{A,B} \colon GA \oplus GB \xrightarrow{\tau_{GA \oplus GB}} GF(GA \oplus GB) \xrightarrow{Gm^F_{GA,GB}} G(FGA \otimes FGB) \xrightarrow{G(\tau_A \otimes \tau_B)} G(A \otimes B)$$

Since the multiplication is defined in terms of m^F which is itself derived from s, the composite can be simplified to:

$$GA \oplus GB \xrightarrow{s_{FGA,B}} G(FGA \otimes B) \xrightarrow{G(\overline{\tau}_A \otimes B)} G(A \otimes B)$$

The lax closed structure of G is the warping

$$m^G_{A,B} \triangleq q_{A,B} \colon G[A,B] \to G\langle GA, GB\rangle \triangleq [\![GA, GB]\!]$$

which can be equivalently expressed in terms of the transpose isomorphism

$$G[A,B] \xrightarrow{G[A,\pi B]} G[A, HGB] \xrightarrow{t_{A,GB}} G\langle GA, GB\rangle \triangleq [\![GA, GB]\!] \qquad \square$$

Lemma 7.2.1 *We have the following identities relating the isomorphisms of Proposition 7.2.1 and the monoidal/closed transformations*

$$\begin{array}{ccc} GA \oplus J & \xrightarrow{m^G_{A,I}} & G(A \otimes I) \\ \cong \big\downarrow & & \big\downarrow G\rho_A \\ TGA & \xrightarrow[G\overline{\tau}_A]{} & GA \end{array} \qquad \begin{array}{ccc} GA & \xrightarrow{G\pi_A} & CGA \\ Gi_A \big\downarrow & & \big\downarrow \cong \\ G[I,A] & \xrightarrow[m^G_{I,A}]{} & [\![J, GA]\!] \end{array}$$

Proof These follow from the expansion properties shown above.

$$\begin{array}{ccc} GA \oplus J & \xrightarrow{m^G_{A,I}} & G(A \otimes I) \\ {\scriptstyle s_{FGA,I}}\big\downarrow & \nearrow_{G(\overline{\tau}_A \otimes I)} & \\ G(FGA \otimes I) & & \big\downarrow {\scriptstyle G\rho_A} \\ {\scriptstyle G\rho_{FGA}}\big\downarrow & \boxed{p} & \\ TGA & \xrightarrow[G\overline{\tau}_A]{} & GA \end{array} \qquad \begin{array}{ccc} GA & \xrightarrow{G\pi_A} & CGA \\ \big\downarrow {\scriptstyle Gi_A} & \boxed{i} & \big\downarrow {\scriptstyle Gi_{HGA}} \\ & & G[I, HGA] \\ G[I,A] & \xrightarrow[m^G_{I,A}]{} & [\![J, GA]\!] \end{array} \qquad \begin{array}{c} \\ \nearrow_{G[I,\pi_A]} \\ \\ \big\downarrow {\scriptstyle t_{I,GA}} \\ \\ \end{array}$$

$\square$

7.2.2 Monadic warpings

The adjunction $F \dashv G \colon \mathcal{V} \to \mathcal{C}$ induces the canonical comparison functor $K \colon \mathcal{V} \to T\text{-}\mathbf{Alg}(\mathcal{C})$, mapping objects $A \in \mathcal{V}$ to objects $GA \in \mathcal{C}$ with T-algebra structure given by the counit $(TGA = GFGA) \xrightarrow{G\overline{\tau}^A} GA$. As G is monoidal, it maps monoids in $\mathcal{V}$ to monoids in $\mathcal{C}$, which themselves are canonically T-algebras due to Proposition 5.2.7: every monoid $N \in \mathcal{C}$ is invariant, and the J-module structure is equivalently a T-algebra:

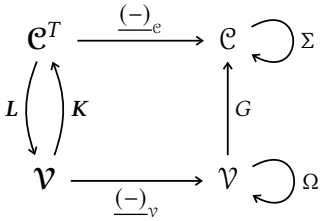
$$t \colon TN \cong N \oplus J \xrightarrow{N \oplus \eta} N \oplus N \xrightarrow{\mu} N$$

If an endofunctor $\Sigma \colon \mathcal{C} \to \mathcal{C}$ lifts to $\Omega \colon \mathcal{V} \to \mathcal{V}$ along G, any Ω-monoid in $\mathcal{V}$ induces a Σ-monoid in $\mathcal{C}$ by G. In this section we identify the sufficient conditions for this mapping to be invertible, establishing the equivalence of Σ-monoids in $\mathcal{C}$ and Ω-monoids in $\mathcal{V}$. In the

presheaf and familial models, this will prove a precise alignment between models of second-order syntax in families and presheaves (see Theorem 10.2.2).

Start by assuming that $G\colon \mathcal{V} \to \mathcal{C}$ is monadic, so the comparison functor $K\colon \mathcal{V} \to T\text{-}\mathbf{Alg}(\mathcal{C})$ has a weak inverse $L\colon T\text{-}\mathbf{Alg}(\mathcal{C}) \to \mathcal{V}$. Assume further that the equivalence extends to pointed categories $I/\mathcal{V} \simeq J/\mathcal{C}^T$ (where $\mathcal{C}^T$ abbreviates $T\text{-}\mathbf{Alg}(\mathcal{C})$), with corresponding functors denoted $\boldsymbol{K}$ and $\boldsymbol{L}$, respectively. We show that the equivalence extends to categories of algebraic monoids using the framework of *synthetic monoidal categories*: the algebra/modular categories $\mathcal{C}^T$ or $J/\mathcal{C}^T$ are not monoidal, but they embed into the skew-monoidal category $\mathcal{C}$ so monoidal structure within may be indirectly connected to $\mathcal{V}$.

Consider the situation below, where boldface symbols denote pointed variants of categories and functors (e.g. $\boldsymbol{\mathcal{C}^T} = J/\mathcal{C}^T$), $\boldsymbol{\mathcal{C}^T}$ is synthetic monoidal with unit and multiplication profunctors $\boldsymbol{\mathcal{C}^T}(\boldsymbol{J}, -)$ and $\mathbf{PMLin}(-,-;-)$, and $\boldsymbol{\mathcal{V}}$ is skew-monoidal with unit and multiplication profunctors $\boldsymbol{\mathcal{V}}(\boldsymbol{I}, -)$ and $\boldsymbol{\mathcal{V}}(- \otimes -, -)$, where the pointed unit and tensor are $\boldsymbol{I} = (I, \mathrm{id}) \in I/\mathcal{V}$ and $\boldsymbol{P} \otimes \boldsymbol{Q} = (P \otimes Q, I \xrightarrow{\rho_I} I \otimes I \xrightarrow{p_P \otimes p_Q} P \otimes Q)$:



Proposition 7.2.3 *If $\boldsymbol{L}$ is a synthetic monoidal functor from $\boldsymbol{\mathcal{C}^T}_{|\mathcal{C}}$ to $\boldsymbol{\mathcal{V}}_{|\mathcal{V}}$, the categories $\mathbf{Mon}(\mathcal{C})$ and $\mathbf{Mon}(\mathcal{V})$ are equivalent.*

Proof The mapping of monoids in $\mathcal{V}$ to monoids in $\mathcal{C}$ is performed by the skew-monoidal functor $G\colon \mathcal{V} \to \mathcal{C}$, which is the underlying functor of the comparison functor $K\colon \mathcal{V} \to \mathcal{C}^T$. Conversely, let $(N, \eta\colon I \to N, \mu\colon N \oplus N \to N)$ be a monoid in $\mathcal{C}$, which is also a pointed invariant monoid $\boldsymbol{N} \in \boldsymbol{\mathcal{C}^T}$ by Proposition 5.2.7 and the point $\eta\colon J \to N$, which is a pointed algebra homomorphism $\boldsymbol{\eta}\colon \boldsymbol{J} \to \boldsymbol{N}$. The multiplication of an invariant monoid is a pointed multilinear map by Lemma 5.2.1, so $\boldsymbol{N}$ is a synthetic monoid object in $\boldsymbol{\mathcal{C}^T}$. As $\boldsymbol{L}$ is synthetic monoidal, it maps the synthetic monoid $\boldsymbol{N} \in \boldsymbol{\mathcal{C}^T}$ to the synthetic monoid $\boldsymbol{LN} \in \boldsymbol{\mathcal{V}}$ by Proposition 6.1.2. Its underlying object is $L(N, a\colon N \otimes I \to N) \in \mathcal{V}$. The mappings form an equivalence, as $M \in \mathcal{V} \mapsto GM \in \mathcal{C} \mapsto KM \in \mathcal{C}^T \mapsto LKM \in \mathcal{V} \cong M$, and conversely $M \in \mathcal{C} \mapsto (N, a) \mapsto \mathcal{C}^T \mapsto L(N, a) \in \mathcal{V} \mapsto GL(N, a) \in \mathcal{C} \cong U(N, a) = N$. □

To relate algebraic monoids, we need to make further assumptions on the signature endofunctors $\Sigma\colon \mathcal{C} \to \mathcal{C}$ and $\Omega\colon \mathcal{V} \to \mathcal{V}$. Given the forgetful functor $\boldsymbol{\mathcal{C}^T} \to \mathcal{C}$, we can define the right $\boldsymbol{\mathcal{C}^T}$-action $(-) \ominus (=)\colon \mathcal{C} \times \boldsymbol{\mathcal{C}^T} \to \mathcal{C}$ as $X \ominus \boldsymbol{Y} \triangleq X \oplus Y$ and consider $\mathcal{C}$ to be an Id-relative synthetic $\boldsymbol{\mathcal{C}^T}_{|\mathcal{C}}$-modular category, with profunctor $\mathcal{S}(X, \boldsymbol{Y}; Z) \triangleq \mathcal{C}(X \oplus Y, Z)$. A synthetic modular endofunctor on $\mathcal{C}$ then has the strength operator $s^{\Sigma}[-]\colon \mathcal{C}(X \oplus Y, Z) \to \mathcal{C}(\Sigma X \oplus Y, \Sigma Z)$, with associativity law given in terms of pointed multilinear maps (as the modular category is over $\boldsymbol{\mathcal{C}^T}_{|\mathcal{C}}$). Similarly, a $\boldsymbol{\mathcal{V}}$-modular functor $\Omega\colon \mathcal{V} \to \mathcal{V}$ with strength $s^{\Omega}_{A,\boldsymbol{B}}\colon \Omega A \oslash \boldsymbol{B} \to \Omega(A \oslash \boldsymbol{B})$ (for $A \oslash \boldsymbol{B} \triangleq A \otimes B$) can be turned into a representable synthetic $\boldsymbol{\mathcal{V}}$-modular endofunctor, with strength operator $s^{\Omega}[-]\colon \mathcal{V}(A \oslash \boldsymbol{B}, C) \to \mathcal{V}(\Omega A \oslash \boldsymbol{B}, \Omega C)$ that maps $f\colon A \oslash \boldsymbol{B} \to C$ to $\Omega A \oslash \boldsymbol{B} \xrightarrow{s^{\Omega}_{A,\boldsymbol{B}}} \Omega(A \oslash \boldsymbol{B}) \xrightarrow{\Omega f} \Omega C$.

Theorem 7.2.1

If two unital endofunctors $(\Sigma\colon \mathcal{C} \to \mathcal{C}, \eta^{\Sigma}\colon \mathrm{Id} \Longrightarrow \Sigma)$ *and* $(\Omega\colon \mathcal{V} \to \mathcal{V}, \eta^{\Omega}\colon \mathrm{Id} \Longrightarrow \Omega)$ *form 1-cells and 2-cells in* **SynMod**

$$(\mathrm{id}, \eta^{\Sigma})\colon (\mathrm{Id}, \mathrm{Id}) \Longrightarrow (\mathrm{Id}, \Sigma) : (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}}) \to (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}})$$
$$(\mathrm{id}, \eta^{\Omega})\colon (\mathrm{Id}, \mathrm{Id}) \Longrightarrow (\mathrm{Id}, \Omega)\colon (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}}) \to (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}})$$

$(\boldsymbol{K}, G)\colon (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}}) \to (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}})$ *is a strong elevator between them, and* $\boldsymbol{L}\colon \mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}} \to \boldsymbol{\mathcal{V}}_{|\mathcal{V}}$ *is synthetic monoidal, then the categories of* Ω*-monoids in* $\mathcal{V}$ *and* Σ*-monoids in* $\mathcal{C}$ *are equivalent.*

Proof See the Appendix on page 334. □

The main value of the theorem is in giving confidence that the structure of second-order syntax with algebraic structure can be equivalently represented both in presheaves (as understood from the presheaf model) and families as well. Combined with the other core results of this thesis – the initial algebra-lifting theorem, and the free Σ-monoid theorem – we obtain a full and precise correspondence between the two models.

In the next chapter we move away from abstract skew-monoidal structure and introduce the components of the familial model.

§ Summary of Part II

This part of the thesis focused on skew-monoidal closed categories and related notions, including modular categories, warpings, and synthetic monoidal structure. Part III puts these pieces together to precisely axiomatise the relationship between the presheaf and familial models, culminating with the derivation of the free Σ-monoid structure for the datatype of terms via initial-algebra semantics.

PART I

THE FAMILIAL MODEL

Parts I and II established the abstract categorical groundwork: categories of algebras, clones, skew-monoidal structures, and modular functors. With these in place, Part III turns to the familial model of second-order abstract syntax, grounding it in the well-understood presheaf setting.

We begin in Chapter 8 with a review of presheaves over small categories, highlighting properties relevant to our model-theoretic framework. Since indexed families are presheaves on discrete categories, we then explore substitution from abstract principles to its concrete realisation in Chapter 9. Chapter 10 introduces the familial model proper, detailing the constructions and proofs used to reconstruct the presheaf theory within this restricted setting. The culmination comes in Chapter 11, where we derive the algebraic monoid structure of the initial syntactic algebra and examine the second-order features of the resulting formalism.

CHAPTER 8

Presheaves

Though a simple concept on the surface, presheaves – or **Set**-valued functors – have a rich and extensive theory that permeates the study of categories (Bunge, 1966; Reyes et al., 2004; MacLane and Moerdijk, 2012). In this section we briefly recall some of the core constructions on presheaves, covering their categorical structure and universal properties (Section 8.2), and the notions of nerve and realisation that underlie the substitution structure in the subsequent Section 8.3. Before that, we introduce a useful categorical tool for manipulating presheaves.

8.1 Calculus of categories

We review a powerful categorical formalism for defining constructions and presenting proofs in an elegant, calculational manner: co/ends and their calculus. Kan extensions – a direct application of co/ends – form the backbone of much of category theory and feature extensively in our work as well. A detailed account of many concepts in this chapter is given by Loregian (2021), so we restrict ourselves to the main definitions and results in order to fix the notation.

Definition 8.1.1 A *covariant presheaf* P on a small category $\mathbb{C}$ is a functor $\mathbb{C} \to \mathbf{Set}$. The category of covariant presheaves and natural transformations is denoted $\widetilde{\mathbb{C}} = \mathbf{Set}^{\mathbb{C}}$.
A *contravariant presheaf* is a functor $\mathbb{C}^{\mathrm{op}} \to \mathbf{Set}$. The category of contravariant presheaves and natural transformations is denoted $\widehat{\mathbb{C}} = \mathbf{Set}^{\mathbb{C}^{\mathrm{op}}}$. ⌟

Notation. In this and subsequent chapters, objects of the base category $\mathbb{C}$ will be written in lowercase, and presheaves in uppercase. ⌟

Definition 8.1.2 The *covariant Yoneda embedding* is the functor $\text{ℱ}\colon \mathbb{C}^{\mathrm{op}} \to \widetilde{\mathbb{C}}$, mapping an object $a \in \mathbb{C}$ to the hom-functor $\mathbb{C}(a, -)\colon \mathbb{C} \to \mathbf{Set}$. The *contravariant Yoneda embedding* is the functor $\text{よ}\colon \mathbb{C} \to \widehat{\mathbb{C}}$, mapping $b \in \mathbb{C}$ to $\mathbb{C}(-, b)$. ⌟

Lemma 8.1.1 (Yoneda lemma) *For any presheaves $P\colon \widetilde{\mathbb{C}}$ and $Q\colon \widehat{\mathbb{C}}$ and objects $a \in \mathbb{C}$, we have the natural isomorphisms*

$$\widetilde{\mathbb{C}}(\text{ℱ}a, P) \cong P(a) \qquad \widehat{\mathbb{C}}(\text{よ}a, Q) \cong Q(a) \tag{ℱ≅}$$

Lemma 8.1.2 *The Yoneda embedding is fully faithful: $a \cong b$ iff $\yo a \cong \yo b \in \widetilde{\mathbb{C}}$ iff $\yoneda a \cong \yoneda b \in \widehat{\mathbb{C}}$.*

8.1.1 Co/ends

Co/ends are both a generalisation and an instance of co/limits, defined in terms of dinatural transformations, the appropriate notion of naturality for functors of mixed variance.

Definition 8.1.3 A *dinatural transformation* $\alpha : P \Longrightarrow Q$ between functors $P, Q : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ is a family of morphisms

$$\alpha^a : P(a,a) \to Q(a,a) : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$$

such that for all $f : a \to b \in \mathcal{C}$, the following hexagon commutes:

$$\begin{array}{ccccc} P(a,a) & \xleftarrow{P(f,a)} & P(b,a) & \xrightarrow{P(b,f)} & P(b,b) \\ {\scriptstyle \alpha^a}\downarrow & & & & \downarrow{\scriptstyle \alpha^b} \\ Q(a,a) & \xrightarrow[Q(f,a)]{} & Q(b,a) & \xleftarrow[Q(b,f)]{} & Q(b,b) \end{array}$$

⌟

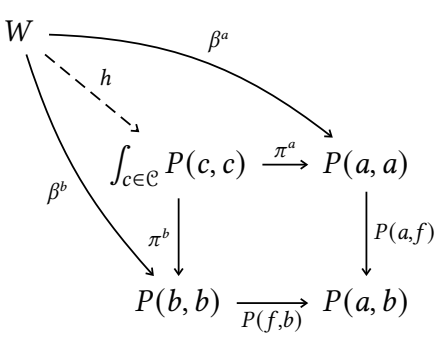
Since co/ends intend to generalise co/limits, we also have an analogue of co/cones as natural transformations between a diagram and the constant functor.

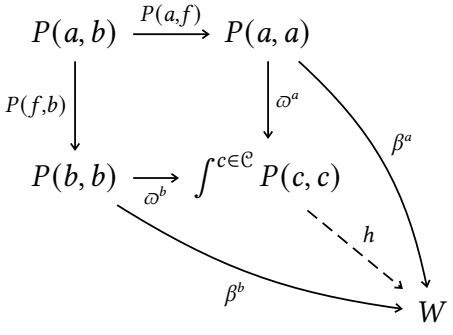
Definition 8.1.4 A *wedge* of a mixed variance functor $P : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ is a dinatural transformation $\Delta_A \Longrightarrow P : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ from the constant functor $\Delta_A : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D} = \Delta_A(\_,\_) = A$ which we may also simply write A. Dually, a *cowedge* is a dinatural transformation $P \Longrightarrow A$. The value A of the constant bivariant functor is called the *apex* of the co/wedge.

A morphism of co/wedges on the same functor P is a morphism of the apices that commutes with the dinatural transformation; accordingly, co/wedges over P form a category. ⌟

Definition 8.1.5 The *end* of a functor $P : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ is its terminal wedge, with apex denoted $\mathrm{end}(P)$ or $\int_{a \in \mathcal{C}} P(a,a)$ and dinatural transformation $\pi : \left(\int_{a \in \mathcal{C}} P(a,a)\right) \Longrightarrow P$. The *coend* of P is the terminal cowedge, with apex denoted $\mathrm{coend}(P)$ or $\int^{a \in \mathcal{C}} P(a,a)$ and dinatural transformation $\varpi : P \Longrightarrow \left(\int^{a \in \mathcal{C}} P(a,a)\right)$. ⌟

The dinaturality and universality condition of co/ends can be pictured as follows: for any co/wedge β of P with apex W, there is a unique co/wedge map h between it and the co/end.





The definition of co/ends is a mouthful and rather hard to unpack, but they admit several properties that allow us to calculate isomorphisms directly and in a satisfyingly elegant manner.

The practical benefit of this *calculus of co/ends* is that we rarely have to think about co/wedges, apices, or even dinaturality in its full "hexagonal" form, instead turning proofs into symbolic, calculational arguments. We now lay out the essential elements of the co/end calculus with brief proof sketches where instructive – for details, see Loregian (2021, Chapters 1-2).

Proposition 8.1.1 (Naturality) *For $F, G \colon \mathbb{C} \to \mathcal{D}$, the set of natural transformations $F \Longrightarrow G$ can be expressed as:*

$$\mathcal{D}^{\mathbb{C}}(F,G) \cong \int_{c\in\mathbb{C}} \mathcal{D}(Fc, Gc) \tag{$\int$NT}$$

Proposition 8.1.2 (Continuity) *For $P \colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathcal{D}$:*

$$\mathcal{D}\big(a, \textstyle\int_{c\in\mathbb{C}} P(c,c)\big) \cong \int_{c\in\mathbb{C}} \mathcal{D}(a, P(c,c)) \tag{$\int\leftrightarrow$}$$

$$\mathcal{D}\big(\textstyle\int^{c\in\mathbb{C}} P(c,c), b\big) \cong \int_{c\in\mathbb{C}} \mathcal{D}(P(c,c), b) \tag{$\int\updownarrow$}$$

Proof Co/ends are co/limits and are therefore preserved/reversed by the hom-functor. □

Corollary 8.1.1 (Ninja Yoneda lemmas) *For presheaves $P \in \widetilde{\mathbb{C}}$ and $Q \in \widehat{\mathbb{C}}$,*

$$P \cong \int_{c\in\mathbb{C}} \text{よ}c \Rightarrow Pc \qquad Q \cong \int_{c\in\mathbb{C}} \text{ヲ}c \Rightarrow Qc \tag{$\int$ヲ$\Rightarrow$}$$

$$P \cong \int^{c\in\mathbb{C}} \text{ヲ}c \times Pc \qquad Q \cong \int^{c\in\mathbb{C}} \text{よ}c \times Qc \tag{$\int$ヲ$\times$}$$

Proof Expanding the RHS of the first isomorphism, we have

$$\int_{c\in\mathbb{C}} \mathbb{C}(a,c) \Rightarrow Pc \cong \int_{c\in\mathbb{C}} \mathbf{Set}(\mathbb{C}(a,c), Pc)$$

and since ends are natural transformations this is isomorphic to the instance of the standard Yoneda lemma

$$\int_{c\in\mathbb{C}} \mathbf{Set}(\mathbb{C}(a,c), Pc) \cong \mathbf{Set}^{\mathbb{C}}(\mathbb{C}(a,-), P) \cong Pa$$

To establish the second isomorphism, we reason via the isomorphism of the Yoneda embeddings: $\mathbf{Set}\big(\int^{c\in\mathbb{C}} \mathbb{C}(c,d) \times Pc, Y\big) \cong \mathbf{Set}(Pd, Y)$.

$$\begin{aligned}
\mathbf{Set}\Big(\int^{c\in\mathbb{C}} \mathbb{C}(c,d) \times Pc, Y\Big) &\cong \int_{c\in\mathbb{C}} \mathbf{Set}\big(\mathbb{C}(c,d) \times Pc, Y\big) && (\int\updownarrow)\\
&\cong \int_{c\in\mathbb{C}} \mathbf{Set}\big(\mathbb{C}(c,d), \mathbf{Set}(Pc, Y)\big) && (\text{currying})\\
&\cong \mathbf{Set}(Pd, Y) && (\int\text{ヲ}\times)
\end{aligned}$$

□

Remark. The presentation of the lemmas above favours syntactic elegance to notational rigour: since よc is a presheaf and Pc is a set, their cartesian product or function space is not defined.

More appropriate would be to write

$$P(-) \cong \int_{c\in\mathbb{C}} \text{よ}(c)(-) \Rightarrow Pc = \int_{c\in\mathbb{C}} \mathbb{C}(-,c) \Rightarrow Pc$$
$$P(-) \cong \int^{c\in\mathbb{C}} \text{ℱ}(c)(-) \times Pc \cong \int_{c\in\mathbb{C}} \mathbb{C}(c,-) \times Pc$$

making it clear how the right-hand side is itself a presheaf with index placeholder $(-)$. Alternatively, we may formally write

$$P \cong \int^{c\in\mathbb{C}} Pc \cdot \text{ℱ}c \qquad Q \cong \int^{c\in\mathbb{C}} Qc \cdot \text{よ}c$$

using the fact that presheaves are **Set**-copowered; this will be relevant in Section 8.2. ⌟

Below is a useful generalisation of cartesian products and function sets which interacts nicely with co/ends.

Definition 8.1.6 Given a co/complete category $\mathcal{C}$, the *power* (resp. *copower*) of an object $A \in \mathcal{C}$ by a set $X \in \mathbf{Set}$ is an object $X \pitchfork A \in C$ (resp. $X \cdot A \in \mathcal{C}$) satisfying for all $B \in \mathcal{C}$:

$$\mathcal{C}(B, X \pitchfork A) \cong \mathbf{Set}(X, \mathcal{C}(B,A)) \quad (\pitchfork\cong) \qquad \mathcal{C}(X \cdot A, B) \cong \mathbf{Set}(X, \mathcal{C}(A,B)) \quad (\cdot\cong)$$

⌟

Corollary 8.1.2 (Ninja Yoneda lemmas for co/powers) *For $F\colon \mathbb{C} \to \mathcal{D}$ and $K\colon \mathbb{C}^{\mathrm{op}} \to \mathcal{D}$, if the relevant co/powers exist,*

$$F \cong \int_{c\in\mathbb{C}} \text{よ}c \pitchfork Fc \qquad K \cong \int_{c\in\mathbb{C}} \text{ℱ}c \pitchfork Kc \quad (\int\text{ℱ}\pitchfork)$$
$$F \cong \int^{c\in\mathbb{C}} \text{ℱ}c \cdot Fc \qquad K \cong \int^{c\in\mathbb{C}} \text{よ}c \cdot Kc \quad (\int\text{ℱ}\cdot)$$

Proof Take $D \in \mathcal{D}$ and $a \in \mathbb{C}$, and calculate as follows:

$$\begin{aligned} \mathcal{D}(D, \int_{c\in\mathbb{C}} \mathbb{C}(a,c) \pitchfork Fc) &\cong \int_{c\in\mathbb{C}} \mathcal{D}(D, \mathbb{C}(a,c) \pitchfork Fc) && (\int\leftrightarrow)\\ &\cong \int_{c\in\mathbb{C}} \mathbf{Set}(\mathbb{C}(a,c), \mathcal{D}(D,Fc)) && (\pitchfork\cong)\\ &\cong \mathcal{D}(D,Fa) && (\int\text{ℱ}\Rightarrow) \end{aligned}$$

By the isomorphism of the Yoneda embeddings, $\int_{c\in\mathbb{C}} \mathbb{C}(a,c) \pitchfork Fc \cong Fa$, as required. The ninja Yoneda lemma for copowers $Fa \cong \int^{c\in\mathbb{C}} \mathbb{C}(c,a) \cdot Fc$ follows via a similar reasoning as above:

$$\mathcal{D}\Big(\int^{c\in\mathbb{C}} \mathbb{C}(c,a) \cdot Fc, D\Big) \cong \int_{c\in\mathbb{C}} \mathcal{D}(\mathbb{C}(c,a) \cdot Fc, D) \cong \int_{c\in\mathbb{C}} \mathbf{Set}(\mathbb{C}(c,a), \mathcal{D}(Fc,D)) \cong \mathcal{D}(Fa,D)$$

□

Proposition 8.1.3 (Fubini Rule) *For $P\colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \times \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathcal{E}$, when all relevant co/ends exist,*

$$\int_{(a,b)\in\mathbb{C}\times\mathbb{D}} P(a,a,b,b) \cong \int_{a\in\mathbb{C}}\int_{b\in\mathbb{D}} P(a,a,b,b) \cong \int_{b\in\mathbb{D}}\int_{a\in\mathbb{C}} P(a,a,b,b) \tag{$\int\int_{\bullet}$}$$

$$\int^{(a,b)\in\mathbb{C}\times\mathbb{D}} P(a,a,b,b) \cong \int^{a\in\mathbb{C}}\int^{b\in\mathbb{D}} P(a,a,b,b) \cong \int^{b\in\mathbb{D}}\int^{a\in\mathbb{C}} P(a,a,b,b) \tag{$\int\int^{\bullet}$}$$

Proof See Loregian (2021, Theorem 1.3.1) and Loregian (2019). □

Co/complete categories are always **Set**-co/powered, with

$$X \cdot A \triangleq \coprod_{x\in X} A \qquad X \pitchfork A \triangleq \prod_{x\in X} A$$

The following identities could be computed through these definitions, but the alternative proof showcases the convenience of the co/end calculus.

Corollary 8.1.3 *For $\mathcal{D}$ copowered by* **Set**, *functors $P\colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbf{Set}$ and $Q\colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathcal{D}$, $A, B \in \mathcal{D}$, $X \in \mathbf{Set}$, we have:*

$$\mathrm{coend}(P) \pitchfork B \cong \int_{c\in\mathbb{C}} P(c,c) \pitchfork B \qquad X \pitchfork \mathrm{end}(Q) \cong \int_{c\in\mathbb{C}} X \pitchfork Q(c,c) \tag{$\int\pitchfork$}$$

$$\mathrm{coend}(P) \cdot A \cong \int^{c\in\mathbb{C}} P(c,c) \cdot A \qquad X \cdot \mathrm{coend}(Q) \cong \int^{c\in\mathbb{C}} X \cdot Q(c,c) \tag{$\int\cdot$}$$

Proof By the Yoneda isomorphism, we have the following calculations:

$$\begin{aligned}
&\mathcal{D}\big(A, \mathrm{coend}(P) \pitchfork B\big) \\
&\cong \mathbf{Set}\big(\mathrm{coend}(P), \mathcal{D}(A,B)\big) && (\pitchfork\cong)\\
&\cong \int_{c\in\mathbb{C}} \mathbf{Set}\big(P(c,c), \mathcal{D}(A,B)\big) && (\int\updownarrow)\\
&\cong \int_{c\in\mathbb{C}} \mathcal{D}\big(A, P(c,c) \pitchfork B\big) && (\pitchfork\cong)\\
&\cong \mathcal{D}\Big(A, \int_{c\in\mathbb{C}} P(c,c) \pitchfork B\Big) && (\int\leftrightarrow)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{D}\big(A, X \pitchfork \mathrm{end}(Q)\big) \\
&\cong \mathbf{Set}\big(X, \mathcal{D}\big(A, \mathrm{end}(Q)\big)\big) && (\pitchfork\cong)\\
&\cong \int_{c\in\mathbb{C}} \mathbf{Set}\big(X, \mathcal{D}(A, Q(c,c))\big) && (\int\leftrightarrow)\\
&\cong \int_{c\in\mathbb{C}} \mathcal{D}\big(A, X \pitchfork Q(c,c)\big) && (\pitchfork\cong)\\
&\cong \mathcal{D}\Big(A, \int_{c\in\mathbb{C}} X \pitchfork Q(c,c)\Big) && (\int\leftrightarrow)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{D}\big(\mathrm{coend}(P) \cdot A, B\big) \\
&\cong \mathbf{Set}\big(\mathrm{coend}(P), \mathcal{D}\big(A,B\big)\big) && (\cdot\cong)\\
&\cong \int_{c\in\mathbb{C}} \mathbf{Set}\big(P(c,c), \mathcal{D}(A,B)\big) && (\int\updownarrow)\\
&\cong \int_{c\in\mathbb{C}} \mathcal{D}\big(P(c,c) \cdot A, B\big) && (\cdot\cong)\\
&\cong \mathcal{D}\Big(\int^{c\in\mathbb{C}} P(c,c) \cdot A, B\Big) && (\int\updownarrow)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{D}\big(X \cdot \mathrm{coend}(Q), B\big) \\
&\cong \mathbf{Set}\big(X, \mathcal{D}(\mathrm{coend}(Q), B)\big) && (\cdot\cong)\\
&\cong \mathbf{Set}\Big(X, \int_{c\in\mathbb{C}} \mathcal{D}\big(Q(c,c), B\big)\Big) && (\int\updownarrow)\\
&\cong \int_{c\in\mathbb{C}} \mathbf{Set}\big(X, \mathcal{D}(Q(c,c), B)\big) && (\int\leftrightarrow)\\
&\cong \int_{c\in\mathbb{C}} \mathcal{D}\big(X \cdot Q(c,c), B\big) && (\cdot\cong)\\
&\cong \mathcal{D}\Big(\int^{c\in\mathbb{C}} X \cdot Q(c,c), B\Big) && (\int\updownarrow)
\end{aligned}$$

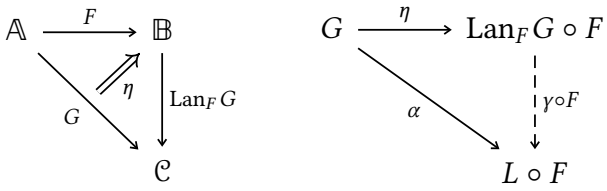
□

An important example of constructions based on co/ends is next.

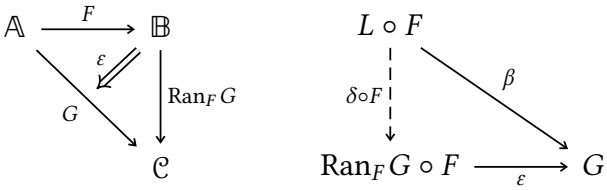
8.1.2 Kan extensions

Kan extensions are often regarded as the "most universal" construction in category theory – their constitutional role in the field is captured by the slogan "all concepts are Kan extensions": see Mac Lane (1971, Chapter IX), Riehl (2017, Chapter 6), Lehner (2014). Indeed, many fundamental definitions in our theory will be expressible as a Kan extension, allowing us to prove important results in highly abstract ways. We only recall the essential properties and definitions used in the thesis, without trying to give a fully comprehensive account.

Definition 8.1.7 Given functors $F\colon \mathbb{A} \to \mathbb{B}$ and $G\colon \mathbb{A} \to \mathcal{C}$, the *left Kan extension* of G along F is the functor $\mathrm{Lan}_F G\colon \mathbb{B} \to \mathcal{C}$ together with a unit $\eta\colon G \Longrightarrow \mathrm{Lan}_F G \circ F$ satisfying the following universality property: for any other extension $(L\colon \mathbb{B} \to \mathcal{C}, \alpha\colon G \Longrightarrow L \circ F)$, there exists a unique natural transformation $\gamma\colon \mathrm{Lan}_F G \Longrightarrow L$ such that $\alpha = \gamma F \circ \eta$.



Dually, a *right Kan extension* of G along F is the functor $\mathrm{Ran}_F G\colon \mathbb{B} \to \mathcal{C}$ together with a counit $\varepsilon\colon \mathrm{Ran}_F G \circ F \Longrightarrow G$ satisfying the following universality property: for any other extension $(R\colon \mathbb{B} \to \mathcal{C}, \beta\colon R \circ F \Longrightarrow G)$, there exists a unique $\delta\colon R \Longrightarrow \mathrm{Ran}_F G$ such that $\beta = \varepsilon \circ \delta F$.



⌟

Example 8.1.1. Co/limits are instances of Kan extensions of a functor $G\colon \mathbb{A} \to \mathcal{C}$ along the canonical functor $!\colon \mathbb{A} \to \top$ into the terminal category (Mac Lane, 1971, Theorem X.7.1). ⌟

We now assume that every functor $\mathbb{A} \to \mathcal{C}$ extends along $F\colon \mathbb{A} \to \mathbb{B}$ (which is the case if $\mathcal{C}$ is co/complete), implying that $\mathrm{Lan}_F, \mathrm{Ran}_F : \mathcal{C}^{\mathbb{A}} \to \mathcal{C}^{\mathbb{B}}$ are functorial.

Notation. We will also use the notation

$$\blacklozenge_F\colon \mathcal{C}^{\mathbb{A}} \to \mathcal{C}^{\mathbb{B}} \qquad \bigstar_F\colon \mathcal{C}^{\mathbb{B}} \to \mathcal{C}^{\mathbb{A}} \qquad \text{and} \qquad \blacksquare_F\colon \mathcal{C}^{\mathbb{A}} \to \mathcal{C}^{\mathbb{B}}$$

for the left Kan extension, precomposition, and right Kan extension functors, respectively, omitting the subscripts if the base functor (F throughout this section) is clear from context. ⌟

Theorem 8.1.1

The Kan extensions along $F\colon \mathbb{A} \to \mathbb{B}$ are two-sided adjoints of the precomposition functor $(-) \circ F = \bigstar\colon \mathcal{C}^{\mathbb{B}} \to \mathcal{C}^{\mathbb{A}}$:

$$\mathrm{Lan}_F \dashv \bigstar \dashv \mathrm{Ran}_F$$

Proof The following hom-set isomorphism for all $G\colon \mathbb{A} \to \mathcal{C}$ and $L\colon \mathbb{B} \to \mathcal{C}$

$$\mathcal{C}^{\mathbb{B}}(\mathrm{Lan}_F G, L) \cong \mathcal{C}^{\mathbb{A}}(G, L \circ F) \tag{Lan ⊣}$$

is nothing but the universal property of the left Kan extension: any candidate extension $(L\colon \mathbb{B} \to \mathcal{C}, \alpha\colon G \Longrightarrow LF)$ factors through the unit $\eta\colon G \Longrightarrow \mathrm{Lan}_F G$ via a unique natural transformation $\gamma\colon \mathrm{Lan}_F G \Longrightarrow L$, giving the bijective correspondence above. The universal property of the right Kan extension dually gives the isomorphism

$$\mathbb{B}^{\mathbb{A}}(R \circ F, G) \cong \mathcal{C}^{\mathbb{B}}(R, \mathrm{Ran}_F G) \tag{⊣Ran}$$

which establish the required adjunctions. □

A direct application of this theorem is the following preservation property of Kan extensions:

Proposition 8.1.4 *Left/right adjoints preserve left/right Kan extensions.*

Proof Let $L\colon \mathcal{C} \to \mathcal{D}$ be left adjoint to $R\colon \mathcal{D} \to \mathcal{C}$. We show that $L \circ \mathrm{Lan}_F G = \mathrm{Lan}_F (L \circ G)$ using the following calculation of adjoint transposes:

$$\begin{aligned}
\mathcal{D}^{\mathbb{B}}\big(L \circ \mathrm{Lan}_F G, K\big) &\cong \mathcal{C}^{\mathbb{B}}\big(\mathrm{Lan}_F G, RK\big) && (L \dashv R)\\
&\cong \mathcal{C}^{\mathbb{A}}\big(G, RKF\big) && (\mathrm{Lan} \dashv)\\
&\cong \mathcal{D}^{\mathbb{A}}\big(LG, KF\big) && (L \dashv R)\\
&\cong \mathcal{D}^{\mathbb{B}}\big(\mathrm{Lan}_F (LG), K\big) && (\mathrm{Lan} \dashv)
\end{aligned}$$

The dual calculation shows that right adjoints preserve right Kan extensions. □

This theorem, along with uniqueness of adjoints, allows us to find a concrete formula to calculate Kan extensions in terms of co/ends.

Theorem 8.1.2

Let $F\colon \mathbb{A} \to \mathbb{B}$ and $G\colon \mathbb{A} \to \mathcal{C}$, with $\mathcal{C}$ cocomplete/complete. Then, the left/right Kan extensions of G along F satisfy the isomorphism:

$$\mathrm{Lan}_F G \cong \int^{a \in \mathbb{A}} \mathbb{B}(Fa, -) \cdot Ga \tag{Lan∫} \qquad \mathrm{Ran}_F G \cong \int_{a \in \mathbb{A}} \mathbb{B}(-, Fa) \pitchfork Ga \tag{Ran∫}$$

Proof By Theorem 8.1.1, if $\mathrm{Lan}_F G$ or $\mathrm{Ran}_F G$ exist, they must be left/right adjoint to the precomposition functor $\star$ – thus, we merely need to show that the co/end formulae above satisfy the required hom-set isomorphisms. The proofs are elegant demonstrations of the convenience of the co/end calculus.

$$
\begin{aligned}
& \mathcal{C}^{\mathbb{B}}\Big(\int^{a\in\mathbb{A}} \mathbb{B}(Fa,-)\cdot Ga, L\Big) && && \mathcal{C}^{\mathbb{B}}\Big(R, \int_{a\in\mathbb{A}} \mathbb{B}(-,Fa) \pitchfork Ga\Big) && \\
\cong & \int_{b\in\mathbb{B}} \mathcal{C}\Big(\int^{a\in\mathbb{A}} \mathbb{B}(Fa,b)\cdot Ga, Lb\Big) && (\int \mathrm{NT}) & \cong & \int_{b\in\mathbb{B}} \mathcal{C}\Big(Rb, \int_{a\in\mathbb{A}} \mathbb{B}(b,Fa) \pitchfork Ga\Big) && (\int \mathrm{NT}) \\
\cong & \int_{b\in\mathbb{B}} \int_{a\in\mathbb{A}} \mathcal{C}\big(\mathbb{B}(Fa,b)\cdot Ga, Lb\big) && (\int\updownarrow) & \cong & \int_{b\in\mathbb{B}} \int_{a\in\mathbb{A}} \mathcal{C}(Rb, \mathbb{B}(b,Fa) \pitchfork Ga) && (\int\leftrightarrow) \\
\cong & \int_{b\in\mathbb{B}} \int_{a\in\mathbb{A}} \mathbf{Set}\big(\mathbb{B}(Fa,b), \mathcal{C}(Ga,Lb)\big) && (\cdot\cong) & \cong & \int_{b\in\mathbb{B}} \int_{a\in\mathbb{A}} \mathbf{Set}\big(\mathbb{B}(b,Fa), \mathcal{C}(Rb,Ga)\big) && (\pitchfork\cong) \\
\cong & \int_{a\in\mathbb{A}} \int_{b\in\mathbb{B}} \mathbf{Set}\big(\mathbb{B}(Fa,b), \mathcal{C}(Ga,Lb)\big) && (\int\int_{\bullet}) & \cong & \int_{a\in\mathbb{A}} \int_{b\in\mathbb{B}} \mathbf{Set}\big(\mathbb{B}(b,Fa), \mathcal{C}(Rb,Ga)\big) && (\int\int_{\bullet}) \\
\cong & \int_{a\in\mathbb{A}} \mathbf{Set}^{\mathbb{B}}\big(\text{Ɏ}_{\mathbb{B}}(Fa), \mathcal{C}(Ga,L-)\big) && (\int \mathrm{NT}) & \cong & \int_{a\in\mathbb{A}} \mathbf{Set}^{\mathbb{B}}\big(\text{ɟ}_{\mathbb{B}}(Fa), \mathcal{C}(R-,Ga)\big) && (\int \mathrm{NT}) \\
\cong & \int_{a\in\mathbb{A}} \mathcal{C}(Ga, LFa) && (\int\text{Ɏ}\Rightarrow) & \cong & \int_{a\in\mathbb{A}} \mathcal{C}(RFa, Ga) && (\int\text{Ɏ}\Rightarrow) \\
\cong & \mathcal{C}^{\mathbb{A}}(G, L\circ F) && (\int \mathrm{NT}) & \cong & \mathbb{B}^{\mathbb{A}}(R\circ F, G) && (\int \mathrm{NT})
\end{aligned}
$$

Since adjoints are unique up to isomorphism, we have that the co/end formulae are isomorphic to the Kan extensions. □

With these prerequisites in place, we are ready to discuss presheaves and their structure.

8.2 Categorical structures

Presheaves carry over much of the categorical structure of sets through pointwise application, and Yoneda serves as a fully faithful embedding of $\mathbb{C}$ in the category $\widetilde{\mathbb{C}}$. Nevertheless, some of the categorical structure of $\widetilde{\mathbb{C}}$ is subtle and presheaf-specific.

8.2.1 Bicartesian closure

For a small category $\mathbb{C}$, the covariant presheaf category $\widetilde{\mathbb{C}}$ is bicartesian closed. Products and coproducts are given pointwise:

$$(P\times Q)(a) \triangleq Pa\times Qa \in \mathbf{Set} \qquad (P+Q)(a) \triangleq Pa+Qa \in \mathbf{Set}$$

but presheaf exponentials need more thought.

Definition 8.2.1 The exponential of presheaves $P, Q \in \widetilde{\mathbb{C}}$ is

$$(P \supset Q)(a) \triangleq \widetilde{\mathbb{C}}\big(\text{Ɏ}a\times P, Q\big) \cong \int_{b\in\mathbb{C}} \mathbf{Set}(\mathbb{C}(a,b)\times Pb, Qb) \cong \int_{b\in\mathbb{C}} \mathbf{Set}(\mathbb{C}(a,b), Pb\Rightarrow Qb)$$

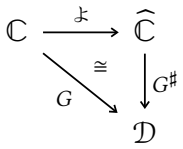
⌟

Notation. When dealing with presheaves over multiple categories, we will use the subscript of the base category to disambiguate where an operation is taken: e.g. we will write $P \supset_{\mathbb{C}} Q$ to denote the exponential of presheaves in $\widetilde{\mathbb{C}}$. ⌟

Presheaves are a particularly important categorical construction as they inherit the rich categorical structure of sets. They also satisfy the following important universal property.

Theorem 8.2.1

$\widehat{\mathbb{C}}$ is the free cocompletion of $\mathbb{C}$: given any cocomplete category $\mathcal{D}$ and functor $G\colon \mathbb{C} \to \mathcal{D}$, there is an essentially unique cocontinuous extension $G^\sharp\colon \widehat{\mathbb{C}} \to \mathcal{D}$, exhibiting an equivalence of categories $\mathcal{D}^{\mathbb{C}}$ and **Cocont**$(\widehat{\mathbb{C}}, \mathcal{D})$*:*



Proof The extension $G^\sharp$ is nothing but the left Kan extension along よ. This can be derived by "wishful thinking": if $G^\sharp$ were to exist, it must be ① cocontinuous and satisfy ② $G^\sharp \circ よ \cong G$. By the ninja Yoneda lemma, every presheaf P can be expressed as a coend of a copower of the Yoneda embedding, so one may calculate the action of $G^\sharp$ on a presheaf $P \in \widehat{\mathbb{C}}$ as follows:

$$
\begin{aligned}
G^\sharp(P) &\cong G^\sharp\Big(\int^{c\in\mathbb{C}} Pc \cdot よc\Big) && (\int ヲ \times)\\
&\cong \int^{c\in\mathbb{C}} G^\sharp(Pc \cdot よc) && (①, \text{coend is colimit})\\
&\cong \int^{c\in\mathbb{C}} Pc \cdot G^\sharp(よc) && (①, \text{copower is colimit})\\
&\cong \int^{c\in\mathbb{C}} Pc \cdot Gc && (②)\\
&\cong \int^{c\in\mathbb{C}} \widetilde{\mathbb{C}}(ヲc, P) \cdot Gc && (ヲ \cong)\\
&\cong \mathrm{Lan}_ヲ G(P) && (\mathrm{Lan}\int)
\end{aligned}
$$

Universality is guaranteed as $G^\sharp \cong \mathrm{Lan}_よ G$ is a left Kan extension, so is initial among all cocontinuous extensions of G. The intermediate representation $G^\sharp(P) \cong \int^{c\in\mathbb{C}} Pc \cdot Gc$ – also known as a *realisation* – will be used extensively in the next section. □

8.2.2 Monoidal closure

The bicartesian structure of presheaves is inherited from that of sets, often by pointwise indexing (in the case of co/limits). If the base category $(\mathbb{C}, \bullet, i)$ is monoidal, presheaves on it can also be equipped with an additional monoidal closed structure.

Definition 8.2.2 The *Day convolution tensor product* $\circledcirc\colon \widetilde{\mathbb{C}} \times \widetilde{\mathbb{C}} \to \widetilde{\mathbb{C}}$ is defined as the coend

$$(P \circledcirc Q)(a) \triangleq \int^{a_1,a_2\in\mathbb{C}} \mathbb{C}(a_1 \bullet a_2, a) \times Pa_1 \times Qa_2$$

⌟

Definition 8.2.3 The *Day convolution hom* $(-) \multimap (=)\colon \widetilde{\mathbb{C}}^{\mathrm{op}} \times \widetilde{\mathbb{C}} \to \widetilde{\mathbb{C}}$ is defined as the end

$$(P \multimap Q)(a) \triangleq \int_{a'\in\mathbb{C}} Pa' \Rightarrow Q(a \bullet a')$$

⌟

The above is only one of two possible Day convolution structures that $\widetilde{\mathbb{C}}$ can be equipped with, the other one being on the reverse monoidal category $(\mathbb{C}, \bullet^r, i)$:

$$(P \circledcirc Q)(a) \triangleq \int^{a_1,a_2 \in \mathbb{C}} \mathbb{C}(a_2 \bullet a_1, a) \times Pa_1 \times Qa_2 \qquad (P \multimap Q)(a) \triangleq \int_{a' \in \mathbb{C}} Pa' \Rightarrow Q(a' \bullet a)$$

Proposition 8.2.1 *The two monoidal structures are isomorphic in the sense that:*

$$P \circledcirc Q \cong Q \circledcirc P \qquad \widetilde{\mathbb{C}}(P, Q \multimap R) \cong \widetilde{\mathbb{C}}(Q, P \multimap R)$$

Proof The first follows easily from commutativity of $\times$ and the Fubini rule:

$$\begin{aligned}
(P \circledcirc Q)(a) &= \int^{a_1,a_2 \in \mathbb{C}} \mathbb{C}(a_1 \bullet a_2, a) \times Pa_1 \times Qa_2 \\
&\cong \int^{a_1,a_2 \in \mathbb{C}} \mathbb{C}(a_1 \bullet a_2, a) \times Qa_2 \times Pa_1 \\
&\cong \int^{a_2,a_1 \in \mathbb{C}} \mathbb{C}(a_1 \bullet a_2, a) \times Qa_2 \times Pa_1 && (\textstyle\int\int^{\bullet}) \\
&= (Q \circledcirc P)(a)
\end{aligned}$$

The second follows from the first using the tensor-hom adjunctions.

$$\widetilde{\mathbb{C}}(P, Q \multimap R) \cong \widetilde{\mathbb{C}}(P \circledcirc Q, R) \cong \widetilde{\mathbb{C}}(Q \circledcirc P, R) \cong \widetilde{\mathbb{C}}(Q, P \multimap R) \qquad \square$$

The following standard result can be established by another set of co/end calculations; since we have more than enough of those in this thesis, we refer the reader to e.g. Day (1970, Theorem 3.3), Im and Kelly (1986, Proposition 4.1) and Loregian (2021, Proposition 6.2.1).

Theorem 8.2.2

$(\widetilde{\mathbb{C}}, \text{ɏ}i, \circledcirc, \multimap)$ and $(\widetilde{\mathbb{C}}, \text{ɏ}i, \circledcirc, \multimap)$ are monoidal closed categories.

If the base category is symmetric monoidal, the two reverse structures are isomorphic; if $\mathbb{C}$ is furthermore cocartesian, both convolution structures collapse to the pointwise cartesian one.

Proposition 8.2.2 *The Day monoidal closed structure over a cocartesian category coincides with the cartesian closed structure of presheaves*

Proof Assume $(\mathbb{C}, +, \bot)$ is cocartesian, and consider the following calculations:

$$\text{ɏ}(\bot)(a) = \mathbb{C}(\bot, a) \cong \{*\} = \top(a)$$

$$\begin{aligned}
(P \circledcirc_{\mathbb{C}} Q)(a) &= \int^{a_1,a_2 \in \mathbb{C}} \mathbb{C}(a_1 + a_2, a) \times Pa_1 \times Qa_2 \\
&\cong \int^{a_1,a_2 \in \mathbb{C}} \mathbb{C}(a_1, a) \times \mathbb{C}(a_2, a) \times Pa_1 \times Qa_2 && \text{(UP of } +) \\
&\cong \int^{a_1 \in \mathbb{C}} \mathbb{C}(a_1, a) \times Pa_1 \times \int^{a_2 \in \mathbb{C}} \mathbb{C}(a_2, a) \times Qa_2 \\
&\cong Pb \times Qb = (P \times_{\mathbb{C}} Q)(a) && (\textstyle\int \text{ɏ} \times)
\end{aligned}$$

Thus, $\text{ɟ}\bot \cong \top$ and $P \circledcirc Q \cong P \times Q$, so the two structures are isomorphic. By uniqueness of adjoints, we also have that $P \multimap Q \cong P \supset Q$. Moreover, since cocartesian categories are symmetric, the reverse Day convolution structure is again isomorphic: $P \circledcirc Q \cong P \circledcirc Q \cong P \times Q$ and $P \multimap Q \cong P \circ\!\!- Q \cong P \supset Q$. □

This property shows that the Day convolution structure over rich enough base categories is uninteresting – indeed, in the presheaf model, we work with presheaves over the free cocartesian category $\mathbb{F}$, so the Day structure on $\widetilde{\mathbb{F}}$ is isomorphic to the cartesian closed one. This is not the case in $\widetilde{\mathbb{N}}$, and our familial model will make extensive use of the Day internal hom in favour of the family exponential.

The Day convolution satisfies its own universal property, enriching that of presheaves in the case that the base category is monoidal. We outline the argument below; for details, see the classic paper by Im and Kelly (1986).

Definition 8.2.4 A category $\mathcal{C}$ is *monoidally cocomplete* if it is monoidal, and the tensor product is cocontinuous in both operands. ⌟

Lemma 8.2.1 *For a category $\mathcal{C}$ co/powered over* **Set**, *with $X, Y \in$ **Set** and $A, B \in \mathcal{C}$, we have*

$$(X \times Y) \cdot A \cong X \cdot (Y \cdot A) \qquad (\times\cdot) \qquad\qquad (X \times Y) \pitchfork A \cong X \pitchfork (Y \pitchfork A) \qquad (\times\pitchfork)$$

Proof By Yoneda, we have the following calculations:

$$\begin{aligned}
&\mathcal{C}\big((X \times Y) \cdot A, B\big) && && \mathcal{C}\big(C, (X \times Y) \pitchfork A\big) && \\
&\cong \mathbf{Set}\big(X \times Y, \mathcal{C}(A, B)\big) && (\triangleq\cdot) && \cong \mathbf{Set}\big(X \times Y, \mathcal{C}(X, A)\big) && (\triangleq\pitchfork) \\
&\cong \mathbf{Set}\big(X, \mathbf{Set}(Y, \mathcal{C}(A, B))\big) && (\text{currying}) && \cong \mathbf{Set}\big(X, \mathbf{Set}(Y, \mathcal{C}(X, A))\big) && (\text{currying}) \\
&\cong \mathbf{Set}\big(X, \mathcal{C}(Y \cdot A, B)\big) && (\triangleq\cdot) && \cong \mathbf{Set}\big(X, \mathcal{C}(X, Y \pitchfork A)\big) && (\triangleq\pitchfork) \\
&\cong \mathcal{C}\big(X \cdot (Y \cdot A), B\big) && (\triangleq\cdot) && \cong \mathcal{C}\big(C, X \pitchfork (Y \pitchfork A)\big) && (\triangleq\cdot)
\end{aligned}$$

□

Lemma 8.2.2 *For $(\mathcal{C}, \otimes, I)$ monoidally cocomplete, $X, Y \in$ **Set** and $A, B \in \mathcal{C}$, we have the following interchange law:*

$$(X \times Y) \cdot (A \otimes B) \cong (X \cdot A) \otimes (Y \cdot B) \qquad (\cdot\times\otimes)$$

Proof We use the fact that copowering by **Set** is the colimit $X \cdot A \triangleq \coprod_{x \in X} A$ so it is preserved by the tensor $\otimes$. We calculate as follows:

$$\begin{aligned}
(X \times Y) \cdot (A \otimes B) &\cong X \cdot (Y \cdot (A \otimes B)) && (\times\cdot) \\
&\cong X \cdot (A \otimes (Y \cdot B)) && (A \otimes (-) \text{ cocontinuous}) \\
&\cong (X \cdot A) \otimes (Y \cdot B) && ((-) \otimes (Y \cdot B) \text{ cocontinuous})
\end{aligned}$$

□

Theorem 8.2.3 (Im and Kelly (1986, Theorem 5.1))

> *$\widehat{\mathbb{C}}$ is the free monoidal cocompletion of a monoidal $\mathbb{C}$: given any monoidally cocomplete category $\mathcal{D}$ and a monoidal functor $F\colon \mathbb{C} \to \mathcal{D}$, there is a unique cocontinuous monoidal extension $F^\sharp\colon \widehat{\mathbb{C}} \to \mathcal{D}$, exhibiting an equivalence of categories* $\mathbf{Mon}(\mathbb{C}, \mathcal{D})$ *and* $\mathbf{MonCocont}(\widehat{\mathbb{C}}, \mathcal{D})$.

Proof For a monoidal $\mathbb{C}$, $\widehat{\mathbb{C}}$ is equipped with a monoidal structure given by Day convolution. This is monoidally cocomplete, since the convolution expands into a coend of cartesian products, $\times$ is cocontinuous and colimits commute with coends. Given a monoidal $F\colon \mathbb{C} \to \mathcal{D}$ for $(\mathbb{C}, \bullet, i)$ and $(\mathcal{D}, \otimes, J)$, the obvious extension to $\widehat{\mathbb{C}}$ is the cocontinuous $F^\sharp\colon \widehat{\mathbb{C}} \to \mathcal{D}$ as given in Theorem 8.2.1; it remains to show that it is monoidal. The preservation of the unit $F^\sharp(\text{ɣ} i) \cong J$ follows from the ninja Yoneda lemma and unit-preservation of F itself:

$$F^\sharp(\text{ɣ} i) \;=\; \int^{a \in \mathbb{C}} \text{ɣ}(i)(a) \cdot Fa \cong Fi \cong J$$

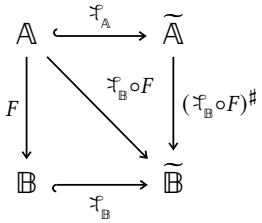
We now prove that $F^\sharp(X \circledcirc Y) \cong F^\sharp X \otimes F^\sharp Y$ as follows:

$$\begin{aligned}
F^\sharp(X \circledcirc Y) &\cong \int^{a \in \mathbb{C}} (X \circledcirc Y)(a) \cdot Fa \\
&\cong \int^{a \in \mathbb{C}} \left(\int^{a_1, a_2 \in \mathbb{C}} \mathbb{C}(a, a_1 \bullet a_2) \times Xa_1 \times Ya_2 \right) \cdot Fa && (\circledcirc^{\triangleq}) \\
&\cong \int^{a_1, a_2 \in \mathbb{C}} \int^{a \in \mathbb{C}} (\mathbb{C}(a, a_1 \bullet a_2) \times Xa_1 \times Ya_2) \cdot Fa && (\textstyle\int\int^\bullet, \int \cdot) \\
&\cong \int^{a_1, a_2 \in \mathbb{C}} \int^{a \in \mathbb{C}} \mathbb{C}(a, a_1 \bullet a_2) \cdot \big((Xa_1 \times Ya_2) \cdot Fa\big) && (\times \cdot) \\
&\cong \int^{a_1, a_2 \in \mathbb{C}} (Xa_1 \times Ya_2) \cdot F(a_1 \bullet a_2) && (\textstyle\int \text{ɣ} \cdot) \\
&\cong \int^{a_1, a_2 \in \mathbb{C}} (Xa_1 \times Ya_2) \cdot (Fa_1 \otimes Fa_2) && (F \text{ monoidal}) \\
&\cong \int^{a_1, a_2 \in \mathbb{C}} \big(Xa_1 \cdot Fa_1\big) \otimes \big(Ya_2 \cdot Fa_2\big) && (\cdot \times \otimes) \\
&\cong \left(\int^{a_1 \in \mathbb{C}} Xa_1 \cdot Fa_1 \right) \otimes \left(\int^{a_2 \in \mathbb{C}} Ya_2 \cdot Fa_2 \right) && (\textstyle\int \cdot, \otimes \text{ cocontinuous}) \\
&\cong F^\sharp X \otimes F^\sharp Y
\end{aligned}$$

If F is lax monoidal, so is $F^\sharp$ – the applications of monoidality above become one-way transformations. The property dualises in the obvious way. □

Corollary 8.2.1 *A monoidal $F\colon \mathbb{A} \to \mathbb{B}$, induces a cocontinuous monoidal $\mathrm{Lan}_F : \widetilde{\mathbb{A}} \to \widetilde{\mathbb{B}}$.*

Proof The universal property of convolution induces the functor $\mathrm{Lan}_{\text{ɣ}_\mathbb{A}} (\text{ɣ}_\mathbb{B} \circ F)\colon \widetilde{\mathbb{A}} \to \widetilde{\mathbb{B}}$:



However, this is just the left Kan extension along F:

$$(\text{ɣ}_\mathbb{B} \circ F)^\sharp(P)(b) = \int^{a \in \mathbb{C}} Pa \times \mathbb{B}(Fa, b) \cong \mathrm{Lan}_F P(b)$$ □

We next consider some presheaf-specific constructions derived from the left Kan extension.

8.3 Nerves and realisations

Combining Kan extensions with presheaves and the Yoneda embedding gives rise to functors that have particularly simple expressions thanks to the co/end calculus, and they will form the basis of the substitution tensor product and hom to be introduced in the next chapter.

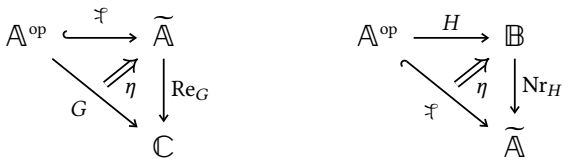
Definition 8.3.1 Given a functor $G\colon \mathbb{A}^{\mathrm{op}} \to \mathbb{C}$, its left Kan extension along the Yoneda embedding $\text{ℱ}\colon \mathbb{A}^{\mathrm{op}} \to \widetilde{\mathbb{A}}$ is called the *realisation* of G:

$$\mathrm{Re}_G \triangleq \mathrm{Lan}_{\text{ℱ}} G\colon \widetilde{\mathbb{A}} \to \mathbb{C}$$

Given a functor $H\colon \mathbb{A}^{\mathrm{op}} \to \mathbb{B}$, the left Kan extension of the Yoneda embedding $\text{ℱ}\colon \mathbb{A}^{\mathrm{op}} \to \widetilde{\mathbb{A}}$ along H will be called the *nerve* of H:

$$\mathrm{Nr}_H \triangleq \mathrm{Lan}_H \text{ℱ}\colon \mathbb{B} \to \widetilde{\mathbb{A}}$$

The two notions fit into a diagram as follows:



Both concepts can be reduced to simpler expressions using the unfolding of the left extension as a coend.

Proposition 8.3.1 *The realisation and nerve functors satisfy the following isomorphisms, for* $G\colon \mathbb{A}^{\mathrm{op}} \to \mathbb{C}$ *and* $H\colon \mathbb{A}^{\mathrm{op}} \to \mathbb{B}$*:*

$$\mathrm{Re}_G(P) \cong \int^{a \in \mathbb{A}} Pa \cdot Ga \qquad (\mathrm{Re}\cong) \qquad\qquad \mathrm{Nr}_H(b) \cong \mathbb{B}(H-, b) \qquad (\mathrm{Nr}\cong)$$

Proof Consider the following calculations, for $P \in \widetilde{\mathbb{A}}$, $b \in \mathbb{B}$ and $a \in \mathbb{A}$:

$$\begin{aligned} & \mathrm{Re}_G(P) \\ &= \mathrm{Lan}_{\text{ℱ}} G(P) \\ &\cong \int^{a \in \mathbb{A}} \widetilde{\mathbb{A}}(\text{ℱ}a, P) \cdot Ga && (\mathrm{Lan}\int) \\ &\cong \int^{a \in \mathbb{A}} Pa \cdot Ga && (\text{ℱ}\cong) \end{aligned}$$

$$\begin{aligned} & \mathrm{Nr}_H(b)(a) \\ &= \mathrm{Lan}_H \text{ℱ}(b)(a) \\ &\cong \int^{a' \in \mathbb{A}} \mathbb{B}(Ha', b) \times \mathbb{A}(a', a) && (\mathrm{Lan}\int) \\ &\cong \mathbb{B}(Ha, b) && (\int \text{ℱ} \times) \end{aligned}$$

□

The following lemmas set out some calculational properties of nerves and realisations, based on the expansions above.

Lemma 8.3.1 *The realisation and the nerve of the Yoneda embedding* $\text{ℶ}\colon \mathbb{A}^{\text{op}} \to \widetilde{\mathbb{A}}$ *is naturally isomorphic to the identity:*

$$\text{Re}_{\text{ℶ}} \;\cong\; \text{Id} \;\cong\; \text{Nr}_{\text{ℶ}} : \widetilde{\mathbb{A}} \to \widetilde{\mathbb{A}}$$

Proof Applying the isomorphisms above and the Yoneda lemmas, we have:

$$\text{Re}_{\text{ℶ}} P \;\cong\; \int^{a \in \mathbb{A}} Pa \times \text{ℶ} a \;\cong\; P \;\cong\; \widetilde{\mathbb{A}}(\text{ℶ} a, P) \;\cong\; \text{Nr}_{\text{ℶ}} P \qquad \square$$

Lemma 8.3.2 *The unit of the realisation of* $G\colon \mathbb{A}^{\text{op}} \to \mathbb{C}$ *is an isomorphism:*

$$\text{Re}_G \circ \text{ℶ} \cong G \tag{Re∘ℶ}$$

Proof A simple application of the Yoneda lemma of Eq. (∫ℶ·):

$$\text{Re}_G(\text{ℶ}(a)) \cong \int^{b \in \mathbb{A}} \text{ℶ}(a)(b) \cdot G \cong \int^{b \in \mathbb{A}} \text{ℸ}(b)(a) \cdot G \cong G(a) \qquad \square$$

Nerves and realisations are also adjoint functors, as established next.

Lemma 8.3.3 *For functors* $H\colon \mathbb{A}^{\text{op}} \to \mathbb{B}$ *and* $G\colon \mathbb{A}^{\text{op}} \to \mathbb{C}$*, morphisms* $\text{Lan}_H G(b) \to c \in \mathbb{C}$ *are in natural bijection with natural transformations* $\text{Nr}_H b \Longrightarrow \text{Nr}_G c$*:*

$$\mathbb{C}\big(\text{Lan}_H G(b), c\big) \cong \widetilde{\mathbb{A}}\big(\text{Nr}_H b, \text{Nr}_G c\big)$$

Proof Consider the following calculation:

$$\begin{aligned}
\mathbb{C}\big(\text{Lan}_H G(b), c\big) &\cong \mathbb{C}\Big(\int^{a \in \mathbb{A}} \mathbb{B}(Ha, b) \cdot Ga, c\Big) && \text{(Lan∫)} \\
&\cong \int_{a \in \mathbb{A}} \mathbb{C}\big(\mathbb{B}(Ha, b) \cdot Ga, c\big) && \text{(∫↕)} \\
&\cong \int_{a \in \mathbb{A}} \mathbf{Set}\big(\mathbb{B}(Ha, b), \mathbb{C}(Ga, c)\big) && \text{(·≅)} \\
&\cong \widetilde{\mathbb{A}}\big(\mathbb{B}(H-, b), \mathbb{C}(G-, c)\big) && \text{(∫NT)} \\
&= \widetilde{\mathbb{A}}\big(\text{Nr}_H b, \text{Nr}_G c\big) && \square
\end{aligned}$$

Corollary 8.3.1 *The realisation and the nerve of a functor* $G\colon \mathbb{A}^{\text{op}} \to \mathbb{C}$ *are adjoint:*

$$\text{Re}_G \dashv \text{Nr}_G : \widetilde{\mathbb{A}} \to \mathbb{C} \tag{Re⊣Nr}$$

Proof The adjunction follows from the hom-set isomorphism constructed from the previous two propositions:

$$\mathbb{C}(\text{Re}_G(P), c) = \mathbb{C}(\text{Lan}_{\text{ℶ}} G(P), c) \cong \widetilde{\mathbb{A}}(\text{Nr}_{\text{ℶ}} P, \text{Nr}_G c) \cong \widetilde{\mathbb{A}}(P, \text{Nr}_G c) \qquad \square$$

Using the universal properties of presheaves, Day convolution, and the definition of realisations, we obtain the following result.

Proposition 8.3.2 *For cartesian categories* $\mathbb{C}$ *and* $\mathcal{D}$ *and cartesian functor* $G\colon \mathbb{C}^{\text{op}} \to \mathcal{D}$*, the realisation functor* $\text{Re}_G : \widetilde{\mathbb{C}} \to \mathcal{D}$ *is cartesian.*

Proof $\mathbb{C}$ cartesian implies $\mathbb{C}^{\mathrm{op}}$ is cocartesian monoidal; the Day convolution based on it reduces to the cartesian structure on $\widetilde{\mathbb{C}}$ (Proposition 8.2.2). Since $\mathrm{Re}_G = \mathrm{Lan}_{\yoneda} G \colon \widetilde{\mathbb{C}} \to \mathcal{D}$ is the free monoidally cocontinuous extension of G, by the universal property of the Day convolution (Theorem 8.2.3) it maps the Day monoidal (i.e. cartesian) structure to the cartesian structure of $\mathcal{D}$, preserved by the cartesian G. □

Further properties of nerves and realisations are the topic of the next section, where we use the universal properties to extract the substitution structure of presheaves through abstract and indirect means, then decompose the proof into lower-level operations involving Nr and Re to reconstruct it in the weaker setting of indexed families.

CHAPTER 9

Substitution

Substitution appears in many guises throughout mathematics: composition and grafting of multicategories, species, opetopes and operads (Bergeron et al., 1997; Leinster, 2004; Fiore, 2005; Fiore et al., 2008; J. Kock et al., 2010), cut elimination in logic (Dyckhoff, 2015), clones in universal algebra (Kerkhoff et al., 2014), and of course, substitution of presheaves in abstract syntax (Fiore et al., 1999; Fiore, 2007; Power, 2007). For the purposes of formalisation, a core contribution of the presheaf model is an algebraic characterisation of simultaneous substitution: rather than a singular operation, it is decomposed into the substitution monoidal structure on presheaves, and the multiplication operation of a monoid therewithin. Objects with unevaluated substitutions can be manipulated directly and heterogeneously, enabling the definitions of algebraic models, parametrised initiality, etc. The practical strength of this perspective is perhaps best illustrated by how non-categorical implementations of intrinsically typed syntax have, over time, independently rediscovered many of its core ideas – such as renaming, substitution, parametrised traversal, and semantic models – as seen in the implementation-driven work of McBride (2005), Benton et al. (2012), and Allais et al. (2021).

In this chapter we give three related accounts of simultaneous substitution for simply-typed syntax. In Section 9.1, universal properties are used to equip the category of presheaves over the free cocartesian category with a monoidal substitution structure. Section 9.2 rederives this structure from weaker assumptions, using nerves, realisations, and the co/end calculus to obtain a more general result over arbitrary small base categories, showing how the results strengthen in the presence of (free) cocartesian structure. Section 9.3 shows how to build a richer substitution structure even in the absence of cocartesian structure on the base category, using an adjoint warping. Throughout, our goal is to identify the minimal assumptions on a base category $\mathbb{C}$ that allow its presheaf category to support a sensible notion of substitution.

Examples in the previous sections used unsorted syntax with the categories $\widetilde{\mathbb{N}}$ and $\widetilde{\mathbb{F}}$. To encompass simply-typed syntax, we will have to consider objects indexed by a type. We fix a set S of *sorts* and consider indexed categories $\mathcal{C}^S$ as *sorted*. We also write objects of the slice category $S/\mathbf{Cat}$ as sorted categories $J_{\mathbb{A}} \colon S \to \mathbb{A}$, $J_{\mathbb{B}} \colon S \to \mathbb{B}$, etc., with functors $F \colon \mathbb{A} \to \mathbb{B}$ satisfying $F \circ J_{\mathbb{A}} \cong J_{\mathbb{B}}$.

9.1 Substitution through universality

First presented is a high-level argument adapted from Trimble (2013), relying on universal properties of cartesian extensions and monoidal cocompletions: if $\mathbb{C}$ is the free cocartesian category on S, $\widetilde{\mathbb{C}}^S$ is strong monoidal closed. While an elegant line of "abstract nonsense" reasoning, it rests on assumptions we aim to relax: namely, that $\mathbb{C}$ must be the free cocartesian category on a set of sorts. After outlining the abstract argument, we dissect elements of the proof and pinpoint where the properties of $\mathbb{C}$ and J are required, and explore how much of the structure can be recovered without these assumptions.

Theorem 9.1.1

Given a set S and its generated free cocartesian category $\mathbb{C}$, the category $\widetilde{\mathbb{C}}^S$ is monoidal.

Proof Let S be a set of sorts and $\mathbb{C}$ the free cocartesian category generated by S, and $\mathcal{D}$ a cocomplete category. $\mathbb{C}^{\mathrm{op}}$ is then the free cartesian category, satisfying the equivalence of categories ① $\mathcal{D}^S \simeq \mathbf{Cart}(\mathbb{C}^{\mathrm{op}}, \mathcal{D})$. The UPs of the presheaf construction (Theorem 8.2.1) and Day convolution (Theorem 8.2.3) in turn say that $\widetilde{\mathbb{C}}$ is the free monoidal cocompletion of $\mathbb{C}^{\mathrm{op}}$, exhibiting the equivalence $\mathbf{Mon}(\mathbb{C}^{\mathrm{op}}, \mathcal{D}) \simeq \mathbf{MonCocont}(\widetilde{\mathbb{C}}, \mathcal{D})$; however, if the cocontinuous monoidal structure on $\mathbb{C}^{\mathrm{op}}$ is selected to be cartesian (so the covariant Day convolution structure on $\widetilde{\mathbb{C}}$ also coincides with the cartesian one, see Proposition 8.2.2), the equivalence reduces to ② $\mathbf{Cart}(\mathbb{C}^{\mathrm{op}}, \mathcal{D}) \simeq \mathbf{CartCocont}(\widetilde{\mathbb{C}}, \mathcal{D})$.

Composing ① and ② gives an equivalence between $\mathcal{D}^S$ and $\mathbf{CartCocont}(\widetilde{\mathbb{C}}, \mathcal{D})$. Explicitly, a sorted $\mathcal{Q} \colon \mathcal{D}^S$ is mapped first to the cartesian functor $\mathcal{Q}^\times \colon \mathbb{C}^{\mathrm{op}} \to \mathcal{D}$ defined as

$$\mathcal{Q}^\times(a) \triangleq \mathrm{Ran}_{J^{\mathrm{op}}} \mathcal{Q}(a) \cong \int_{\tau \in S} \mathbb{C}(J\tau, a) \pitchfork \mathcal{Q}_\tau$$

which, in turn, maps to the cartesian cocontinuous functor $(-) \odot \mathcal{Q} \colon \widetilde{\mathbb{C}} \to \mathcal{D}$, defined as

$$X \odot \mathcal{Q} \triangleq \mathrm{Re}_{\mathcal{Q}^\times} \cong \int^{a \in \mathbb{C}} Xa \cdot \mathcal{Q}^\times(a)$$

where the realisation of a cartesian functor is cartesian by Proposition 8.3.2. In the opposite direction, the transformations are simply pre-compositions with $\yoneda$ and J: a cartesian cocontinuous functor $G \colon \widetilde{\mathbb{C}} \to \mathcal{D}$ becomes a cartesian functor $G \circ \yoneda \colon \mathbb{C}^{\mathrm{op}} \to \mathcal{D}$ and then a functor (or sorted $\mathcal{D}$-object) $G \circ \yoneda \circ J^{\mathrm{op}} \colon S \to \mathcal{D}$.

Choosing $\mathcal{D}$ to be $\widetilde{\mathbb{C}}$ itself (which is a cocomplete presheaf category), the chain of equivalences $\mathcal{D}^S \simeq \mathbf{Cart}(\mathbb{C}^{\mathrm{op}}, \mathcal{D}) \simeq \mathbf{CartCocont}(\widetilde{\mathbb{C}}, \mathcal{D})$ reduces to:

$$\widetilde{\mathbb{C}}^S \simeq \mathbf{CartCocont}(\widetilde{\mathbb{C}}, \widetilde{\mathbb{C}})$$

The cartesian cocontinuous endofunctor category is strictly monoidal, given by the identity functor and functor composition. This monoidal structure transfers across the equivalence to $\widetilde{\mathbb{C}}^S$, exhibiting it as a monoidal category too. Explicitly, the unit is $\mathcal{J} \triangleq \mathrm{Id} \circ \yoneda \circ J^{\mathrm{op}} = \yoneda \circ J^{\mathrm{op}}$, and the multiplication is the functor $(-) \otimes (=) \colon \widetilde{\mathbb{C}}^S \times \widetilde{\mathbb{C}}^S \to \widetilde{\mathbb{C}}^S$ calculated as:

$$\mathcal{P} \otimes \mathcal{Q} \cong \mathrm{Re}_{\mathcal{Q}^\times} \circ \mathrm{Re}_{\mathcal{P}^\times} \circ \yoneda \circ J^{\mathrm{op}}$$

$$\begin{aligned}
&\cong \mathrm{Re}_{\mathcal{Q}^\times} \circ \mathcal{P}^\times \circ J^{\mathrm{op}} \\
&\cong \mathrm{Re}_{\mathcal{Q}^\times} \circ \mathcal{P} \\
&\cong \tau \mapsto \int^{a \in \mathbb{C}} \mathcal{P}_\tau(a) \cdot \mathcal{Q}^\times a
\end{aligned}$$

where the middle isomorphisms follow from the invertibility of Re and $(-)^\times$. □

The proof above extracts a unital and associative tensor product from an equivalence between the sorted presheaf category $\widetilde{\mathbb{C}}^S$ and the cartesian cocontinuous endofunctor category $\mathbf{CartCocont}(\widetilde{\mathbb{C}}, \widetilde{\mathbb{C}})$. Under the assumption that $\mathbb{C}$ is the free cocartesian category on S, the theorem sets out the core categorical infrastructure of the presheaf model of abstract syntax: $\mathcal{J}$ acts as the presheaf of variables, and $\otimes$ as the substitution tensor. However, if $\mathbb{C}$ is *not* freely cocartesian, or lacks cocartesian structure altogether, crucial elements of the proof collapse: we lose access to universality conditions needed to establish equivalences and transport the composition structure into a monoidal tensor product. This raises the central question: how much can we weaken the assumptions on $\mathbb{C}$ while still recovering a workable notion of substitution, and what categorical structure on $\widetilde{\mathbb{C}}^S$ remains in such cases?

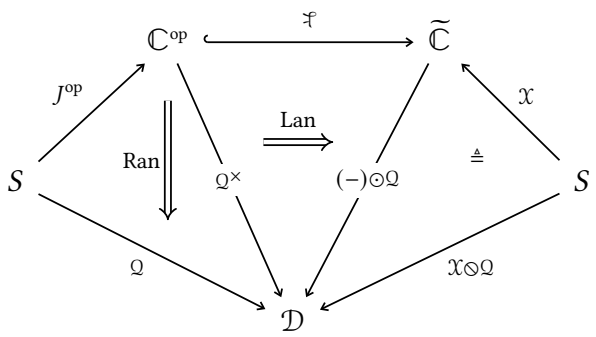
Somewhat surprisingly, a *skew-monoidal* structure persists even if $\mathbb{C}$ is merely an arbitrary small category with sorting $S \to \mathbb{C}$. We now redevelop the proof of Theorem 9.1.1 from the ground up, starting from minimal assumptions and building up structure until the proof of the full strong monoidal structure is possible.

9.2 Substitution from first principles

The abstract proof above induces several important constructions by universality, and the construction of the tensor product – transported from the strict monoidal category of endofunctors – guarantees the monoidal structure to be strong. However, the weaker, skew-monoidal structure arises quite naturally when defining the operations from scratch.

9.2.1 Skew-monoidal structure

Suppose $\mathbb{C}$ is a small category and $J\colon S \to \mathbb{C}$ is a sorting. The universal properties of the abstract proof may be weakened to extensions without the invertibility condition, resulting in the following situation:



Definition 9.2.1 Given an object $\mathcal{Q}$ of the sorted category $\mathcal{D}^S$, the *$\mathbb{C}$-extension* $\mathcal{Q}^{\times_{\mathbb{C}}} : \mathbb{C}^{\mathrm{op}} \to \mathcal{D}$ is the right Kan extension $\mathcal{Q}^{\times_{\mathbb{C}}} \triangleq \mathrm{Ran}_{J_{\mathbb{C}}^{\mathrm{op}}} \mathcal{Q}$ with counit $\mathcal{Q}^{\times_{\mathbb{C}}} \circ J^{\mathrm{op}} \Longrightarrow \mathcal{Q}$:

$$\mathcal{Q}^{\times_{\mathbb{C}}}(a) \cong \int_{\tau \in S} \mathbb{C}^{\mathrm{op}}(a, J\tau) \pitchfork \mathcal{Q}_\tau \cong \int_{\tau \in S} \mathbb{C}(J\tau, a) \pitchfork \mathcal{Q}_\tau$$ ⌟

Definition 9.2.2 The *substitution operator* $(-) \odot^{\mathbb{C}} (=) : \widetilde{\mathbb{C}} \times \mathcal{D}^S \to \mathcal{D}$ is defined as:

$$(-) \odot^{\mathbb{C}} \mathcal{Q} \triangleq \mathrm{Re}_{\mathcal{Q}^{\times_{\mathbb{C}}}} \qquad X \odot^{\mathbb{C}} \mathcal{Q} \triangleq \int^{a \in \mathbb{C}} Xa \cdot \mathcal{Q}^{\times_{\mathbb{C}}} a$$

with unit $\mathcal{Y}(-) \odot \mathcal{Q} \cong \mathcal{Q}^{\times_{\mathbb{C}}}$. The *substitution action* $(-) \circledcirc^{\mathbb{C}} (=) : \widetilde{\mathbb{C}}^S \times \mathcal{D}^S \to \mathcal{D}^S$ is

$$(\mathcal{X} \circledcirc^{\mathbb{C}} \mathcal{Q})_\tau \triangleq \mathcal{X}_\tau \odot^{\mathbb{C}} \mathcal{Q}$$ ⌟

Definition 9.2.3 The *substitution bracket* $\lfloor -, = \rceil^{\mathbb{C}} : (\mathcal{D}^S)^{\mathrm{op}} \times \mathcal{D} \to \widetilde{\mathbb{C}}$ is defined as:

$$\lfloor \mathcal{P}, - \rceil^{\mathbb{C}} \triangleq \mathrm{Nr}_{\mathcal{P}^{\times_{\mathbb{C}}}} \qquad \lfloor \mathcal{P}, \mathcal{Q} \rceil^{\mathbb{C}} \triangleq \mathcal{D}(\mathcal{P}^{\times_{\mathbb{C}}} -, \mathcal{Q})$$

with unit $\mathcal{Y} \to \lfloor \mathcal{P}, \mathcal{Q}^{\times}(-) \rceil$. The *substitution enrichment* $\langle -, = \rangle^{\mathbb{C}} : (\mathcal{D}^S)^{\mathrm{op}} \times \mathcal{D}^S \to \widetilde{\mathbb{C}}^S$ is then

$$\langle \mathcal{P}, \mathcal{Q} \rangle^{\mathbb{C}}_\tau \triangleq \lfloor \mathcal{P}, \mathcal{Q}_\tau \rceil^{\mathbb{C}}$$ ⌟

From Eq. (Re ⊣ Nr), the two operators and brackets form the adjunction for all $\mathcal{Q} \in \mathcal{D}^S$:

$$(-) \odot^{\mathbb{C}} \mathcal{Q} \dashv \lfloor \mathcal{Q}, = \rceil^{\mathbb{C}} : \widetilde{\mathbb{C}} \to \mathcal{D} \qquad (-) \circledcirc^{\mathbb{C}} \mathcal{Q} \dashv \langle \mathcal{Q}, = \rangle^{\mathbb{C}} : \widetilde{\mathbb{C}}^S \to \mathcal{D}^S$$

The category superscripts will be omitted when irrelevant.

The operations above mirror some preservation properties of the underlying extensions, as first suggested by Fiore and Turi (2001).

Proposition 9.2.1 *For all $K : \mathcal{D} \to \mathcal{E}$, there is a canonical transformation $K \circ \mathcal{Q}^{\times} \to (K \circ \mathcal{Q})^{\times}$.*

Proof The transformation $K \circ \mathcal{Q}^{\times} \to (K \circ \mathcal{Q})^{\times}$ is the transpose derived from the counit of the right Kan extension:

$$\frac{K \circ \mathcal{Q}^{\times} \to \mathrm{Ran}_{J^{\mathrm{op}}} (K \circ \mathcal{Q})}{K \circ \mathcal{Q}^{\times} \circ J^{\mathrm{op}} \xrightarrow{K\varepsilon_{\mathcal{Q}}} K \circ \mathcal{Q}}$$ □

Corollary 9.2.1 *If $L : \mathcal{D} \to \mathcal{E}$ is a left adjoint, we have the natural transformation*

$$L(X \odot \mathcal{Q}) \to X \odot (L \circ \mathcal{Q}) : \widetilde{\mathbb{C}} \times \mathcal{D}^S \to \mathcal{E}$$

Proof Left adjoints preserve left Kan extensions (Proposition 8.1.4), so

$$L(- \odot \mathcal{Q}) = L \circ \mathrm{Re}_{\mathcal{Q}^{\times}} \cong \mathrm{Re}_{L \circ \mathcal{Q}^{\times}} \xrightarrow{\text{Prop. 9.2.1}} \mathrm{Re}_{(L \circ \mathcal{Q})^{\times}} = - \odot (L \circ \mathcal{Q})$$ □

Corollary 9.2.2 *If $R : \mathcal{E} \to \mathcal{D}$ is a right adjoint, there is a transformation $\lfloor \mathcal{P}, \mathcal{Q} \rceil \to \lfloor R \circ \mathcal{P}, R\mathcal{Q} \rceil$.*

Proof Assuming $L \dashv R \colon \mathcal{D} \to \mathcal{E}$, the Yoneda embedding of $L(X \odot \mathcal{P}) \to X \odot (L \circ \mathcal{P})$ from Corollary 9.2.1 induces $\lfloor L \circ \mathcal{P}, Q \rceil \to \lfloor \mathcal{P}, RQ \rceil$ as

$$\frac{\mathcal{E}(X \odot (L \circ \mathcal{P}), Q) \to \mathcal{E}(L(X \odot \mathcal{P}), Q)}{\widetilde{\mathbb{C}}(X, \lfloor L \circ \mathcal{P}, Q \rceil) \to \widetilde{\mathbb{C}}(X, \lfloor \mathcal{P}, RQ \rceil)}$$

from which we extract the required transformation as

$$\lfloor \mathcal{P}, Q \rceil \xrightarrow{\lfloor \varepsilon_{\mathcal{P}}, Q \rceil} \lfloor L \circ R \circ \mathcal{P}, Q \rceil \to \lfloor R \circ \mathcal{P}, GQ \rceil$$ □

In the case that $\mathcal{D}$ is itself a presheaf category, we can turn the Yoneda embedding into a sorted presheaf to stand for the unit.

Definition 9.2.4 The sorted presheaf of *$\mathbb{C}$-embeddings* $\mathcal{J}^{\mathbb{C}} \in \widetilde{\mathbb{C}}^{S}$ is defined as

$$\mathcal{J}^{\mathbb{C}} \triangleq \yo_{\mathbb{C}} \circ J_{\mathbb{C}}^{\mathrm{op}} \qquad \mathcal{J}^{\mathbb{C}}_{\tau}(a) = \mathbb{C}(J_{\mathbb{C}}\tau, a)$$ ⌟

Note that the presheaf of embeddings is the nerve $\mathrm{Nr}_{J^{\mathrm{op}}}(a)(\tau) \cong \mathbb{C}(J\tau, a)$ with swapped arguments – this is also referred to as the restricted Yoneda embedding. The Yoneda lemma for embeddings simplifies to

$$\widetilde{\mathbb{C}}(\mathcal{J}_{\tau}, Y) = \int_{a \in \mathbb{C}} \mathbb{C}(J\tau, a) \Rightarrow Ya \cong Y(J\tau)$$

and the extension $\mathcal{Q}^{\times} \colon \mathbb{C}^{\mathrm{op}} \to \mathcal{D}$ is equivalently $\int_{\tau \in S} \mathcal{J}_{\tau}(-) \pitchfork \mathcal{Q}_{\tau}$.

Notation. The extension into a presheaf $\mathcal{Q}^{\times} \colon \mathbb{C}^{\mathrm{op}} \to \widetilde{\mathbb{D}}$ becomes a profunctor whose elements $\mathcal{Q}^{\times}(a)(b) \in \mathbf{Set}$ will be denoted ${}^{a}\mathcal{Q}_{b}$. ⌟

Lemma 9.2.1 *We have a natural transformation $\yo \Longrightarrow \mathcal{J}^{\times}$.*

Proof The reasoning is as follows, using Proposition 9.2.1 and the unit $\mathrm{Id} \to \mathrm{Ran}_{J^{\mathrm{op}}} J^{\mathrm{op}}$:

$$\yo \to \yo \circ (J^{\mathrm{op}})^{\times} \to (\yo \circ J^{\mathrm{op}})^{\times} = \mathcal{J}^{\times}$$ □

We are ready to analyse the categorical properties of the substitution operator and bracket.

Lemma 9.2.2 *We have the following natural transformations:*

$$\lambda_{\mathcal{Q}} \colon \mathcal{J} \circledcirc^{\mathbb{C}} \mathcal{Q} \to \mathcal{Q} \colon \mathcal{D}^{S} \to \mathcal{D}^{S} \qquad \rho_{X} \colon X \to (X \odot^{\mathbb{C}} \mathcal{J}) \colon \widetilde{\mathbb{C}} \to \widetilde{\mathbb{C}}$$

$$\alpha_{X,\mathcal{Q},\mathcal{U}} \colon (X \odot^{\mathbb{C}} \mathcal{Q}) \odot^{\mathbb{D}} \mathcal{U} \to X \odot^{\mathbb{C}} (\mathcal{Q} \circledcirc^{\mathbb{D}} \mathcal{U}) \colon \widetilde{\mathbb{C}} \times \widetilde{\mathbb{D}}^{S} \times \mathcal{E}^{S} \to \mathcal{E}$$

Proof For the first, we use the isomorphic unit of $\mathrm{Re}_{\mathcal{Q}^{\times}}$ (Lemma 8.3.2) and the counit of the $\mathbb{C}$-extension:

$$\mathcal{J} \circledcirc \mathcal{Q} \triangleq \mathrm{Re}_{\mathcal{Q}^{\times}} \circ \yo \circ J^{\mathrm{op}} \cong \mathcal{Q}^{\times} \circ J^{\mathrm{op}} \to \mathcal{Q}$$

For the second, we have, by Lemma 8.3.1 and Lemma 9.2.1:

$$\mathrm{Id} \cong \mathrm{Re}_{\yo} \to \mathrm{Re}_{\mathcal{J}^{\times}} = (-) \odot \mathcal{J}$$

The third is an instance of Corollary 9.2.1, with $L = (-) \odot^{\mathbb{D}} \mathcal{U} = \mathrm{Re}_{\mathcal{U}^\times}$ which is left adjoint to $\mathrm{Nr}_{\mathcal{U}^\times}$ and therefore preserves the operator $\odot$:

$$\left(X \odot^{\mathbb{C}} \mathcal{Q}\right) \odot^{\mathbb{D}} \mathcal{U} = \mathrm{Re}_{\mathcal{U}^\times}\left(X \odot^{\mathbb{C}} \mathcal{Q}\right) \longrightarrow X \odot^{\mathbb{C}} \left(\mathrm{Re}_{\mathcal{U}^\times} \circ \mathcal{Q}\right) = X \odot^{\mathbb{C}} \left(\mathcal{Q} \oslash^{\mathbb{D}} \mathcal{U}\right) \qquad \square$$

Lemma 9.2.3 *We have the following natural transformations:*

$$j_{\mathcal{Q}} \colon \mathcal{J} \longrightarrow \langle \mathcal{Q}, \mathcal{Q} \rangle^{\mathbb{C}} \colon \mathcal{D}^S \longrightarrow \mathcal{D}^S \qquad i_Y \colon \lfloor \mathcal{J}, Y \rceil^{\mathbb{C}} \longrightarrow Y \colon \widetilde{\mathbb{C}} \longrightarrow \widetilde{\mathbb{C}}$$

$$L^{\mathcal{P}}_{\mathcal{Q},R} \colon \lfloor \mathcal{P}, R \rceil^{\mathbb{E}} \longrightarrow \lfloor \langle \mathcal{P}, \mathcal{Q} \rangle^{\mathbb{D}}, \lfloor \mathcal{P}, R \rceil^{\mathbb{D}} \rceil^{\mathbb{E}} \colon \mathcal{C}^S \times (\mathcal{C}^S)^{\mathrm{op}} \times \mathcal{C} \longrightarrow \widetilde{\mathbb{E}}$$

Proof The first is the adjoint transpose of $\lambda_{\mathcal{Q}} \colon \mathcal{J} \oslash \mathcal{Q} \longrightarrow \mathcal{Q}$ via $(-) \oslash \mathcal{Q} \dashv \langle \mathcal{Q}, = \rangle$. The second is computed using the Yoneda embedding of ρ, and adjunction $(-) \odot \mathcal{Q} \dashv \lfloor \mathcal{Q}, = \rceil$:

$$\frac{\widetilde{\mathbb{C}}(X \odot \mathcal{J}, Y) \longrightarrow \widetilde{\mathbb{C}}(X, Y)}{\widetilde{\mathbb{C}}(X, \lfloor \mathcal{J}, Y \rceil) \longrightarrow \widetilde{\mathbb{C}}(X, Y)}$$

The third is an instance of Corollary 9.2.2, with $R \triangleq \lfloor \mathcal{P}, - \rceil$ right adjoint to $(-) \odot \mathcal{P}$:

$$\lfloor \mathcal{Q}, R \rceil^{\mathbb{E}} \longrightarrow \lfloor \mathrm{Nr}_{\mathcal{P}^\times} \circ \mathcal{Q}, \mathrm{Nr}_{\mathcal{P}^\times} R \rceil^{\mathbb{E}} = \lfloor \langle \mathcal{P}, \mathcal{Q} \rangle^{\mathbb{D}}, \lfloor \mathcal{P}, R \rceil^{\mathbb{D}} \rceil^{\mathbb{E}} \qquad \square$$

The transformations above are highly heterogeneous, but by instantiating $\mathcal{D}$ with $\widetilde{\mathbb{C}}$, the operators and the transformations obtain the shape of tensors, homs, and actions. We will make the following notational distictions:

$$\begin{array}{lll} \oslash \triangleq \odot^{\mathbb{C}} \colon \widetilde{\mathbb{C}} \times \widetilde{\mathbb{C}}^S \longrightarrow \widetilde{\mathbb{C}} & & [-,=\rangle \triangleq \lfloor -,= \rceil^{\mathbb{C}} \colon (\widetilde{\mathbb{C}}^S)^{\mathrm{op}} \times \widetilde{\mathbb{C}} \longrightarrow \widetilde{\mathbb{C}} \\ \otimes \triangleq \oslash^{\mathbb{C}} \colon \widetilde{\mathbb{C}}^S \times \widetilde{\mathbb{C}}^S \longrightarrow \widetilde{\mathbb{C}}^S & & [-,=] \triangleq \langle -,= \rangle^{\mathbb{C}} \colon (\widetilde{\mathbb{C}}^S)^{\mathrm{op}} \times \widetilde{\mathbb{C}}^S \longrightarrow \widetilde{\mathbb{C}}^S \end{array}$$

Despite minimal assumptions on $\mathbb{C}$, the properties of the Kan extensions imply the following (cf. Altenkirch et al. (2010, Theorem 4)):

Theorem 9.2.1

> *The category $\widetilde{\mathbb{C}}^S$ is skew-monoidal closed, with unit $\mathcal{J} \in \widetilde{\mathbb{C}}^S$, tensor $\otimes \colon \widetilde{\mathbb{C}}^S \times \widetilde{\mathbb{C}}^S \longrightarrow \widetilde{\mathbb{C}}^S$ and hom $[-,=] \colon (\widetilde{\mathbb{C}}^S)^{\mathrm{op}} \times \widetilde{\mathbb{C}}^S \longrightarrow \widetilde{\mathbb{C}}^S$.*

Corollary 9.2.3 *The substitution operator $\oslash \colon \widetilde{\mathbb{C}} \times \widetilde{\mathbb{C}}^S \longrightarrow \widetilde{\mathbb{C}}$ and bracket $[-,=\rangle \colon \widetilde{\mathbb{C}}^S \times \widetilde{\mathbb{C}} \longrightarrow \widetilde{\mathbb{C}}$ make $\widetilde{\mathbb{C}}$ a right monoidal closed $\widetilde{\mathbb{C}}^S$-modular category.*

Corollary 9.2.4 *For all $\mathbb{D}$, the action $\oslash^{\mathbb{C}} \colon \widetilde{\mathbb{C}}^S \times \widetilde{\mathbb{D}}^S \longrightarrow \widetilde{\mathbb{D}}^S$ and enrichment $\langle -,= \rangle^{\mathbb{D}} \colon (\widetilde{\mathbb{D}}^S)^{\mathrm{op}} \times \widetilde{\mathbb{D}}^S \longrightarrow \widetilde{\mathbb{C}}^S$ make $\widetilde{\mathbb{D}}^S$ a left monoidal closed $\widetilde{\mathbb{C}}^S$-modular category.*

Although we are ultimately looking for a skew-monoidal structure, the one derived purely from Kan extensions will not be rich enough for our purposes. In the standard presheaf model, $\mathbb{C}$ is instantiated with the free cocartesian category $\mathbb{F}[S]$ on the set of sorts S, so the presheaf of variables $\mathcal{V}_\tau \Gamma = \mathbb{F}[S]([\tau], \Gamma)$ represents proof that τ is in the context Γ. In the familial model, $\mathbb{C}$ is the discrete category S^* of lists over S, and the embedding reduces to the family $\mathcal{N}_\tau \Gamma = S^*([\tau], \Gamma)$, which is inhabited only when $\Gamma = [\tau]$. The extension, substitution tensor and hom also behave in counterintuitive ways (see Section 9.3.2). Thus, some additional structure on the homsets of $\mathbb{C}$ is desirable to model the membership relation appropriately.

9.2.2 Strong monoidal structure

We obtain the classic presheaf model from the above theory by making $\mathbb{C}$ the free cocartesian category over S. Theorem 9.1.1 proves that this directly leads to the monoidal structure on the presheaf category $\widetilde{\mathbb{C}^S}$ by the universal properties of extensions and Day convolution. Here we spell out the details, showing how the constructions in the previous section strengthen with the assumption that $\mathbb{C}$ is the free cocartesian category $\mathbb{F}[S]$ over the set of sorts S. The freeness has several consequences:

- The embedding $[-] : S \to \mathbb{F}[S]$ is fully faithful: for all $\alpha \in S$, there is exactly one renaming rule $\mathbb{F}[S]([\alpha],[\alpha])$.
- The embedding $[-] : S \to \mathbb{F}[S]$ is dense: the nerve $\mathrm{Nr}_{[-]} : \mathbb{F}[S] \to \mathbf{Set}^S$ satisfies the full faithfulness isomorphism $\mathbf{Set}^S(\mathbb{F}[S]([-],\Gamma), \mathbb{F}[S]([-],\Delta)) \cong \mathbb{F}[S](\Gamma,\Delta)$, or equivalently, $\mathrm{Lan}_{[-]}\,[-] : \mathbb{F}[S]$ is naturally isomorphic to $\mathrm{Id}_{\mathbb{F}[S]}$.
- For a cartesian category $\mathcal{D}$, every $\mathcal{Q} \in \mathcal{D}^S$ has the free cartesian extension $\mathcal{Q}^\times : \mathbb{F}[S]^{\mathrm{op}} \to \mathcal{D}$ satisfying $\mathcal{Q}^{\times(\Gamma+\Delta)} \cong \mathcal{Q}^{\times\Gamma} \times \mathcal{Q}^{\times\Delta}$ and $\mathcal{Q}^{\times[\alpha]} \cong \mathcal{Q}_\alpha$.

Proposition 9.2.2 *For all $\mathcal{Q} \in \mathcal{D}^S$, the substitution action $(-) \odot \mathcal{Q} : \widetilde{\mathbb{F}[S]} \to \mathcal{D}$ is cartesian.*

Proof $\mathcal{Q}^\times : \mathbb{F}[S]^{\mathrm{op}} \to \mathcal{D}$ is a cartesian extension, and $(-) \odot \mathcal{Q} = \mathrm{Re}_{\mathcal{Q}^\times}$ is cartesian by Proposition 8.3.2.

□

Corollary 9.2.5 *The transformations of Proposition 9.2.1 and Corollary 9.2.1 strengthen to the following isomorphisms, for $K, L : \mathcal{D} \to \mathcal{E}$ cartesian, and L a left adjoint.*

$$K \circ \mathcal{Q}^\times \cong (K \circ \mathcal{Q})^\times \qquad L(- \odot \mathcal{Q}) \cong (-) \odot (L \circ \mathcal{Q})$$

Proof Since $(K \circ \mathcal{Q})^\times : \mathbb{F}[S]^{\mathrm{op}} \to \mathcal{E}$ is the cartesian extension of $K \circ \mathcal{Q} : S \to \mathcal{E}$, it's sufficient to show that $K \circ \mathcal{Q}^\times$ is cartesian and factorises $K \circ \mathcal{Q}$. The first is immediate from composition of cartesian functors $\mathcal{Q}^\times$ and K; the latter holds since $K \circ \mathcal{Q}^{\times[-]} \cong K \circ \mathcal{Q}$ by above. The maps in Corollary 9.2.1 become isomorphisms, establishing $L(- \odot \mathcal{Q}) \cong (-) \odot (L \circ \mathcal{Q})$. □

Remark. The transformation $\lfloor \mathcal{P}, \mathcal{Q} \rceil \to \lfloor R \circ \mathcal{P}, R\mathcal{Q} \rceil$ is not invertible due to its derivation using the counit $L \circ R \Longrightarrow \mathrm{Id}$. ⌟

Lemma 9.2.4 *We have $\mathcal{J}^\times \cong \mathcal{H}$.*

Proof As $[-] : S \to \mathbb{F}[S]$ is dense, the opposite $[-]^{\mathrm{op}} : S \to \mathbb{F}[S]^{\mathrm{op}}$ is codense, so its right Kan extension along itself is the identity: $\mathrm{Ran}_{[-]^{\mathrm{op}}}\,[-]^{\mathrm{op}} \cong \mathrm{Id}$. As $(-)^\times$ is a right Kan extension along $[-]^{\mathrm{op}}$, we can calculate:

$$\mathcal{J}^\times = (\mathcal{H} \circ [-]^{\mathrm{op}})^\times \cong \mathcal{H} \circ ([-]^{\mathrm{op}})^\times \cong \mathcal{H}$$

where $\mathcal{H} : \mathbb{F}[S]^{\mathrm{op}} \to \widetilde{\mathbb{F}[S]}$ is cartesian (turning coproducts in $\mathbb{F}[S]$ into products of hom-sets) and therefore Corollary 9.2.5 applies. □

Corollary 9.2.6 *The structure maps of Lemma 9.2.2 and i of Lemma 9.2.3 are invertible.*

Proof The lemmas above allow us to turn the one-way maps in the definition of the structure maps into isomorphisms. □

Thus, with our strengthened condition of $\mathbb{F}[S]$ being a free cocartesian category, we recover the expected result, but having arrived at it from first principles.

Theorem 9.2.2

The category $\widetilde{\mathbb{F}[S]}^S$ over the free cocartesian category on S is strong monoidal closed, with unit $\mathcal{J} \in \widetilde{\mathbb{F}[S]}^S$, tensor $\otimes : \widetilde{\mathbb{F}[S]}^S \times \widetilde{\mathbb{F}[S]}^S \to \widetilde{\mathbb{F}[S]}^S$, and hom $[-,=] : (\widetilde{\mathbb{F}[S]}^S)^{\mathrm{op}} \times \widetilde{\mathbb{F}[S]}^S \to \widetilde{\mathbb{F}[S]}^S$.

In the presheaf model, with $\mathbb{F}[S]$ consisting of lists freely generated from a set of sorts S along a dense and fully faithful $J : S \to \mathbb{F}[S]$, and morphisms corresponding to arbitrary renamings of elements of the list, the preconditions of the above result are satisfied. List concatenation is the coproduct, with the injections $\Gamma \to \Gamma + \Delta$ and $\Delta \to \Gamma + \Delta$ representing weakening and copairing of $\Gamma \to \Theta$ and $\Delta \to \Theta$ to $(\Gamma + \Delta) \to \Theta$ implemented by case-analysis on the input.

9.3 Substitution through warping

In the previous section, we showed how skew-monoidal structure on a presheaf category emerges from the definition of substitution operators and brackets (for any base category), and how the unit, associator, and compositor become invertible under stronger assumptions – specifically, when the base category is the free cocartesian category on a set of sorts. However, this richer structure is difficult to formalize: implementing the presheaf model directly results in a computationally impractical system that requires numerous workarounds in a dependently-typed setting. Conversely, the weaker skew-monoidal structure is too rigid to model syntactic substitution, especially in its handling of contexts and renamings (see the end of Section 9.2.1 and Section 9.3.2). If presheaves over a discrete category of contexts are too weak to express key syntactic constructs (such as variables, renaming, and substitution), and presheaves over the cocartesian category are too unwieldy to formalize constructively, we must seek a middle ground: a lightweight theory of context-indexed families of sets with just enough renaming structure to serve as a foundation for abstract syntax. The categorical tool of adjoint warpings, introduced in Section 7.1, provides precisely this framework.

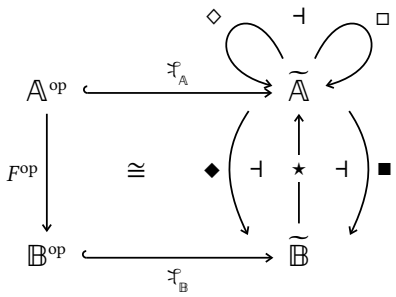
9.3.1 Adjoint modalities

Recall, from Theorem 8.1.1, the adjoint triple induced by a functor $F : \mathbb{A} \to \mathbb{B}$ between the corresponding presheaf categories, with $\mathrm{Lan}_F \dashv F^* \dashv \mathrm{Ran}_F$ denoted $\blacklozenge \dashv \star \dashv \blacksquare$. Suppose, to this end, that we have two small categories connected by a functor $F : \mathbb{A} \to \mathbb{B}$, where $\mathbb{B}$ will generally have a richer structure that $\mathbb{A}$ embeds into. The precomposition functor $\star = F^* = P \mapsto P \circ F : \widetilde{\mathbb{B}} \to \widetilde{\mathbb{A}}$ then acts as a forgetful functor that turns "richer" presheaves into "simpler" ones. This process is adjoint to universal procedures of equipping an $\mathbb{A}$-presheaf with extra structure to admit functoriality over $\mathbb{B}$: the left Kan extension $\mathrm{Lan}_F = \blacklozenge$ constructs a free $\mathbb{B}$-presheaf, while the right one $\mathrm{Ran}_F = \blacksquare$ the cofree $\mathbb{B}$-presheaf.

We generalise our running example of naturals that embed into finite sets, with the embedding $n \in \mathbb{N} \mapsto [n] \in \mathbb{F}$ inducing the forgetful functor $\star : \widetilde{\mathbb{F}} \to \widetilde{\mathbb{N}}$. Let $\mathbb{A}$ be the discrete category underlying $\mathbb{B}$, so presheaves over $\mathbb{A}$ are merely sets indexed by objects with no inherent

reindexing operation (as the hom-sets $\mathbb{A}(a,b)$ are inhabited only if $a = b$), and the forgetful functor $\star\colon \widetilde{\mathbb{B}} \to \widetilde{\mathbb{A}}$ maps a "proper" presheaf to its underlying indexed set. The extensions $\blacklozenge, \blacksquare\colon \widetilde{\mathbb{A}} \to \widetilde{\mathbb{B}}$ are the co/free ways of turning an indexed set into a $\mathbb{B}$-presheaf that can be reindexed along a morphism $f \in \mathbb{B}(a,b)$: $\blacklozenge Xf\colon \blacklozenge X(a) \to \blacklozenge X(b)$ and $\blacksquare Xf\colon \blacksquare X(a) \to \blacksquare X(b)$. The adjoint triple also induces an adjoint monad-comonad pair $\Diamond \dashv \Box$ on $\mathbb{A}$ that we will be using extensively.

Proposition 9.3.1 *We have the following adjoint situation:*



where the adjoint monad-comonad pair $\Diamond \dashv \Box\colon \widetilde{\mathbb{A}} \to \widetilde{\mathbb{A}}$ *is*

$$\Diamond X = \blacklozenge X \circ F\colon \mathbb{A} \to \mathbf{Set} \qquad \Box X = \blacksquare X \circ F\colon \mathbb{A} \to \mathbf{Set}$$

Proof The adjoint triple on the right is an instance of Theorem 8.1.1, for $\mathcal{C} = \mathbf{Set}$. We also need to show that

$$\blacklozenge \circ \mathcal{F}_{\mathbb{A}} \cong \mathcal{F}_{\mathbb{B}} \circ F^{\mathrm{op}}\colon \mathbb{A}^{\mathrm{op}} \to \mathbf{Set}^{\mathbb{B}} \tag{$\blacklozenge\mathcal{F}$}$$

We take $a \in \mathbb{A}$ and $b \in \mathbb{B}$, and calculate:

$$\begin{aligned} \blacklozenge(\mathcal{F}_{\mathbb{A}} a)b &\cong \int^{a' \in \mathbb{A}} \mathbb{B}(Fa', b) \times \mathcal{F}_{\mathbb{A}}(a)(a') && \text{(Lan}\textstyle\int\text{)} \\ &= \int^{a' \in \mathbb{A}} \mathbb{B}(Fa', b) \times \mathbb{A}(a, a') \\ &\cong \mathbb{B}(Fa, b) = \mathcal{F}_{\mathbb{B}}(Fa)(b) && (\textstyle\int\mathcal{F}\times) \end{aligned}$$

□

Being adjoints, the co/free functors and modalities satisfy various co/continuity properties.

This connection between the presheaf categories $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ is canonically induced by any functor $F\colon \mathbb{A} \to \mathbb{B}$, but it deepens in situations when F is an embedding in a more formal sense: it is bijection on objects. From this condition (satisfied, for example, by our running example of a discrete category embedding) we obtain an equivalent characterisation of a "rich" $\mathbb{B}$-presheaf as an object of $\widetilde{\mathbb{A}}$ with additional co/algebra structure.

Theorem 9.3.1

If $F\colon \mathbb{A} \to \mathbb{B}$ *is bijective-on-objects, we have the following equivalence of categories:*

$$\Diamond\text{-}\mathbf{Alg}(\widetilde{\mathbb{A}}) \simeq \widetilde{\mathbb{B}} \simeq \Box\text{-}\mathbf{Coalg}(\widetilde{\mathbb{A}})$$

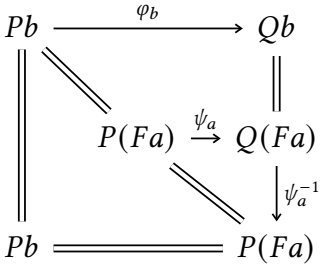
Proof We reason by showing that the precomposition functor $\star\colon \widetilde{\mathbb{B}} \to \widetilde{\mathbb{A}}$ is co/monadic, which gives us the required equivalences. Beck's (1968) crude monadicity theorem says that $\star$ is monadic if the following properties hold:

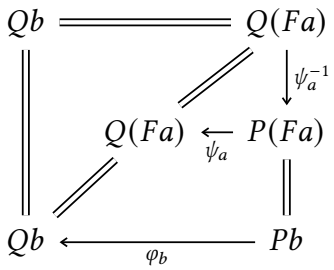
It has a left adjoint By assumption, we have the left Kan extension $\mathrm{Lan}_F : \widetilde{\mathbb{A}} \to \widetilde{\mathbb{B}}$.

It reflects isomorphisms Let $\varphi\colon P \to Q$ be a morphism in $\widetilde{\mathbb{B}}$. We show that if $\star\varphi\colon \star P \to \star Q$ is an isomorphism, then so is φ. To this end, let $\psi = \star\varphi\colon P\circ F \to Q\circ F$ and $\psi^{-1}\colon Q\circ F \to P\circ F$ be the isomorphism in $\widetilde{\mathbb{A}}$, and define $\varphi^{-1}\colon Q \to P$ for all $b \in \mathbb{B}$ as follows:

$$\varphi^{-1}_b\colon Qb = Q(Fa) \xrightarrow{\psi^{-1}_a} P(Fa) = Pb$$

where $a = F^{-1}b$ is unique since F is bijective-on-objects. Then, the inverse laws follow from those of ψ, and the fact that $\psi_a = (\star\varphi)_a = \varphi_{Fa} = \varphi_b$.





It preserves reflexive coequalisers $\star$ is left adjoint to Ran_F so preserves colimits, including reflexive coequalisers.

Comonadicity of $\star$ is established analogously: Ran_F is the right adjoint, and $\mathrm{Lan}_F \dashv \star$ so $\star$ also preserves limits, including reflexive equalisers. □

The theorem states that if F is a bijective-on-objects embedding of a simpler category $\mathbb{A}$ into a richer $\mathbb{B}$, then presheaves over $\mathbb{B}$ can be equivalently represented as $\Diamond$-algebras or $\Box$-coalgebras in $\widetilde{\mathbb{A}}$, with natural transformations corresponding to co/algebra homomorphisms. The key advantage of this representation is that it separates the object-level data from the functorial action, expressing a rich presheaf as a simpler one equipped with extra structure. In the extreme case where $\mathbb{A}$ is the discrete category underlying $\mathbb{B}$, every presheaf $P \in \widetilde{\mathbb{B}}$ is equivalent to its underlying indexed family $X = \{Pa\}_{a\in\mathbb{A}}$, along with an algebra structure $\Diamond X \to X$ or a coalgebra structure $X \to \Box X$. Framed this way, the result is intuitive: the co/algebra structure encodes precisely the functorial action:

$$\begin{aligned}
&\widetilde{\mathbb{A}}(\Diamond X, X) && &&\widetilde{\mathbb{A}}(X, \Box X)\\
&\cong \int_{a'\in\mathbb{A}} \mathbf{Set}(\Diamond X a', X a') && (\textstyle\int\mathrm{NT}) &&\cong \int_{a\in\mathbb{A}} \mathbf{Set}(X a, \Box X a) && (\textstyle\int\mathrm{NT})\\
&\cong \int_{a'\in\mathbb{A}} \mathbf{Set}\Big(\int^{a\in\mathbb{A}} \mathbb{B}(Fa, Fa') \times X a, X a' \Big) && (\mathrm{Lan}\textstyle\int) &&\cong \int_{a\in\mathbb{A}} \mathbf{Set}\Big(X a, \int_{a'\in\mathbb{A}} \mathbf{Set}\big(\mathbb{B}(Fa, Fa'), X a'\big) \Big) && (\mathrm{Ran}\textstyle\int)\\
&\cong \int_{a,a'\in\mathbb{A}} \mathbf{Set}\big(\mathbb{B}(Fa, Fa'), \mathbf{Set}(X a, X a')\big) && (\textstyle\int\updownarrow) &&\cong \int_{a,a'\in\mathbb{A}} \mathbf{Set}\big(\mathbb{B}(Fa, Fa'), \mathbf{Set}(X a, X a')\big) && (\textstyle\int\leftrightarrow)
\end{aligned}$$

and F is bijective-on-objects, any morphism $\mathbb{B}(b, b')$ can be written as $\mathbb{B}(Fa, Fa')$ for some $a, a' \in \mathbb{A}$, allowing us to functorially apply a $\mathbb{B}$-morphism to any $X \in \widetilde{\mathbb{A}}$. By developing the

theory in terms of co/algebras in $\widetilde{\mathbb{A}}$, we can precisely identify when a co/algebra structure (i.e. functoriality) is necessary, and when a simple indexed family of sets suffices. This approach also improves computational tractability: rather than manipulating complex presheaf constructions throughout, most of the formalisation relies on straightforward indexed sets, with co/algebra structures introduced only where required.

9.3.2 Warped substitution

The abstract theory of Section 9.2 shows that the definition of substitution operators equips the category of presheaves $\widetilde{\mathbb{C}}$ with skew-monoidal structure no matter what $\mathbb{C}$ is. For $\mathbb{C} = \mathbb{F}[S]$, the structure strengthens to a strong monoidal one, giving rise to the presheaf model with the syntactically intuitive interpretation for the categorical structure: variables $\mathcal{J}_\tau\Gamma$ correspond to the predicate $\tau \in \Gamma$, ${}^{\Gamma}\mathcal{Q}_{\Delta}$ is a simultaneous substitution rule from variables in Γ to $\mathcal{Q}$-terms in Δ, and elements of the tensor $(\Gamma, t, \sigma) \in (P \odot \mathcal{Q})(\Delta)$ associate a term $t \in P(\Gamma)$ with a substitution from Γ to $\mathcal{Q}$-terms in Δ. However, if we instantiate $\mathbb{C}$ with a discrete category (such as the set of lists S^*, the discrete category underlying $\mathbb{F}[S]$) the resulting notions of "variables" and "substitution" are rather strange: $\mathcal{J}_\tau\Gamma$ is only defined if $\Gamma = [\tau]$, a substitution rule ${}^{\Gamma}\mathcal{Y}_{\Delta}$ is a term $\mathcal{Y}_\tau\Delta$ if $\Gamma = [\tau]$ and the singleton set $\{*\}$ otherwise, and $(X \odot \mathcal{Y})(\Delta)$ contains tuples $([\tau], t \in X[\tau], s \in \mathcal{Y}_\tau\Delta)$ for $\Gamma = [\tau]$, or $(\Gamma, t \in X\Gamma, *)$ if $\Gamma \neq [\tau]$. Clearly, this is very restrictive and wouldn't be suitable for the representation of syntactic metatheory.

How do we bring this intuitive structure to the category of indexed families? Looking at the internals of the monoidal structure proof laid out in Section 9.2.1, that the only place the base category $\mathbb{C}$ appears in is the embedding $\mathcal{J}^{\mathbb{C}}_\tau(a) = \mathbb{C}(J\tau, a)$, and, consequently, the extension ${}^{a}\mathcal{Q}_{b} = \mathcal{Q}^{\times_{\mathbb{C}}}(a)(b) \cong \int_{\tau\in S} \mathcal{J}^{\mathbb{C}}_\tau(a) \Rightarrow \mathcal{Q}_\tau(b)$. That is, even if $\mathcal{Y} \in \widetilde{\mathbb{A}}^S$ is merely an indexed family of sets, the substitution $\mathcal{Y}^{\times_{\mathbb{B}}}(\Gamma)(\Delta) = \int_{\tau\in S} \mathcal{J}^{\mathbb{B}}_\tau\Gamma \Rightarrow \mathcal{Y}_\tau\Delta$ behaves in the expected way: mapping variables $\tau \in \Gamma$ to $\mathcal{Y}$-terms $\mathcal{Y}_\tau\Delta$. The choice of this category is determined by the first operand of the substitution action $(-) \oslash^{\mathbb{B}} (=) \colon \widetilde{\mathbb{B}}^S \times \widetilde{\mathbb{A}}^S \to \widetilde{\mathbb{A}}^S$, which associates a $\mathbb{B}$-presheaf with a substitution that maps $\mathbb{B}$-variables to $\mathbb{A}$-presheaves. We can exploit this "mixed" operator (a left action as given in Corollary 9.2.4) to construct an appropriate substitution tensor product on $\widetilde{\mathbb{A}}^S$: turn the first operand $\mathcal{X}$ into a $\mathbb{B}$-presheaf, then apply the action $\oslash^{\mathbb{B}}$ with another sorted $\mathbb{A}$-presheaf $\mathcal{Y}$ to obtain an indexed family that nevertheless behaves like a tensor of $\mathbb{B}$-presheaves. The crucial step here – turning a $\mathbb{A}$-presheaf into a $\mathbb{B}$ presheaf – is precisely the job of the free presheaf functor $\blacklozenge \colon \widetilde{\mathbb{A}}^S \to \widetilde{\mathbb{B}}^S$.

The story is similar for the substitution bracket and enrichment. The output category of $\langle -, = \rangle^{\mathbb{C}}$ determines the base category of embeddings, so taking the enrichment of two sorted $\mathbb{A}$-presheaves with $\langle -, = \rangle^{\mathbb{B}} \colon (\widetilde{\mathbb{A}}^S)^{\mathrm{op}} \times \widetilde{\mathbb{A}}^S \to \widetilde{\mathbb{B}}^S$, then converting the output back into $\widetilde{\mathbb{A}}^S$ using the forgetful functor $\bigstar \colon \widetilde{\mathbb{B}}^S \to \widetilde{\mathbb{A}}^S$ would give us an internal hom on $\widetilde{\mathbb{A}}^S$. Rather than verifying that the tensor and hom thus obtained indeed make $\widetilde{\mathbb{A}}^S$ into a skew-monoidal category, we make use of the generic framework of adjoint warpings as developed in Section 7.1.3 where all the hard work has already been done in the abstract.

Proposition 9.3.2 *The functors $\blacklozenge \dashv \bigstar \colon \widetilde{\mathbb{A}}^S \to \widetilde{\mathbb{B}}^S$ form adjoint warpings over $\widetilde{\mathbb{B}}^S$.*

Proof The category $\widetilde{\mathbb{B}}^S$ is skew-monoidal closed by Theorem 9.2.1 (also monoidal closed, but we do not need this here) and $\oslash \colon \widetilde{\mathbb{B}}^S \times \widetilde{\mathbb{A}}^S \to \widetilde{\mathbb{A}}^S$ is a left $\widetilde{\mathbb{B}}^S$-action. Applying Theorem 7.1.4 to

- the adjoint triple $\blacklozenge \dashv \star \dashv \blacksquare \colon \widetilde{\mathbb{A}}^S \to \widetilde{\mathbb{B}}^S$ constructed via Theorem 8.1.1;
- the strong $\widetilde{\mathbb{B}}^S$-module functor $\star \colon \widetilde{\mathbb{B}}^S \to \widetilde{\mathbb{A}}^S$, with isomorphism $\mathcal{P} \oslash \star\mathcal{Q} \cong \star(\mathcal{P} \otimes \mathcal{Q})$ by Corollary 9.2.5

gives us the required adjoint warping structure on $\blacklozenge \dashv \star$, along the way also inducing the maps and isomorphisms

$$[\mathcal{P}, \mathcal{Q}] \to \langle \star\mathcal{P}, \star\mathcal{Q} \rangle \qquad \langle \mathcal{X}, \mathcal{Y} \rangle \to [\blacksquare\mathcal{X}, \blacksquare\mathcal{Y}] \qquad \langle \star\mathcal{P}, \mathcal{Y} \rangle \cong [\mathcal{P}, \blacksquare\mathcal{Y}] \qquad \square$$

Using Corollary 7.1.1, we transport the skew-monoidal closed structure of $\widetilde{\mathbb{B}}^S$ onto $\widetilde{\mathbb{A}}^S$.

Theorem 9.3.2

Given $F \colon \mathbb{A} \to \mathbb{B}$, the category $\widetilde{\mathbb{A}}^S$ is skew-monoidal closed, with warped unit and operators

$$\mathcal{I} \triangleq \star\mathcal{J} \qquad \mathcal{X} \oplus \mathcal{Y} \triangleq (\blacklozenge\mathcal{X}) \oslash^{\mathbb{B}} \mathcal{Y} \qquad [\![\mathcal{X}, \mathcal{Y}]\!] \triangleq \star\langle \mathcal{X}, \mathcal{Y} \rangle^{\mathbb{B}}$$

Though not part of the warping framework, the actions corresponding to the operators are induced in a similar way.

Corollary 9.3.1 *The unsorted versions of the operators constitute left $\widetilde{\mathbb{C}}^S$-actions:*

$$X \ominus \mathcal{Y} \triangleq (\blacklozenge X) \odot^{\mathbb{B}} \mathcal{Y} \qquad [\![\mathcal{X},\ Y]\!\rangle \triangleq \star\lfloor \mathcal{X}, Y \rceil^{\mathbb{B}}$$

Corollary 9.3.2 *We have the isomorphisms of categories*

$$\mathcal{I}\text{-}\mathbf{RMod}^{\oplus}(\widetilde{\mathbb{A}}) \cong \Diamond\text{-}\mathbf{Alg}(\widetilde{\mathbb{A}}) \cong \widetilde{\mathbb{B}} \cong \Box\text{-}\mathbf{Coalg}(\widetilde{\mathbb{A}}) \cong \mathcal{I}\text{-}\mathbf{RMod}^{[\![]\!]}(\widetilde{\mathbb{A}})$$

Proof A combination of Theorem 9.3.1 and Corollary 7.2.1. $\square$

When one operand of a presheaf operator is co/free, the underlying family of the tensor or hom is equivalently given by a family operator.

Lemma 9.3.1 *We have the following isomorphisms:*

$$X \oplus (\star\mathcal{Q}) \cong \star(\blacklozenge X \otimes \mathcal{Q}) \qquad [\![\star\mathcal{P}, Y]\!] \cong \star[\mathcal{P}, \blacksquare Y]$$

Proof We calculate as follows, using the definitions of the warped operators and the isomorphisms $\mathcal{P} \oslash \star\mathcal{Q} \cong \star(\mathcal{P} \otimes \mathcal{Q})$ and $\langle \star\mathcal{P}, \mathcal{Y} \rangle \cong [\mathcal{P}, \blacksquare\mathcal{Y}]$

$$X \oplus (\star\mathcal{Q}) = (\blacklozenge X) \oslash (\star\mathcal{Q}) \cong \star(\blacklozenge X \otimes \mathcal{Q}) \qquad [\![\star\mathcal{P}, Y]\!] = \star\langle \star\mathcal{P}, Y \rangle \cong \star[\mathcal{P}, \blacksquare Y] \qquad \square$$

The warped substitution operators give us the valuable abstract results of induced skew-monoidal structure, but, still being defined in terms of the operations over $\widetilde{\mathbb{B}}^S$, they are impractical: in the formalisation, we need to avoid referring to $\mathbb{B}$-presheaves as they cannot be captured exactly. The substitution structure of $\mathbb{A}$-presheaves (which will simply be indexed families of sets) may be defined more directly by a change-of-base-like construction.

9.3.3 Rebased substitution

In the setting of families, a more direct characterisation of substitution structure exists, which can be shown to be equivalent to the one induced by warping. It directly captures the intu-

ition at the beginning of the chapter of working with $\mathbb{A}$-presheaves, but taking the cartesian extension in the richer base category of $\mathbb{B}$. As before, fix a functor $F\colon \mathbb{A} \to \mathbb{B}$.

Definition 9.3.1 The *rebased extension* of a sorted object $\mathcal{Q} \in \mathcal{D}^S$ is defined as

$$\mathcal{Q}^{\times_F} \triangleq \mathcal{Q}^{\times_{\mathbb{B}}} \circ F^{\mathrm{op}}\colon \mathbb{A}^{\mathrm{op}} \to \mathcal{D}$$

The *rebased substitution operator* $(-) \odot^F (=)\colon \widetilde{\mathbb{A}} \times \mathcal{D}^S \to \mathcal{D}$ is:

$$(-) \odot^F \mathcal{Q} \triangleq \mathrm{Re}_{\mathcal{Q}^{\times_F}} : \widetilde{\mathbb{A}} \to \mathcal{D} \qquad (X \odot^F \mathcal{Q})(b) \cong \int^{a \in \mathbb{A}} Xa \times \int_{\tau \in S} \mathbb{B}(J_{\mathbb{B}}\tau, Fa) \Rightarrow \mathcal{Q}_\tau(b)$$

and, correspondingly, the *rebased substitution action* $(-) \oslash^F (=)\colon \widetilde{\mathbb{A}}^S \times \mathcal{D}^S \to \mathcal{D}^S$ is:

$$(\mathcal{X} \oslash^F \mathcal{Q})_\tau \triangleq \mathcal{X}_\tau \odot^F \mathcal{Q}$$

Similarly, we define the *rebased substitution bracket* $\lfloor -, = \rceil^F\colon (\mathcal{D}^S)^{\mathrm{op}} \times \mathcal{D} \to \widetilde{\mathbb{A}}$ and *enrichment* $\langle -, = \rangle^F\colon (\mathcal{D}^S)^{\mathrm{op}} \times \mathcal{D}^S \to \widetilde{\mathbb{A}}^S$:

$$\lfloor \mathcal{Q}, Z \rceil^F(a) \triangleq \mathrm{Nr}_{\mathcal{Q}^{\times_F}}(Z)(a) \cong \mathcal{D}\big(\mathcal{Q}^{\times_{\mathbb{B}}}(Fa), Z\big) \qquad \langle \mathcal{Q}, \mathcal{Z} \rangle^F_\tau \triangleq \lfloor \mathcal{Q}, \mathcal{Z}_\tau \rceil$$ ⌟

Remark. Note that if F were full and faithful, the mixed extension would collapse to the $\mathbb{A}$-extension:

$$\mathcal{Y}^{\times_{\mathbb{B}}}(Fa) = \int_{\tau \in S} \mathbb{B}(J_{\mathbb{B}}\tau, Fa) \pitchfork \mathcal{Y}_\tau \cong \int_{\tau \in S} \mathbb{B}(F(J_{\mathbb{A}}\tau), Fa) \pitchfork \mathcal{Y}_\tau \cong \int_{\tau \in S} \mathbb{A}(J_{\mathbb{A}}\tau, a) \pitchfork \mathcal{Y}_\tau = \mathcal{Y}^{\times_{\mathbb{A}}}(a)$$

In our case, F will be a discrete subcategory embedding, which is certainly not fully faithful. ⌟

The rebased substitution operations are defined in terms of precomposition with F (same as the forgetful functor $\star\colon \widetilde{\mathbb{B}} \to \widetilde{\mathbb{A}}$), conceptually simpler than the warping via the left Kan extension $\blacklozenge$. We can show that the two structures are nevertheless equivalent using some lemmas concerning the adjoint triple $\blacklozenge \dashv \star \dashv \blacksquare$, nerves, and realisations.

Lemma 9.3.2 *For $P\colon \mathbb{C}^{\mathrm{op}} \to \widetilde{\mathbb{B}}$, we have the isomorphisms*

$$\star \circ \mathrm{Re}_P \cong \mathrm{Re}_{\star \circ P} \quad (\star\mathrm{Re}) \qquad \mathrm{Nr}_P \circ \blacksquare \cong \mathrm{Nr}_{\star \circ P} \quad (\mathrm{Nr}\blacksquare)$$

Proof $\star$ is a left adjoint, so preserves the left Kan extension that defines Re_P. The second isomorphism is a right adjoint of the second: $\star \dashv \blacksquare$ and $\mathrm{Re}_P \dashv \mathrm{Nr}_P$ composes to the adjunction $\star \circ \mathrm{Re}_P \dashv \mathrm{Nr}_P \circ \blacksquare$, and the latter is isomorphic to the unique right adjoint $\mathrm{Nr}_{\star \circ P}$ of $\mathrm{Re}_{\star \circ P}$. □

Lemma 9.3.3 *For $G\colon \mathbb{B}^{\mathrm{op}} \to \mathcal{D}$, we have the isomorphisms*

$$\mathrm{Re}_G \circ \blacklozenge \cong \mathrm{Re}_{G \circ F^{\mathrm{op}}} \quad (\mathrm{Re}\blacklozenge) \qquad \star \circ \mathrm{Nr}_G \cong \mathrm{Nr}_{G \circ F^{\mathrm{op}}} \quad (\star\mathrm{Nr})$$

Proof We prove the second isomorphism; the first is its left adjoint. For $C \in \mathbb{C}$, we have

$$(\star \circ \mathrm{Nr}_G)(C) \cong \mathbb{C}(G(F-), C) = \mathbb{C}((G \circ F^{\mathrm{op}})-, C) \cong \mathrm{Nr}_{G \circ F^{\mathrm{op}}}(C)$$ □

Proposition 9.3.3 *The warped and the rebased substitution structures are equivalent:*

$$X \ominus \mathcal{Y} \cong X \odot^F \mathcal{Y} \qquad [\![\mathcal{X}, Y]\!\rangle \cong \lfloor \mathcal{X}, Y \rceil^F$$

Proof The first isomorphism is an instance of Eq. (Re◆), and the second is its right adjoint:

$$(-) \ominus \mathcal{Y} \triangleq \blacklozenge(-) \odot^{\mathbb{B}} \mathcal{Y} = \mathrm{Re}_{\mathcal{Y} \times \mathbb{B}} \circ \blacklozenge \cong \mathrm{Re}_{\mathcal{Y} \times \mathbb{B} \circ F^{\mathrm{op}}} = \mathrm{Re}_{\mathcal{Y} \times F} = (-) \odot^{F} \mathcal{Y} \qquad \square$$

Thus, the results for the warped substitution structure equivalently apply to the rebased one, so the two can be used interchangeably in the theory. In practice, working with the rebased definitions is more amenable to formalisation, as it doesn't assume the existence of the monoidal structure on $\widetilde{\mathbb{B}}^{S}$.

To summarise the results of this section, we use the warping induced by the adjoint triple $\blacklozenge \dashv \star \dashv \blacksquare$ to transport the monoidal-closed structure of $\widetilde{\mathbb{B}}^{S}$ to a skew-monoidal closed structure on $\widetilde{\mathbb{A}}^{S}$, and a skew $\widetilde{\mathbb{A}}^{S}$-module structure on $\widetilde{\mathbb{A}}$. The category of co/algebras on $\widetilde{\mathbb{C}}$ for the induced co/monad is then isomorphic to the categories of right modules, and the category of $\mathbb{B}$-presheaves $\widetilde{\mathbb{B}}$. The warped structure is equivalent to the one defined via rebasing along the functor F, which is conceptually simpler and possible to formalise. This sets up the universe of discourse for the familial model, to be discussed next.

CHAPTER 10

Discrete families

The theory of presheaves developed in the previous section will now be instantiated with a specific base category of contexts and renamings, giving rise to the familial model for second-order abstract syntax. We start by summarising the core definitions and fixing notation, then step through many concepts introduced in previous chapters, concretising them in the setting of sorted families of sets.

10.1 Families as a model of syntax

Before setting out to prove deeper results connecting families with presheaves, we first need to set up our universe of discourse and justify our claim that indexed families of sets can serve as a model of second-order abstract syntax. At the very least, this should involve the support for *variables*, *substitution*, and *algebraic structure* over a signature. These are conveniently packaged up in the notion of an *algebraic monoid* that we build towards in this section.

As we will be dealing with *simply-sorted syntax*, we fix a set of sorts S throughout this chapter, with elements denoted α, β, τ, etc. Unlike Arkor and Fiore (2020), we do not permit any algebraic structure on the syntax of sorts itself, keeping them as opaque, unrelated elements in the model; this is of course not to say that S won't actually be generated from a simple grammar of types.

10.1.1 Contexts and variables

Following the presheaf model and the principle of intrinsic typing and scoping, terms keep track of their own sort and context: the set of all possible terms of the syntax is stratified by sorts and typing contexts, distinguishing different terms by their indices.

Definition 10.1.1 The *category of contexts* $\mathbb{F}[S]$ is the free cocartesian category on the set of sorts S. Explicitly, its objects are finite lists of sorts generated by the initial *empty context* $\emptyset$ and the *cons operator* $\alpha \cdot (-) \colon \mathbb{F}[S] \to \mathbb{F}[S]$ for all $\alpha \in S$, and morphisms are arbitrary functions between elements of the lists taken as finite sets. The coproduct of the free cocartesian category

is the *concatenation operator* $(-) + (=) : \mathbb{F}[S] \times \mathbb{F}[S] \to \mathbb{F}[S]$, defined recursively as

$$\emptyset + \Delta \triangleq \Delta$$
$$(\alpha \cdot \Gamma) + \Delta \triangleq \alpha \cdot (\Gamma + \Delta)$$

The singleton list containing $\alpha \in S$ will be denoted $[\alpha]$. The *discrete contexts* over S is the discrete category underlying $\mathbb{F}[S]$, denoted S^*. ⌟

As $\mathbb{F}[S]$ is freely generated, every object is either the initial object $\emptyset$, or of the form $\alpha \cdot \Gamma = [\alpha] + \Gamma$ with coproduct structure

$$[\alpha] \xrightarrow{\mathsf{new}_{\alpha,\Gamma}} [\alpha] + \Gamma \xleftarrow{\mathsf{old}_{\alpha,\Gamma}} \Gamma$$

Definition 10.1.2 From now on, we will call the category of *presheaves* the presheaf category $\mathbf{PSh} \triangleq \widetilde{\mathbb{F}[S]}$, and the category of *families* the discrete presheaf category $\mathbf{Fam} \triangleq \widetilde{S^*}$ consisting of list-indexed families of sets and indexed functions between them (since naturality is trivial over discrete categories). Sorted presheaves and sorted families are further indexed by the set of sorts S, and denoted $\mathbf{PSh}_S = (\widetilde{\mathbb{F}[S]})^S$ and $\mathbf{Fam}_S = (\widetilde{S^*})^S$. In either case, the forgetful functor induced by precomposing with the canonical injection $S^* \to \mathbb{F}[S]$ is denoted $\star : \mathbf{PSh} \to \mathbf{Fam}$. ⌟

Notation. Sorted objects will be written using calligraphic font: $\mathcal{P}, \mathcal{Q}$ for sorted presheaves, $\mathcal{X}, \mathcal{Y}$ for sorted families. If it is unambiguous, functors and operators on sorted categories will not be distinguished from unsorted versions: for example, $(\mathcal{P} \times \mathcal{Q})_\alpha = \mathcal{P}_\alpha \times \mathcal{Q}_\alpha$. ⌟

Definition 10.1.3 The *presheaf of variables* $\mathcal{V} \in \mathbf{PSh}_S$ and *family of indices* $\mathcal{I} \in \mathbf{Fam}_S$ are given for all $\alpha \in S$ as:

$$\mathcal{V}_\alpha \triangleq \text{Ⓨ}_{\mathbb{F}[S]}[\alpha] \qquad \mathcal{I} \triangleq \star\mathcal{V}$$ ⌟

As contexts have an inductive structure, so do variables: they can only live in nonempty contexts $\alpha \cdot \Gamma$, and correspond to a left injection new at the appropriate sort followed by a finite number of right injections old. For example, given the context $[\alpha, \beta, \gamma, \delta]$, the third variable of sort γ corresponds to the composite of the following injections:

$$\mathcal{V}_\gamma[\alpha, \beta, \gamma, \delta] = \mathbb{F}[S]([\gamma], [\alpha, \beta, \gamma, \delta]) = [\gamma] \xrightarrow{\mathsf{new}_{\gamma,[\delta]}} [\gamma, \delta] \xrightarrow{\mathsf{old}_{\beta,[\gamma,\delta]}} [\beta, \gamma, \delta] \xrightarrow{\mathsf{old}_{\alpha,[\beta,\gamma,\delta]}} [\alpha, \beta, \gamma, \delta]$$

Thus, we can pattern-match on a variable $v \in \mathcal{V}_\tau(\alpha \cdot \Gamma)$ as being either $\mathsf{new}_{\alpha,\Gamma} \in \mathcal{V}_\alpha(\alpha \cdot \Gamma)$ or $\mathsf{old}_{\tau,\Gamma}(u) \in \mathcal{V}_\tau(\alpha \cdot \Gamma)$ for a $u \in \mathcal{V}_\tau(\Gamma)$. As we will next see, morphisms in $\mathbb{F}[S]$ may be equivalently characterised by sort-preserving mappings of indices.

10.1.2 Substitution structure

The substitution structure of families can formally be constructed from that of presheaves as laid out in Chapter 9. We build up the definitions from bottom up, indicating where they appear in the abstract development.

Substitution rules derive from cartesian extensions of presheaves (Definition 9.2.1), equivalently represented by a tuple of $\mathcal{X}$-terms in Δ with the sorts taken in Γ, or a sort-preserving mapping of variables in Γ to $\mathcal{X}$-terms in Δ.

Definition 10.1.4 The *Γ-wise product* of a sorted family $\mathcal{X}$ is defined as:

$$\mathcal{X}^{\times\Gamma} \triangleq \prod_{\alpha\in\Gamma} \mathcal{X}_\alpha$$

At a context Δ, this is equivalently the function space

$$\mathcal{X}^{\times\Gamma}(\Delta) \cong \prod_{\alpha\in S} \mathcal{I}_\alpha\Gamma \to \mathcal{X}_\alpha\Delta$$

since membership in Γ is captured by the inhabitance of the variable presheaf $\mathcal{V}_\alpha\Gamma \triangleq \mathbb{F}([\alpha],\Gamma)$ with $\mathcal{I}$ as its underlying family, which is nonempty only when α appears in Γ. The set $\mathcal{X}^{\times\Gamma}(\Delta)$ will be denoted ${}^{\Gamma}\mathcal{X}_\Delta$ and generally called a *substitution rule* from Γ to Δ. ⌟

Associating a term with a substitution rule (either as a tuple, or a dependence relationship) gives rise to the skew-monoidal closed structure of families. The definitions instantiate the rebased substitution operations from Section 9.3.3, which are equivalent to the structure induced by the adjoint warping $\blacklozenge \dashv \star$:

Definition 10.1.5 The *substitution action* and *substitution tensor product* are defined as

$$(-)\ominus(=)\colon \mathbf{Fam}\times\mathbf{Fam}_S \to \mathbf{Fam} \qquad (-)\oplus(=)\colon \mathbf{Fam}_S\times\mathbf{Fam}_S\to\mathbf{Fam}_S$$
$$(X\ominus\mathcal{Y})(\Delta) \triangleq \sum_{\Gamma\in S^*} X(\Gamma)\times {}^{\Gamma}\mathcal{Y}_\Delta \qquad (\mathcal{X}\oplus\mathcal{Y})_\alpha \triangleq \mathcal{X}_\alpha\ominus\mathcal{Y}$$

The *right substitution hom* and *internal substitution hom* are defined as

$$[\![-,=\rangle\!]\colon \mathbf{Fam}_S^{\mathrm{op}}\times\mathbf{Fam}\to\mathbf{Fam} \qquad [\![-,=]\!]\colon \mathbf{Fam}_S^{\mathrm{op}}\times\mathbf{Fam}_S\to\mathbf{Fam}_S$$
$$[\![\mathcal{X},\ Y\rangle\!](\Gamma) \triangleq \prod_{\Delta\in S^*} {}^{\Gamma}\mathcal{X}_\Delta \to \mathcal{Y}(\Delta) \qquad [\![\mathcal{X},\mathcal{Y}]\!]_\tau \triangleq [\![\mathcal{X},\ \mathcal{Y}_\alpha\rangle\!]$$

⌟

As shown in several examples, the reason the substitution structure on families is merely skew-monoidal is that triples $(\Gamma,t,\rho)\in X\ominus\mathcal{Y}$ may only be compared componentwise, without any quotienting that would identify unequal triples. The tensor product of presheaves is a combination of the tensor product on families, and the quotienting condition that expresses an internal renaming-invariance.

Proposition 10.1.1 *The underlying family of the presheaf substitution action/product can be expressed by quotienting the family substitution action/product:*

$$\star(P\oslash\mathcal{Q}) \ \cong\ (\star P\ominus\star\mathcal{Q})\ /\ \approx$$

where $\approx$ is the equivalence relation generated by identifying "inter-renamable" terms: for $t\in P(\Gamma_1)$, $s\in P(\Gamma_2)$, $\sigma\colon {}^{\Gamma_1}\mathcal{Q}_\Delta$ and $\varsigma\colon {}^{\Gamma_2}\mathcal{Q}_\Delta$,

$$(\Gamma_1,t,\sigma)\sim(\Gamma_2,s,\varsigma)\in\sum_{\Gamma\in S^*} P(\Gamma)\times{}^{\Gamma}\mathcal{Q}_\Delta$$

if there exists a map $\rho\colon\Gamma_1\to\Gamma_2$ such that $\sigma=\varsigma\circ\rho$ and $s=P(\rho)(t)$.

In the presheaf model, this quotienting is responsible for bridging the gap that occur when

proving the monoidal axioms: for example, the inverse laws of the unitors $\lambda_{\mathcal{Q}} : \mathcal{V} \otimes \mathcal{Q} \to \mathcal{Q}$ and $\rho_{\mathcal{P}} : \mathcal{P} \otimes \mathcal{V} \to \mathcal{P}$, the calculation has to equate triples

$$\lambda^{-1}(\lambda(\Gamma, v, \sigma)) = ([\alpha], \mathrm{id}_{[\alpha]}, (\alpha \mapsto \sigma(v))) \qquad \rho^{-1}(\rho(\Gamma, t, \rho)) = (\Delta, \mathcal{P}(\rho)(t), \mathrm{id}_{\Delta})$$

which are both instances of the quotienting condition, as is the triangle coherence axiom:

$$(\mathcal{P} \otimes \lambda)\big(\alpha(\Delta, (\Gamma, t, \rho), \sigma)\big) = \big(\Gamma, t, \sigma \circ \rho\big) = \big(\Delta, \mathcal{P}(\rho)(t), \sigma\big) = (\rho \otimes \mathcal{R})\big(\Delta, (\Gamma, t, \varrho), \sigma\big)$$

As the family tensor product is merely a dependent sum, the triples to be equated remain unidentifiable. The situation is similar with the internal hom, where presheaves benefit from the hom $[\mathcal{P}, \mathcal{Q}]$ itself being a natural transformation: given $h \in [\![P, Q]\!]_{\alpha}\Gamma$, $\sigma \in {}^{\Gamma}\mathcal{P}_{\Delta}$ and $\rho : \Delta \to \Theta$, the naturality condition internal to the hom is $\mathcal{Q}(\rho)(h_{\Delta}\, \sigma) = h_{\Theta}(\mathcal{P}(\rho) \circ \sigma)$. Again, in families, no such condition can be imposed, as the hom is an indexed family of functions without any naturality condition. For these reasons, the weaker, skew-monoidal closed structure is required, which only asks for the "safe" directions of the structural transformations, which satisfy the skew axioms definitionally:

$$\begin{aligned}
&\lambda : \mathcal{I} \oplus \mathcal{X} \to \mathcal{X} && \rho : \mathcal{X} \to \mathcal{X} \oplus \mathcal{I} \\
&\lambda_{\Delta}\big(\Gamma, v \in \mathcal{I}_{\tau}(\Gamma), \sigma : {}^{\Gamma}\mathcal{Q}_{\Delta}\big) \triangleq \sigma(v) && \rho_{\Delta}(t : \mathcal{X}_{\tau}(\Delta)) \triangleq (\Delta, t, \mathrm{id}_{\Delta}) \\
&j : \mathcal{I} \to [\![\mathcal{X}, \mathcal{X}]\!] && i : [\![\mathcal{I}, \mathcal{X}]\!] \to \mathcal{X} \\
&j_{\Gamma}(v : \mathcal{I}_{\tau}(\Gamma)) \triangleq (\Delta, \sigma : {}^{\Gamma}\mathcal{X}_{\Delta}) \mapsto \sigma(v) && i_{\Gamma}(h : [\![\mathcal{I}, \mathcal{X}]\!]_{\tau}(\Gamma)) \triangleq h_{\Gamma}(\mathrm{id}_{\Gamma})
\end{aligned}$$

$$\begin{aligned}
&\alpha : (\mathcal{X} \oplus \mathcal{Y}) \oplus \mathcal{Z} \to \mathcal{X} \oplus (\mathcal{Y} \oplus \mathcal{Z}) \\
&\alpha_{\Theta}\big(\Delta, (\Gamma, t \in \mathcal{X}_{\tau}(\Gamma), \sigma : {}^{\Gamma}\mathcal{Y}_{\Delta}), \varsigma : {}^{\Delta}\mathcal{Z}_{\Theta}\big) \triangleq \big(\Gamma, t, (v \in \Gamma \mapsto (\Delta, \sigma(v), \varsigma)) : {}^{\Gamma}(\mathcal{Y} \oplus \mathcal{Z})_{\Theta}\big)
\end{aligned}$$

$$\begin{aligned}
&L : [\![\mathcal{Y}, \mathcal{Z}]\!] \to [\![[\![\mathcal{X}, \mathcal{Y}]\!], [\![\mathcal{X}, \mathcal{Z}]\!]]\!] \\
&L_{\Gamma}(h : [\![\mathcal{Y}, \mathcal{Z}]\!]_{\tau}(\Gamma)) \triangleq \Delta, \sigma : {}^{\Gamma}[\![\mathcal{X}, \mathcal{Y}]\!]_{\Delta} \mapsto \Theta, \varsigma : {}^{\Delta}\mathcal{X}_{\Theta} \mapsto h_{\Theta}\big(v \in \Gamma \mapsto \sigma(v)(\varsigma)\big)
\end{aligned}$$

10.1.3 Renaming structure

Presheaves are equipped with a renaming operation a priori: for $t \in P(\Gamma)$ and a renaming $\rho : \Gamma \to \Delta$, we have $P(\rho)(t) \in P(\Delta)$. For families, this sort of reindexing is not possible without equipping them with some extra structure. Fortunately, both morphisms in $\mathbb{F}[S]$ as well as the renaming operation may be fully internalised in the category of families, without reference to presheaves or even the category of contexts explicitly.

Proposition 10.1.2 *We have the following isomorphism for all* $\Gamma \in \mathbb{F}[S]$*:*

$$\mathcal{V}^{\times\Gamma} \cong \mathcal{Y}(\Gamma)$$

Proof Since every context can be decomposed into a coproduct of its sorts, and the Yoneda

embedding turns colimits into limits, we have:

$$\mathcal{V}^{\times\Gamma} = \prod_{\alpha\in\Gamma} \mathcal{V}_\alpha = \prod_{\alpha\in\Gamma} \mathcal{F}[\alpha] \cong \mathcal{F}\Big(\coprod_{\alpha\in\Gamma} [\alpha]\Big) \cong \mathcal{F}(\Gamma) \qquad \square$$

We have consequently have that the singleton set embedding $[-] : S \to \mathbb{F}[S]$ is

- fully faithful: $S(\alpha, \beta)$ is a singleton if $\alpha = \beta$, and then $\mathbb{F}[S]([\alpha], [\beta])$ contains the identity renaming only;
- dense: for $\Gamma, \Delta \in \mathbb{F}[S]$, the set of renamings $\mathbb{F}[S](\Gamma, \Delta)$ and the set of natural transformations $\mathbf{Set}^S(\mathbb{F}[S]([-], \Gamma), \mathbb{F}[S]([-], \Delta)) \cong \big(\prod_{\alpha\in S} \mathcal{V}_\alpha\Gamma \to \mathcal{V}_\alpha\Delta\big) \cong \mathcal{V}^{\times\Gamma}(\Delta)$ are equivalent.

As the underlying family of the variable presheaf is the family of indices, we get that morphisms in $\mathbb{F}[S]$ may be expressed sort-preserving maps of indices in $\mathbf{Fam}_S$:

$$\mathbb{F}[S](\Gamma, \Delta) \cong \prod_{\alpha\in S} \mathcal{I}_\alpha\Gamma \to \mathcal{I}_\alpha\Delta$$

As discussed in Section 9.3.1, the functorial action of a presheaf may be equivalently expressed as a co/algebra structure for the co/free presheaf co/monad, given by left and right Kan extensions.

Definition 10.1.6 Given a family $X \in \mathbf{Fam}$, the *free presheaf* and *cofree presheaf* are defined, respectively, as

$$\blacklozenge X(\Delta) \triangleq \sum_{\Gamma\in S^*} X(\Gamma) \times (\Gamma \to \Delta) \qquad \blacklozenge X(\varrho) \triangleq (\Gamma, x, \quad \rho : \Gamma \to \Delta) \in \blacklozenge X(\Delta) \mapsto (\Gamma, x, \varrho\circ\rho : \Gamma \to \Theta) \in \blacklozenge X(\Theta)$$

$$\blacksquare X(\Gamma) \triangleq \prod_{\Delta\in S^*} (\Gamma \to \Delta) \to X(\Delta) \qquad \blacksquare X(\rho) \triangleq p \in \blacksquare X(\Gamma) \mapsto \Theta, \varrho : (\Delta \to \Theta) \mapsto p_\Theta(\varrho \circ \rho) \in \blacksquare X(\Delta)$$

Postcomposition by the forgetful functor $\star : \mathbf{PSh} \to \mathbf{Fam}$ gives the co/free presheaf co/monads $\Diamond, \Box : \mathbf{Fam} \to \mathbf{Fam}$ which are adjoint as they arise from the adjoint triple $\blacklozenge \dashv \star \dashv \blacksquare$. ⌟

The adjoint triple thus constructed defines a warping, and consequently a skew-monoidal structure on families; this, however, is equivalent to the rebased definition we gave above. By Theorem 9.3.1, with the canonical identity-on-objects discrete subcategory embedding $S^* \hookrightarrow \mathbb{F}[S]$, we get that the categories of $\Diamond$-algebras and $\Box$-coalgebras in $\mathbf{Fam}$ are equivalent to the category of presheaves. Thus, a presheaf can be equivalently given by its underlying family X, and a $\Diamond$-algebra structure $\Diamond X \to X$ or $\Box$-coalgebra structure $X \to \Box X$ that acts as the functorial renaming operation.

Definition 10.1.7 A *$\Diamond$-algebra* is a family X with a structure map $x : \Diamond X \to X$, satisfying:

$$x\,(\Gamma, t, \mathrm{id}) = t \qquad x\,(\Delta, (\Gamma, t, \rho), \varrho) = x\,(\Gamma, t, \varrho \circ \rho)$$

A *$\Box$-coalgebra* is a family Y with a structure map $y : Y \to \Box Y$ satisfying:

$$y\,\mathrm{id}\,t = t \qquad y\,(y\,t\,\rho)\,\varrho = y\,t\,(\varrho \circ \rho)$$

⌟

Definition 10.1.8 A *pointed $\Diamond$-algebra* is a $\Diamond$-algebra or $\Box$-coalgebra $(\mathcal{X}, x)$ in $\mathbf{Fam}_S$ or with a point $\eta : \mathcal{I} \to \mathcal{X}$ compatible with the co/algebra structure:

$$x(\Gamma, \eta\, v, \rho) = \eta\, (\rho\, v) \qquad x\, (\eta\, v)\, \rho = \eta\, (\rho\, v)$$ ⌟

As the morphisms in $\mathbb{F}[S]$ (and equivalently, sort-preserving functions between indices) correspond to the *renaming rules*, the structure maps for the co/algebras correspond to the *renaming operation*: they modify the context of a term based on a renaming rule.

From Corollary 7.2.1, we have that $\Diamond X \cong X \ominus \mathcal{I}$, and $\Box X \cong [\![\mathcal{I}, X]\!]$. The $\Diamond$-algebra and $\Box$-coalgebra structure maps are right module actions: $\Diamond X \cong X \ominus \mathcal{I} \to X$ and $X \to [\![\mathcal{I}, X]\!] \cong \Box X$.

Notation. For the sake of simplicity and clarity, we will stop distinguishing between the monoidal and closed structure when reasoning equationally, as the laws and diagrammatic reasoning reduces to very similar syntactic notation differing only in whether applications are written in curried or uncurried form. In particular, we will collectively call $\Diamond$-algebras and $\Box$-coalgebras *modules* and denote their category **Mod**, with the specific nature of the operation clear from context. Applications of the co/algebra structure for $(X, x) \in \mathbf{Mod}$ will be written $X\langle\rho\rangle t$ for either $x(\Gamma, t, \rho)$ or $x\, t\, \rho$, further abbreviated to simply $\langle\rho\rangle t$ when X is clear from the context. More generally, a parametrised morphism $f : X \ominus \mathcal{Y} \to Z$ or $f : X \to [\![\mathcal{Y}, Z]\!]$ unfolds to the same substitution rule-incorporating transformation, and will be commonly denoted $f : X \multimap\!\boxed{\mathcal{Y}}\!\to Z$. The application to $t \in X(\Gamma)$ and $\sigma : {}^{\Gamma}\mathcal{Y}_{\Delta}$ will be written $f\{\sigma\}t$, abbreviating either $f(\Gamma, t, \sigma)$ or $f\, t\, \sigma$. Note the reverse argument ordering, which will also allow us to partially apply such a map to a substitution rule and obtain a context-altering family function: $f\{\sigma\} : X(\Gamma) \to Y(\Delta)$. ⌟

10.1.4 Monoids

The skew-monoidal closed structure of families associates terms with substitution rules, but does not actually evaluate the result of the substitution. A family which supports evaluation of substitutions is nothing but a monoid object in the skew-monoidal closed category, which also carries the expected unit and associativity laws of substitution.

Definition 10.1.9 A *substitution monoid* is a sorted family $\mathcal{M}$ equipped with a *variable operation* $\eta : \mathcal{I} \to \mathcal{M}$ and a *substitution operation* $\mu : \mathcal{M} \multimap\!\boxed{\mathcal{M}}\!\to \mathcal{M}$, satisfying the expected monoid laws of Definition 5.2.1, corresponding to the unit and associativity axioms:

$$\mu\{\sigma\}(\eta\, x) = \sigma\, x \qquad \mu\{\eta\}t = t \qquad \mu\{\varsigma\}(\mu\{\sigma\}t) = \mu\{\mu\{\varsigma\} \circ \sigma\}t$$ ⌟

As given in Definition 5.2.11, an *invariant monoid* is a substitution monoid with a compatible module action: renaming is equivalent to substitution of variables for variables. While every monoid is automatically invariant, in practice we will need the module structure on the family of a syntax in order to equip it with the monoid structure. This corresponds to the well-known – if not initially obvious – fact that the renaming operation on the syntax of terms must be present before the substitution operation can be defined, as pushing substitutions under binders involves weakening, a special case of renaming.

Definition 10.1.10 An *invariant substitution monoid* is a monoid $(\mathcal{M}, \eta, \mu)$ with a module action such that the compatibility condition of Definition 5.2.11 holds; for $\rho : \Gamma \to \Delta$, we have

$$\mu\{\eta \circ \rho\} = \mathcal{M}\langle\rho\rangle$$

⌟

Finally, a Σ-monoid for a family endofunctor $\Sigma : \mathbf{Fam}_S \to \mathbf{Fam}_S$ has a monoid and Σ-algebra structure, with the compatibility condition expressing that substitution into a term involves substitution into subexpressions, which may involve pushing substitution under binders. However, to state the compatibility condition, we need to ask for a *pointed strength* on Σ – and, after several chapters of build-up, we have all the tools to tackle this challenge head-on.

10.2 Skew parametrisation

Morphisms $f : X \to Y \in \mathbf{Fam}$ are families of functions indexed by a context, expressing transformations of X-terms in Γ to Y-terms in Γ. Being families of functions indexed by contexts (and sorts, in $\mathbf{Fam}_S$), they represent context- and sort-preserving transformations of terms. If we wish to alter the underlying context of the term, we need a rule to tell us how variables in the input term transform in the new context. *Skew-parametrised maps* therefore allow for an additional (sorted) family parameter either tensored with the argument, or parametrising the output. They generalise substitution, renaming, and other generic traversal operations that involve the alteration of the term context.

For a sorted parameter family $\mathcal{Y}$, a $\mathcal{Y}$-parametrised map from X to Z is a map $f : X \ominus \mathcal{Y} \to Z$ or $g : X \to [\![\mathcal{Y}, Z]\!\rangle$ which, when unwrapped, correspond to the family of functions

$$\prod_{\Gamma,\Delta \in S^*} X(\Gamma) \to {}^{\Gamma}\mathcal{Y}_{\Delta} \to Z(\Delta)$$

Varying the parameter family results in different operations on terms: $X = Z \in \mathbf{Fam}$ and $\mathcal{P} = \mathcal{I}$ becomes nothing more than an $\mathcal{I}$-module action, i.e. a renaming operation; for a sorted family $\mathcal{M}$, an $\mathcal{M}$-parametrised map $\mathcal{M} \multimap \mathcal{M}$ is a substitution operation. Indeed, our discussion of initial-algebra semantics in the next chapter will be as generic over parameters as possible.

10.2.1 Multilinear maps

A key point to note is that parametrised maps allow us to recover the renaming-invariance discussed in Section 10.1.3 by shifting axioms from the structure of families to properties of parametrised maps. Given $f : X \ominus \mathcal{Y} \to Z$ with X a $\diamond$-algebra, even if $(\Gamma, t, \sigma \circ \rho)$ is not equal to $(\Delta, x(t, \rho), \sigma)$ componentwise, both tuples may give the same term after application of f:

$$(\Gamma, t, \sigma \circ \rho) \sim (\Delta, x(t, \rho), \sigma) \implies f(\Gamma, t, \sigma \circ \rho) = f(\Delta, x(t, \rho), \sigma) \in Z(\Delta)$$

Similarly, even if for $h \in [\![\mathcal{Y}, Z]\!](\Gamma)$ with $\mathcal{Y}, Z$ $\Box$-coalgebras we do not have the internal naturality $z\,(h_\Delta\,\sigma)\,\rho = h_\Theta(v \mapsto y\,(\sigma\,v)\,\rho)$, the equality may be satisfied if h is the value of a parametrised map $g : X \to [\![\mathcal{Y}, Z]\!]$: for all $t \in X(\Gamma)$, $z\,(g\,t\,\sigma)\,\rho = g\,t\,(v \mapsto y\,(\sigma\,v)\,\rho)$. This is precisely what is axiomatised in the notion of a *multilinear map*, discussed extensively in Section 5.2.3.

Definition 10.2.1 For modules X, Z and sorted modules $\mathcal{Y}$, a parametrised map $f : X \multimap[\mathcal{Y}] Z$ is *multilinear* if for all $\rho : \Gamma \to \Delta$, $\sigma : {}^{\Gamma}\mathcal{Y}_{\Delta}$, $\varrho : \Delta \to \Theta$, and $\varsigma : {}^{\Delta}\mathcal{Y}_{\Theta}$ the following hold:

$$f\{\mathcal{Y}\langle\varrho\rangle \circ \sigma\} = Z\langle\varrho\rangle \circ f\{\sigma\} \qquad f\{\varsigma \circ \rho\} = f\{\varsigma\} \circ X\langle\rho\rangle$$

For pointed modules $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathbf{PMod}$, the map $g : \mathcal{X} \multimap[\mathcal{Y}] \mathcal{Z}$ is *pointed multilinear* if it is multilinear, and furthermore preserves the points:

$$g\{\eta^{\mathcal{Y}}\} \circ \eta^{\mathcal{X}} = \eta^{\mathcal{Z}}$$ ⌟

From Lemma 5.2.1, we know that several maps of interest are pointed multilinear: the application $\mathcal{I} \multimap[\mathcal{Y}] \mathcal{Y}$ (implemented either as the left unitor $\lambda_{\mathcal{Y}} : \mathcal{I} \oplus \mathcal{Y} \to \mathcal{Y}$ or the applicator $j_{\mathcal{Y}} : \mathcal{I} \to [\![\mathcal{Y}, \mathcal{Y}]\!]$), the module action $\mathcal{X} \multimap[\mathcal{I}] \mathcal{X}$ for a pointed module $\mathcal{X}$, and the multiplication $\mathcal{M} \multimap[\mathcal{M}] \mathcal{M}$ for an invariant monoid $\mathcal{M}$. These three will be the most important examples of pointed multilinear maps in the initiality proof. In general, however, we have the following result, relating parametrised maps in presheaves and multilinear maps in families:

Proposition 10.2.1 *For presheaves $P, R \in \mathbf{PSh}$ and sorted presheaves $\mathcal{Q} \in \mathbf{PSh}_S$, every morphism $f : \star P \ominus \star\mathcal{Q} \to \star R \in \mathbf{Fam}$ induces a unique natural transformation $f^\sharp : P \oslash \mathcal{Q} \to Q \in \mathbf{PSh}$ if and only if f is a multilinear map. For pointed presheaves $\mathcal{P}, \mathcal{Q}, \mathcal{R} \in \mathcal{V}/\mathbf{PSh}_S$, $g : \star\mathcal{P} \oplus \star\mathcal{Q} \to \star\mathcal{R}$ induces a unique natural transformation $g^\sharp : \mathcal{P} \otimes \mathcal{Q} \to \mathcal{R}$ if and only if g is pointed multilinear.*

$$\begin{array}{ccc} \star P \ominus \star\mathcal{Q} & \xrightarrow{m^\star_{P,\mathcal{Q}}} & \star(P \oslash \mathcal{Q}) \\ & \searrow^{f} & \downarrow{\scriptstyle \star f^\sharp} \\ & & \star R \end{array} \qquad \begin{array}{ccc} \star\mathcal{P} \oplus \star\mathcal{Q} & \xrightarrow{m^\star_{\mathcal{P},\mathcal{Q}}} & \star(\mathcal{P} \otimes \mathcal{Q}) \\ & \searrow^{g} & \downarrow{\scriptstyle \star g^\sharp} \\ & & \star\mathcal{R} \end{array}$$

Proof Take a parametrised map $f : \star P \ominus \star\mathcal{Q} \to \star R$ and assume it is pointed multilinear. We define the natural transformation $f^\sharp : P \oslash \mathcal{Q} \to R$ to have the same components as f:

$$f^\sharp_\Delta(\Gamma, t \in P(\Gamma), \sigma : {}^{\Gamma}\mathcal{Q}_{\Delta}) \triangleq f(\Gamma, t, \sigma)$$

This is uniquely determined by f. The naturality condition uses the functorial action of the presheaf tensor, which is delegated to that of $\mathcal{Q}$: for $\varrho : \Delta \to \Theta$, the naturality square commutes just when $f^\sharp_\Theta(\Gamma, t, \mathcal{Q}(\varrho) \circ \sigma) = R(\varrho)(f^\sharp_\Delta(\Gamma, t, \sigma))$ which, using the canonical module structure on $\star P$, precisely coincides with one of the multilinearity axioms for f: $f\{(\star\mathcal{Q})\langle\varrho\rangle \circ \sigma\} = (\star R)\langle\varrho\rangle \circ f\{\sigma\}$. Furthermore, since $f^\sharp$ is a natural transformation out of a coend, we need to ensure that it preserves the equivalence relation: given related tuples $(\Gamma, t, \varsigma \circ \rho) \sim (\Delta, P(\rho)(t), \sigma)$, the outputs $f_\Theta(\Gamma, t, \varsigma \circ \rho)$ and $f_\Theta(\Delta, P(\rho)(t), \sigma)$ must be equal. This, however, is the second linearity property of f: $f\{\varsigma \circ \rho\} = f\{\varsigma\} \circ (\star P)\langle\rho\rangle$. For a pointed multilinear map $g : \star\mathcal{P} \oplus \star\mathcal{Q} \to \star\mathcal{R}$, the induced natural transformation of pointed presheaves $g^\sharp : \mathcal{P} \otimes \mathcal{Q} \to \mathcal{R}$ also preserves the points: for $v \in \mathcal{V}_\alpha\Gamma$, $g^\sharp_\Delta(\Gamma, \eta^{\mathcal{P}}\, v, \eta^{\mathcal{Q}}) = g_\Delta(\Gamma, \eta^{\star\mathcal{P}}\, v, \eta^{\star\mathcal{Q}}) = \eta^{\star\mathcal{R}}\, v$, with the last equality being the point-preserving axiom of the pointed multilinear map g.

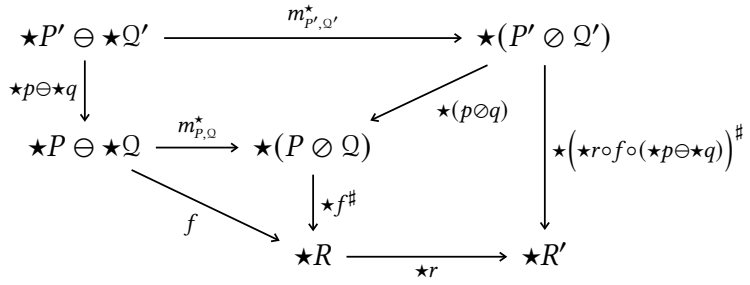
Conversely, take a natural transformation $\varphi : \mathcal{P} \otimes \mathcal{Q} \to \mathcal{R}$ and take the composite $g \triangleq \star\varphi \circ m^\star_{\mathcal{P},\mathcal{Q}} : \star\mathcal{P} \oplus \star\mathcal{Q} \to \star\mathcal{R}$, given by $g_\Delta(\Gamma, t, \sigma) \triangleq \varphi_\Delta(\Gamma, t, \sigma)$. As seen above, the pointed multilinearity axioms correspond exactly to the naturality, dinaturality, and point-preservation of φ, which are then inherited by g. □

Corollary 10.2.1 *For natural transformations $p\colon P' \to P$, $q\colon \mathcal{Q}' \to \mathcal{Q}$, and $r\colon R \to R'$, the multilinear extension satisfies the naturality condition*

$$(\star r \circ g \circ (\star p \oplus \star q))^\sharp = r \circ g^\sharp \circ (p \otimes q)$$

for all multilinear maps $g\colon \star P \oplus \star \mathcal{Q} \to \star R$.

Proof The extension of the composite decomposes into the following:



Thus, the left composite factors through both $\star(\star r \circ g \circ (\star p \oplus \star q))^\sharp$ and $\star(r \circ g^\sharp \circ (p \otimes q))$, making the unique extension $(\star r \circ g \circ (\star p \oplus \star q))^\sharp$ equal to $r \circ g^\sharp \circ (p \otimes q)$, as required. □

The uniqueness part of the theorem gives us a proof technique for equating morphisms in **PSh** by calculating in $\mathbf{Fam}_S$: $g = h\colon P \oslash \mathcal{Q} \to R$ in **PSh** if we have an equality of multilinear maps $e = f\colon \star P \ominus \star \mathcal{Q} \to \star R$ in $\mathbf{Fam}_S$ such that $e = \star g \circ m^\star_{P,\mathcal{Q}}$ and $f = \star h \circ m^\star_{P,\mathcal{Q}}$.

We have an analogous relationship between natural transformation $P \to [\mathcal{Q}, R\rangle$ and multilinear maps $\star P \to [\![\star \mathcal{Q}, \star R\rangle\!]$. Writing $\mathbf{MLin}(X; \mathcal{Y}; Z)$ for the set of multilinear maps $f\colon X \multimap \boxed{\mathcal{Y}} \to Z$, and $\mathbf{PMLin}(\mathcal{X}; \mathcal{Y}; \mathcal{Z})$ for the set of pointed multilinear maps $g\colon \mathcal{X} \multimap \boxed{\mathcal{Y}} \to \mathcal{Z}$, a classification theorem now follows.

Theorem 10.2.1

For all presheaves $P, R \in \mathbf{PSh}$, $\mathcal{Q} \in \mathbf{PSh}_S$, we have the natural isomorphisms:

$$\mathbf{PSh}(P \oslash \mathcal{Q}, R) \;\cong\; \mathbf{MLin}(\star P\,;\, \star \mathcal{Q}\,;\, \star R) \;\cong\; \mathbf{PSh}(P, [\mathcal{Q}, R\rangle)$$

For pointed presheaves $\mathcal{P}, \mathcal{Q}, \mathcal{R} \in \mathcal{V}/\mathbf{PSh}_S$, we have the isomorphisms:

$$(\mathcal{V}/\mathbf{PSh}_S)(\mathcal{P} \otimes \mathcal{Q}, \mathcal{R}) \;\cong\; \mathbf{PMLin}(\star \mathcal{P}\,;\, \star \mathcal{Q}\,;\, \star \mathcal{R}) \;\cong\; (\mathcal{V}/\mathbf{PSh}_S)(\mathcal{P}, [\mathcal{Q}, \mathcal{R}])$$

10.2.2 Synthetic monoidal structure

As we have alluded to several times, the primary challenge of adapting the presheaf model to the family setting is that the substitution monoidal structure of presheaves cannot be faithfully represented in families, or the category of modules. In particular, while $\mathbf{Fam}_S$ is skew-monoidal closed, the structure does not lift to modules: given modules $X, \mathcal{Y}$, the object $X \ominus \mathcal{Y}$ cannot be given a module structure in such a way that the action lifts to $\widehat{\ominus}\colon \mathbf{Mod} \times \mathbf{Mod}_S \to \mathbf{Mod}$. Following the functorial action of $P \oslash \mathcal{Q}$ in **PSh**, one guess is to delegate the renaming to $\mathcal{Y}$:

$$(X \ominus \mathcal{Y})\langle\rho\rangle\big(\Gamma, t \in X(\Gamma), \sigma\colon {}^{\Gamma}\mathcal{Y}_{\Delta}\big) \triangleq (\Gamma, t, (\mathcal{Y}\langle\rho\rangle \circ \sigma)\colon {}^{\Gamma}\mathcal{Y}_{\Theta})$$

This gives a functor $\mathbf{Fam} \times \mathbf{Mod}_S \to \mathbf{Mod}$ which in turn induces a functor $\mathbf{Mod} \times \mathbf{Mod}_S \to \mathbf{Mod}$ which simply forgets the module structure of the first argument and proceeds as above. This, however, does not make $\mathbf{Mod}$ a $\mathbf{Mod}_S$-modular category, and $\mathbf{Mod}_S$ a skew-monoidal category, for attempting to define the relevant structural transformations will be unsuccessful. For example, take the unitor $\rho_X \colon X \to X \ominus \mathcal{I}$ over a module X, with the usual definition mapping $t \in X(\Gamma)$ to $(\Gamma, t, \mathrm{id}) \in X \ominus \mathcal{I}$; to be a unitor in $\mathbf{Mod}$, it must itself be a module morphism, satisfying, for all $\rho \colon \Gamma \to \Delta$,

$$\begin{array}{ccc} X(\Gamma) & \xrightarrow{(\rho_X)_\Gamma} & (X \ominus \mathcal{I})(\Gamma) \\ {\scriptstyle X\langle\rho\rangle}\downarrow & & \downarrow{\scriptstyle (X\ominus\mathcal{I})\langle\rho\rangle} \\ X(\Delta) & \xrightarrow[(\rho_X)_\Delta]{} & (X \ominus \mathcal{I})(\Delta) \end{array}$$

This, however, reduces to the equality of

$$(\Delta, X\langle\rho\rangle t, \mathrm{id}_\Delta) \quad \text{and} \quad (\Gamma, t, \rho)$$

which, without quotienting at our disposal, cannot be established. Similarly, given pointed modules $\mathcal{X}, \mathcal{Y} \in \mathcal{I}/\mathbf{Mod}$, the point for the tensor $\mathcal{X} \oplus \mathcal{Y}$ that maps $v \in \mathcal{I}_\alpha\Gamma$ to $(\Gamma, \eta^{\mathcal{X}}\, v, \eta^{\mathcal{Y}})$ is not a module homomorphism, as the condition is $(\Delta, \eta^{\mathcal{X}}(\rho\, v), \eta^{\mathcal{Y}}) = (\Gamma, \eta^{\mathcal{X}}\, v, \mathcal{Y}\langle\rho\rangle \circ \eta^{\mathcal{Y}})$.

These discrepancies get in the way of translating between presheaves and modules as seamlessly as we would like: while every presheaf is equivalent to a module, Proposition 10.1.1 states that the underlying family of the presheaf tensor is not merely the family tensor of underlying families, so the presheaf tensor product cannot be decomposed to a family tensor product with a module structure. The family tensor – being an unquotiented dependent sum – will always be weaker than the presheaf tensor product, and therefore does not carry a functorial module structure. As a result, any part of the presheaf model that depends on presheaves' (strong) monoidal structure will not have a direct analogue in the familial model – pointed strength being the most important example.

This is precisely the motivating situation for our development of *synthetic monoidal categories* in Chapter 6, and we happily reap the rewards now. While $\mathcal{I}/\mathbf{Mod}_S$ is not skew-monoidal, it is synthetic monoidal over $\mathbf{Fam}_S$, with forgetful functor embedding $\underline{\mathcal{X}} = \underline{(\mathcal{X}, x, \eta)} \triangleq \mathcal{X}$, unit presheaf $\mathcal{J} \triangleq (\mathcal{I}, \lambda_{\mathcal{J}} \colon \mathcal{I} \multimap \mathcal{I} \to \mathcal{I}, \mathrm{id} \colon \mathcal{I} \to \mathcal{I})$ for which $\underline{\mathcal{J}} = \mathcal{I}$, and the profunctor of multiplicative maps $\mathscr{M}[\mathcal{X}, \mathcal{Y}; \mathcal{Z}]$ consisting of pointed multilinear maps $\mathcal{X} \multimap \mathcal{Y} \to \mathcal{Z}$. The category of unsorted modules $\mathbf{Mod}$ (with embedding $\mathbf{Mod} \to \mathbf{Fam}$) is then an synthetic $\mathcal{I}/\mathbf{Mod}_{S|\mathbf{Fam}_S}$-modular category, with set of synthetic actions $\mathscr{S}[X, \mathcal{Y}; Z]$ consisting of parametrised maps $X \multimap \mathcal{Y} \to Z$. We may also take $\mathcal{V}/\mathbf{PSh}_S$ to be a representative monoidal category over $\mathbf{PSh}_S$ and $\mathbf{PSh}$ a $\mathcal{V}/\mathbf{PSh}_S$-modular category, with the expected forgetful functor embeddings.

Relating these categories is easy, thanks to Theorem 10.2.1. The forgetful functor $\star \colon \mathbf{PSh}_S \to \mathbf{Fam}_S$ is monoidal, so the pointed comparison functor $\boldsymbol{K} \colon \mathcal{V}/\mathbf{PSh}_S \to \mathcal{I}/\mathbf{Mod}_S$ from the monoidal category of pointed presheaves to the synthetic monoidal category of pointed modules is synthetic monoidal (Proposition 6.1.1): it maps natural transformations $\mathcal{V} \to \mathcal{R}$ to point-preserving module homomorphisms $\star\mathcal{V} = \mathcal{I} \to \star\mathcal{R}$, and natural transformations $\varphi \colon \mathcal{P} \otimes \mathcal{Q} \to \mathcal{R}$ to pointed multilinear maps $\star\varphi \circ m^{\star}_{\mathcal{P},\mathcal{Q}} \colon \star\mathcal{P} \oplus \star\mathcal{Q} \to \star\mathcal{R}$. Conversely, the inverse functor $L \colon \mathbf{Mod}_S \to \mathbf{PSh}_S$ from modules to presheaves maps pointed multilinear

maps $f\colon \mathcal{X} \oplus \mathcal{Y} \to \mathcal{Z}$ to natural transformations $f^\sharp\colon \boldsymbol{L}\mathcal{X} \otimes \boldsymbol{L}\mathcal{Y} \to \boldsymbol{L}\mathcal{Z}$. Thus, the equivalence $\mathcal{V}/\mathbf{PSh}_S \simeq \mathcal{I}/\mathbf{Mod}_S$ is a *synthetic monoidal equivalence*, as is $\mathbf{PSh}_S \cong \mathbf{Mod}_S$ and $\mathbf{PSh} \cong \mathbf{Mod}$.

10.2.3 Strength and algebraic monoids

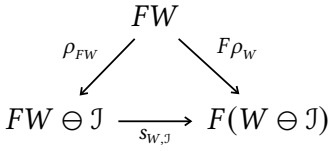
Let us now return to the definition of algebraic monoids, which was deferred until we have a clear characterisation of pointed strength for endofunctors. A functor $F\colon \mathbf{Fam} \to \mathbf{Fam}$ may well be strong over $\mathbf{Fam}$ or $\mathbf{Fam}_S$, but in practice, we will encounter functors that manipulate the variable context and can only be equipped with a strength over pointed modules. While $\mathcal{I}/\mathbf{Mod}_S$ is not skew-monoidal, $\mathbf{Fam}$ is a synthetic $\mathcal{I}/\mathbf{Mod}_{S|\mathbf{Fam}_S}$-modular category, with the action of $\mathcal{I}/\mathbf{Mod}_S$ onto $\mathbf{Fam}$ simply given by the combination of the forgetful functor $\mathcal{I}/\mathbf{Mod}_S \to \mathbf{Fam}_S$ and the action $\ominus\colon \mathbf{Fam} \times \mathbf{Fam}_S \to \mathbf{Fam}$. An endofunctor over such a modular category consists of a functor $F\colon \mathbf{Fam} \to \mathbf{Fam}$ and a strength transformation

$$s^F[-]\colon \mathbf{Fam}(X \ominus \mathcal{Y}, Z) \to \mathbf{Fam}(FX \ominus \mathcal{Y}; FZ)$$

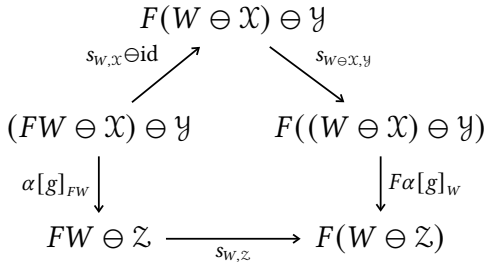
satisfying unit and associativity axioms, the latter parametrised by a pointed multilinear map (see Diagram ($s\alpha\oslash L$)). As the artificial module structure is representable, we equivalently assume a strength

$$s^F_{W,\mathcal{X}}\colon FW \ominus \mathcal{X} \to F(W \ominus \mathcal{X})\colon \mathbf{Fam} \times \mathcal{I}/\mathbf{Mod}_S \to \mathbf{Fam}$$

satisfying parametric unit and associativity laws: for every pointed multilinear map $g\colon \mathcal{X} \multimap \boxed{\mathcal{Y}} \to \mathcal{Z}$, we have



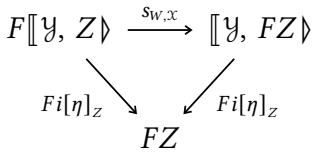
$$F(\rho[\eta])(t) = s(t, \eta)$$



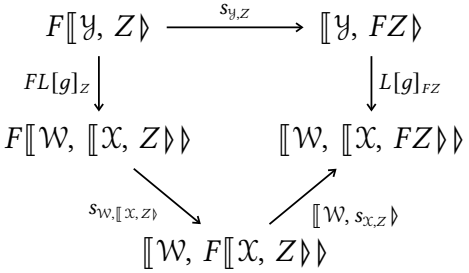
$$F(\alpha[g])(s\,(s\,(t, \sigma), \varsigma)) = s\,(t, g[\varsigma] \circ \sigma)$$

where $\alpha[g]$ is the composite $(W \ominus \mathcal{X}) \ominus \mathcal{Y} \xrightarrow{\alpha_{W,\mathcal{X},\mathcal{Y}}} W \ominus (\mathcal{X} \oplus \mathcal{Y}) \xrightarrow{W \ominus g} W \ominus \mathcal{Z}$. In the closed setting, the strength is the transformation satisfying the laws below:

$$s^F_{\mathcal{Y},Z}\colon F[\![\mathcal{Y}, Z]\!] \to [\![\mathcal{Y}, FZ]\!]\colon (\mathcal{I}/\mathbf{Mod}_S)^{\mathrm{op}} \times \mathbf{Fam} \to \mathbf{Fam}$$



$$F(i[\eta])(t) = s\;t\;\eta$$



$$F(L[g])(s\,(s\,t\,\sigma)\,\varsigma) = s\,t\,(g[\varsigma] \circ \sigma)$$

As will be made explicit in Section 12.1.3, this is indeed the appropriate axiomatisation of pointed strength in that it generalises the right interaction axioms for substitution rule lifting (see e.g. Section 2.1.3) without relying on quotienting or other hard-to-formalise concepts. The approach via synthetic monoidal categories put this generalisation on formal foundations, demonstrating that the presence of a pointed multilinear map in the associativity axiom is motivated by the lack of the required monoidal structure on pointed modules.

With the appropriately general notion of strength at hand, we can define algebraic monoids – the models of second-order abstract syntax.

Definition 10.2.2 Given an endofunctor $\Sigma\colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ of $\mathcal{I}/\mathbf{Mod}_S$-modular categories, an *Σ-monoid* is a substitution monoid $(\mathcal{M}, \eta, \mu)$ with an algebra structure $a\colon \Sigma\mathcal{M} \to \mathcal{M}$ such that either of the following equivalent diagrams commute:

$$\begin{array}{ccccc} \Sigma\mathcal{M} \oplus \mathcal{M} & \xrightarrow{s_{\mathcal{M},\mathcal{M}}} & \Sigma(\mathcal{M} \oplus \mathcal{M}) & \xrightarrow{\Sigma\mu} & \Sigma\mathcal{M} \\ {\scriptstyle a\oplus\mathcal{M}}\downarrow & & & & \downarrow{\scriptstyle a} \\ \mathcal{M} \oplus \mathcal{M} & & \xrightarrow[\mu]{} & & \mathcal{M} \end{array} \qquad \begin{array}{ccccc} \Sigma\mathcal{M} & \xrightarrow{\Sigma\mu} & \Sigma[\![\mathcal{M}, \mathcal{M}]\!] & \xrightarrow{s_{\mathcal{M},\mathcal{M}}} & [\![\mathcal{M}, \Sigma\mathcal{M}]\!] \\ {\scriptstyle a}\downarrow & & & & \downarrow{\scriptstyle [\![\mathcal{M},a]\!]} \\ \mathcal{M} & & \xrightarrow[\mu]{} & & [\![\mathcal{M}, \mathcal{M}]\!] \end{array}$$

$$\mu(a\,t, \sigma) = a\,(\Sigma\,\mu\,(s\,(t, \sigma))) \qquad \mu\,(a\,t)\,\sigma = a\,(s\,(\Sigma\mu\,t)\,\sigma)$$

Back in Theorem 7.2.1 we proved an equivalence theorem between algebraic monoids in monoidal and warped categories. Most of the preconditions of the theorem – namely, the warping and the synthetic monoidal relationships – are already satisfied, so a straightforward instantiation establishes the fundamental correctness theorem of the familial model.

Theorem 10.2.2

Given a synthetic $\mathcal{I}/\mathbf{Mod}_S$-strong endofunctor $\Sigma\colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ and a $\mathcal{V}/\mathbf{PSh}_S$-strong endofunctor $\Omega\colon \mathbf{PSh}_S \to \mathbf{PSh}_S$ such that both functors are unital, $\star\Omega \cong \Sigma\star$ and for all $\mathcal{P} \in \mathbf{PSh}_S$, $\mathcal{Q} \in \mathcal{V}/\mathbf{PSh}_S$:

$$\begin{array}{ccccc} \Sigma(\star\mathcal{P}) \oplus (\star\mathcal{Q}) & \xrightarrow{s^{\Sigma}_{\star\mathcal{P},\star\mathcal{Q}}} & \Sigma(\star\mathcal{P} \oplus \star\mathcal{Q}) & \xrightarrow{\Sigma m^{\star}_{\mathcal{P},\mathcal{Q}}} & \Sigma\star(\mathcal{P} \otimes \mathcal{Q}) \\ \downarrow & & & & \downarrow \\ \star(\Omega\mathcal{P}) \oplus (\star\mathcal{Q}) & & \xrightarrow[m^{\star}_{\Omega\mathcal{P},\mathcal{Q}}]{} & & \xrightarrow[\star s^{\Omega}_{\mathcal{P},\mathcal{Q}}]{} \star\Omega(\mathcal{P} \otimes \mathcal{Q}) \end{array}$$

the categories of Σ-monoids in families and Ω-monoids in presheaves are equivalent.

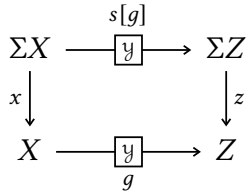
Proof The assumptions on the endofunctors exhibit Ω as the strong lifting of Σ along $\star$ in the category of synthetic modular categories. More precisely, the diagram can be reinterpreted by considering the objects to be actions of pointed modules on sorted families, where the comparison functor $\boldsymbol{K}\colon \mathcal{V}/\mathbf{PSh}_S \to \mathcal{I}/\mathbf{Mod}_S$ is synthetic monoidal and $\star\colon \mathbf{PSh}_S \to \mathbf{Fam}_S$ is $\boldsymbol{K}$-relative strong, with operator $s^{\star,\boldsymbol{K}}_{\mathcal{P},\boldsymbol{\mathcal{Q}}}\colon \star\mathcal{P} \oplus \boldsymbol{K}\boldsymbol{\mathcal{Q}} \to \star(\mathcal{P} \otimes \boldsymbol{\mathcal{Q}})$. Thus, $(\boldsymbol{K}, \star)\colon (\mathcal{V}/\mathbf{PSh}_S, \mathbf{PSh}_S) \to (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}_S)$ is an elevator from $(\mathrm{Id}, \Sigma)\colon (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}_S) \to (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}_S)$ to $(\mathrm{Id}, \Omega)\colon (\mathcal{V}/\mathbf{PSh}_S, \mathbf{PSh}_S) \to (\mathcal{V}/\mathbf{PSh}_S, \mathbf{PSh}_S)$, instantiating the last precondition of Theorem 7.2.1. □

Remark. We can characterise Σ-monoids currying-agnostically, like we did with modules and parametrised maps. The strength operator $s[-]$ for a synthetic strong Σ factors through the

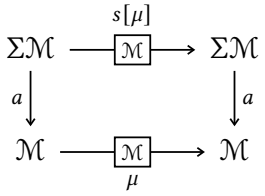
strength transformation: for a multilinear map $f : X \ominus \mathcal{Y} \to Z$, the composite

$$s[f] : \Sigma X \ominus \mathcal{Y} \xrightarrow{s_{X,\mathcal{Y}}} \Sigma(X \ominus \mathcal{Y}) \xrightarrow{\Sigma f} \Sigma Z$$

is itself a multilinear map by Proposition 6.2.3, and similarly with closed strengths. Thus, the strength operator can in fact be seen as a *parametrised functorial action* from $f : X \multimap \boxed{\mathcal{Y}} \to Z$ to $s[f] : \Sigma X \multimap \boxed{\mathcal{Y}} \to \Sigma Z$. This is a valuable perspective, as the strength is generally needed when we would like to evaluate a multilinear map, but some sort of algebraic structure is "in the way". Case in point is the compatibility of algebraic and substitution structure: given a constructor (i.e. output of an Σ-algebra structure map) and a substitution rule, the pointed strength will allow us to perform syntactic substitution by lifting the rule over and inside the constructors. With the parametrised map notation, algebraic linearity for a multilinear map looks strikingly simple: a multilinear map $f : X \multimap \boxed{\mathcal{Y}} \to Z$ is Σ-linear and a monoid $\mathcal{M}$ is an Σ-monoid if:



$$g\{\sigma\} \circ x = z \circ s[g]\{\sigma\}$$



$$\mu\{\sigma\} \circ a = a \circ s[\mu]\{\sigma\}$$

10.3 Convolutional structure

Unlike presheaves, the category of families has admits a Day convolutional structure that is different from its cartesian structure. The resulting operations play a crucial role both in the encoding of variable binding as well as second-order features.

10.3.1 Context extension

As families are formally presheaves over the discrete monoidal category of contexts, they possess the expected bicartesian structure of presheaf categories, with products and coproducts taken pointwise. Since the base category is discrete, the presheaf exponential also simplifies to the pointwise set exponential.

$$(X \times Y)(\Gamma) \triangleq X(\Gamma) \times Y(\Gamma) \qquad (X + Y)(\Gamma) \triangleq X(\Gamma) + Y(\Gamma) \qquad (X \Rightarrow Y)(\Gamma) \triangleq X(\Gamma) \Rightarrow Y(\Gamma)$$

The last definition showcases another limitation of the category of families: while it is simple, it is also not as "interesting" as presheaves, exponentials of which are far richer than simple pointwise transformations. In fact, one of the most important identities of the presheaf model relates exponentiation to context extension, a fundamental building block in the specification of second-order signatures.

Definition 10.3.1 The *context extension endofunctor* $\delta_\Theta : \mathbf{PSh} \to \mathbf{PSh}$ is defined as

$$\delta_\Theta(P)(\Gamma) \triangleq P(\Theta + \Gamma) \qquad \delta_\Theta(\rho) \triangleq P(\Theta + \rho)$$

Variants on sorted presheaves and un/sorted families are defined analogously. ⌟

Proposition 10.3.1 *For a presheaf $P \in \mathbf{PSh}$, we have the following isomorphism:*

$$\text{ℱ}(\Theta) \supset P \;\cong\; \delta_\Theta P$$

Proof We calculate at a context Γ as follows, using the definition of presheaf exponentials (Definition 8.2.1), and the Yoneda lemma (Eq. ($\text{ℱ}\cong$)):

$$(\text{ℱ}(\Theta) \supset P)(\Gamma) \triangleq \mathbf{PSh}\big(\text{ℱ}(\Gamma) \times \text{ℱ}(\Theta), P\big) \cong \mathbf{PSh}\big(\text{ℱ}(\Theta + \Gamma), P\big) \cong P(\Theta + \Gamma) \triangleq \delta_\Theta P(\Gamma) \qquad \square$$

The implication is that the bicartesian closed structure of presheaves is sufficient for modelling second-order abstract syntax, but this is not the case for families: neither co/products nor exponentials modify the variable context so are not suitable for the interpretation of variable-binding operators. The task, then, is to derive the context extension endofunctor from a different categorical structure on families.

The key is to notice that the discrete category of contexts is monoidal under context concatenation, so **Fam** is equipped with a different monoidal closed structure given by *Day convolution* and the *Day internal hom* as defined in Section 8.2.2. The abstract definitions simplify to the following in the discrete presheaf category:

Definition 10.3.2 The Day tensors in **Fam** are given by

$$(X \otimes Y)(\Theta) \triangleq \sum_{\Gamma+\Delta=\Theta} X(\Gamma) \times Y(\Delta) \qquad (X \oslash Y)(\Theta) \triangleq \sum_{\Gamma+\Delta=\Theta} X(\Delta) \times Y(\Gamma)$$

The Day internal homs in **Fam** are

$$(X \multimap Y)(\Gamma) \triangleq \prod_{\Delta \in S^*} X(\Delta) \to Y(\Gamma + \Delta) \qquad (X \circ\!\!- Y)(\Gamma) \triangleq \prod_{\Delta \in S^*} X(\Delta) \to Y(\Delta + \Gamma)$$

⌟

As per Theorem 8.2.2, both structures make **Fam** monoidal closed, with unit $E = \text{ℱ}_{S^*}[]$ that checks if its argument is the empty list. The structures of course also apply to sorted families, where an analogue of the variable/index presheaf is the following:

Definition 10.3.3 The sorted family of *names* $\mathcal{N} \in \mathbf{Fam}_S$ is defined as $\mathcal{N}_\alpha \triangleq \text{ℱ}_{S^*}[\alpha]$. $\mathcal{N}_\alpha\Gamma$ checks if Γ is equal to the singleton list $[\alpha]$. ⌟

Note that families are presheaves over the discrete monoidal category of contexts, so the associators and unitors in S^* are strict equalities: $(\Gamma + \Delta) + \Theta = \Gamma + (\Delta + \Theta)$ and $[] + \Delta = \Delta$ and $\Gamma + [] = \Gamma$. Consequently, the functorial action of a family Y is simply an equality rewriting operation: for $t \in X((\Gamma + \Delta) + \Theta)$, $X(\alpha_{\Gamma,\Delta,\Theta})t = t \in X(\Gamma + (\Delta + \Theta))$. Though the action itself is equality on terms, it will often be useful to be explicit about the equality rewriting: thus, for example, the internal currying map $c_Z^{\otimes,X,Y} : (X \otimes Y) \multimap Z \to (X \multimap (Y \multimap Z))$ is defined as

$$c_Z^{\otimes,X,Y}\,(l \in ((X \otimes Y) \multimap Z)(\Gamma))\,(x \in X(\Delta))\,(y \in Y(\Theta)) \triangleq Z(\alpha^{-1}_{\Gamma,\Delta,\Theta})(l\,(x, y))$$

since $l\,(x, y) \in Z(\Gamma + (\Delta + \Theta))$, but $(X \multimap (Y \multimap Z))_{\Delta,\Theta} = X(\Delta) \to (Y(\Theta) \to Z((\Gamma + \Delta) + \Theta))$. In fact, the opposing convolution tensors and homs come with the following currying

isomorphisms, the bottom two exhibiting **Fam** as a powered category (see Section 4.1):

$$(X \otimes Y) \multimap Z \cong X \multimap (Y \multimap Z) \qquad (X \otimes Y) \circ\!\!- Z \cong X \otimes (Y \circ\!\!- Z)$$
$$(X \otimes Y) \multimap Z \cong Y \multimap (X \multimap Z) \qquad (X \otimes Y) \circ\!\!- Z \cong Y \otimes (X \circ\!\!- Z)$$

If $X \in \mathbf{Fam}$ also has a module structure, equality proofs in S^* give rise to renaming rules in $\mathbb{F}[S]$ by the functorial action of $\mathcal{I}_\alpha$, and they are equal to renaming rules constructed via the cocartesian structure of $\mathbb{F}[S]$. For example, $\mathcal{I}_\alpha(\alpha^{S^*}_{\Gamma,\Delta,\Theta}) : \mathcal{I}_\alpha((\Gamma + \Delta) + \Theta) \to \mathcal{I}_\alpha(\Gamma + (\Delta + \Theta))$ is equal to the associator $\alpha^{\mathbb{F}[S]}_{\Gamma,\Delta,\Theta} : (\Gamma + \Delta) + \Theta \to \Gamma + (\Delta + \Theta)$ in $\mathbb{F}[S]$. An interesting side-effect of this is that the quotienting condition $(\Delta, X\langle\rho\rangle\, t, \sigma)\ (\Gamma, t, \sigma \circ \rho)$ *does* hold in the case that $\Gamma = \Delta \in S^*$, which makes both $\rho : \Gamma \to \Delta$ and $X\langle\rho\rangle : X(\Gamma) \to X(\Delta)$ the identity morphisms. For example, even without application of a multilinear map, we have

$$\big(\Gamma + (\Delta + \Theta), X\langle\alpha_{\Gamma,\Delta,\Theta}\rangle t, \sigma\big) = \big((\Gamma + \Delta) + \Theta, t, \sigma \circ \alpha_{\Gamma,\Delta,\Theta}\big)$$

for all $t \in X((\Gamma + \Delta) + \Theta)$ and $\sigma \in {}^{\Gamma+(\Delta+\Theta)}\mathcal{Y}_{\Xi}$.

With the definition of the Day homs already incorporating explicit context extension, the following analogue of Proposition 10.3.1 is not surprising:

Proposition 10.3.2 *For a family $X \in \mathbf{Fam}$, we have the following isomorphism:*

$$\mathcal{Y}(\Theta) \circ\!\!- X \;\cong\; \delta_\Theta X$$

Proof The Yoneda embedding in the category of families is the equality relation, so $\mathcal{Y}(\Theta)(\Delta)$ is inhabited only if $\Theta = \Delta$.

$$\begin{aligned}(\mathcal{Y}\Theta \circ\!\!- X)(\Gamma) &\triangleq \prod_{\Delta \in S^*} \mathcal{Y}(\Theta)(\Delta) \to X(\Delta + \Gamma) \\ &\cong \prod_{\Delta \in S^*} (\Theta = \Delta) \to X(\Delta + \Gamma) \cong X(\Theta + \Gamma) \triangleq \delta_\Theta X(\Gamma)\end{aligned}$$

□

In fact, we can relate the Day monoidal closed structure of families and cartesian closed structure of presheaves precisely, serving as a reassurance that the natural translation of the latter into the familial model requires structure beyond pointwise products and exponentials.

Proposition 10.3.3 *The presheaf of variables, the family of names and family of indices are related by the following:*

$$\blacklozenge\mathcal{N} \cong \mathcal{V} \qquad \Diamond\mathcal{N} \cong \mathcal{I}$$

Proof We calculate as follows:

$$\blacklozenge\mathcal{N}_\alpha(\Delta) = \sum_{\Gamma \in S^*} \mathbb{F}[S](\Gamma, \Delta) \times \mathcal{N}_\alpha(\Gamma) = \sum_{\Gamma \in S^*} \mathbb{F}[S](\Gamma, \Delta) \times ([\alpha] = \Gamma) \cong \mathbb{F}[S]([\alpha], \Delta) \cong \mathcal{V}_\alpha(\Delta)$$

Applying $\star$ to both sides gives the second isomorphism, as $\mathcal{I} \triangleq \star\mathcal{V}$. □

Proposition 10.3.4 *We have the following isomorphisms and transformations relating cartesian closed presheaves with Day monoidal closed families:*

$$\begin{aligned} \star P \circledcirc \star Q &\to \star(P \times Q) & \blacklozenge(X \circledcirc Y) &\cong \blacklozenge X \times \blacklozenge Y \\ \blacklozenge(X \multimap Y) &\to (\blacklozenge X) \supset (\blacklozenge Y) & \star(\blacklozenge X \supset Q) &\cong X \multimap (\star Q) \\ \star(P \supset Q) &\to \star P \Rightarrow \star Q & \blacksquare(\star P \Rightarrow Y) &\cong P \supset (\blacksquare Y) \end{aligned}$$

Proof The isomorphism $\blacklozenge(X \circledcirc Y) \cong \blacklozenge X \times \blacklozenge Y$ is an instance of the universal property of Day convolution from Theorem 8.2.3: taking the monoidal category to be cartesian presheaves, we have that the free presheaf functor $\blacklozenge\colon$ **Fam** $\to$ **PSh** preserves tensors. Instantiating Proposition 3.2.1 with $J_2 = \blacklozenge$ and the "lifting" of $\times$ to $\circledcirc$ along $\blacklozenge$, we also get a coelevator

$$\star P \circledcirc \star Q \to \star(P \times Q)$$

The isomorphism $\blacklozenge X \times \blacklozenge Y \to \blacklozenge(X \circledcirc Y)$ above is used for $\blacklozenge(X \multimap Y) \to (\blacklozenge X) \supset (\blacklozenge Y)$:

$$\blacklozenge(X \multimap Y) \times \blacklozenge X \to \blacklozenge((X \multimap Y) \times X) \to \blacklozenge Y$$

and the isomorphism itself Yoneda transposes to $\star(\blacklozenge X \supset Q) \cong X \multimap (\star Q)$. Similarly, the isomorphism $\star(P \times Q) \cong \star P \times \star Q$ gives us the lax strength $\star(P \supset Q) \to \star P \Rightarrow \star Q$, and the Yoneda transpose $\blacksquare(\star P \Rightarrow Y) \cong P \supset (\blacksquare Y)$. □

Remark. Since the cartesian product is commutative, the above isomorphisms also apply to the reversed Day tensor and internal hom. ⌟

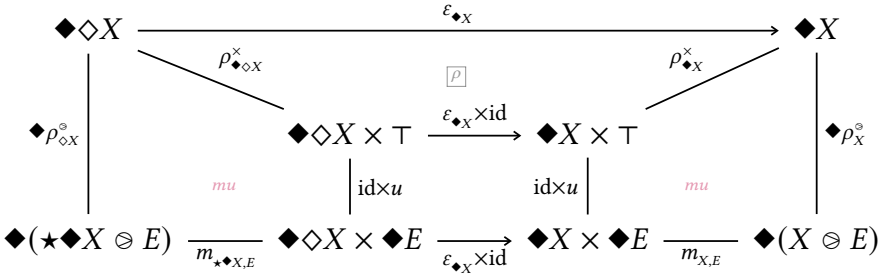
Proposition 10.3.5 *$\Diamond$ and $\square$ are module functors with respect to the Day convolution and hom:*

$$\Diamond X \circledcirc Y \to \Diamond(X \circledcirc Y) \qquad \Diamond(X \multimap Y) \to (X \multimap \Diamond Y) \qquad (X \multimap \square Y) \to \square(X \multimap Y)$$

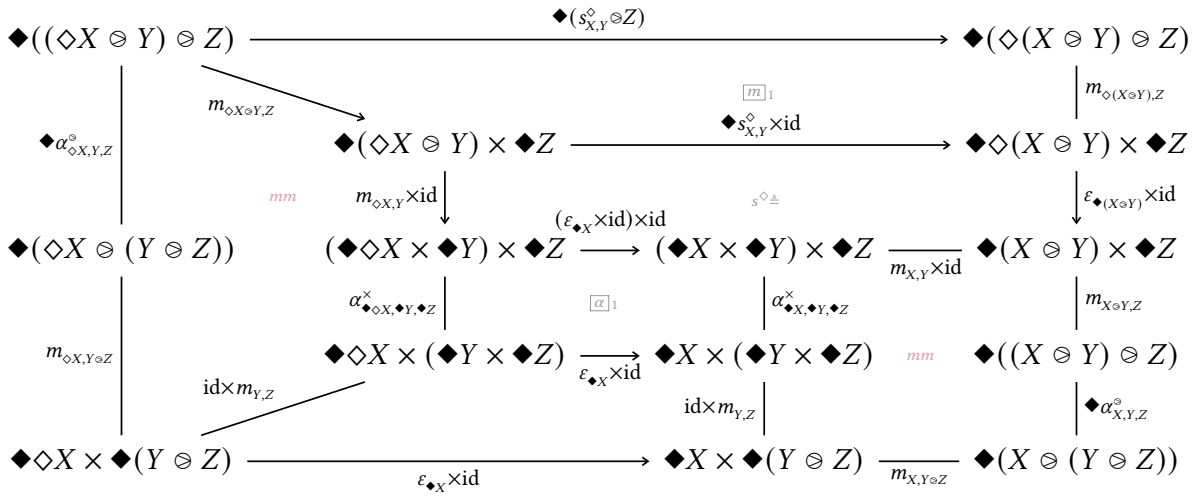
Proof The strength $\Diamond X \circledcirc Y \to \Diamond(X \circledcirc Y)$, expanded $\star\blacklozenge X \circledcirc Y \to \star\blacklozenge(X \circledcirc Y)$, is the transpose of the following, using the strong monoidality of $\blacklozenge$:

$$\blacklozenge(\star\blacklozenge X \circledcirc Y) \cong \blacklozenge\star\blacklozenge X \times \blacklozenge Y \xrightarrow{\varepsilon_X \times \mathrm{id}} \blacklozenge X \times \blacklozenge Y \cong \blacklozenge(X \circledcirc Y)$$

The transpose of the strength unit law is:



The transpose of the associativity law is as follows:



The monoidal strength $\Diamond X \circledcirc Y \to \Diamond(X \circledcirc Y)$ induces $\Diamond(X \multimap Y) \to (X \multimap \Diamond Y)$ by the usual calculation given in Theorem 5.1.1. Finally, $(X \multimap \Box Y) \to \Box(X \multimap Y)$ is the Yoneda transpose of $\Diamond X \circledcirc Y \to \Diamond(X \circledcirc Y)$. □

Explicitly, the strength transformations behave as follows:

$$s^{\Diamond}_{X,Y}\big(\Delta + \Theta, (s \in X(\Gamma), \rho \colon \Gamma \to \Delta), t \in Y(\Theta)\big) \triangleq \big(\Delta + \Theta, (s, t), \rho + \Theta \colon (\Gamma + \Theta) \to (\Delta + \Theta)\big)$$
$$s^{\Diamond}_{X,Y}\big(l \in (X \multimap Y)(\Gamma), \rho \colon \Gamma \to \Delta\big) \triangleq \Theta, x \in X(\Theta) \mapsto (l\, x \in Y(\Gamma + \Theta), \rho + \Theta \colon (\Gamma + \Theta) \to (\Delta + \Theta))$$
$$s^{\Box}_{X,Y}(l \colon (X \multimap \Box Y)(\Gamma)) \triangleq \rho \colon \Gamma \to \Delta, \Theta, x \in X(\Theta) \mapsto l\, x\, (\rho + \Theta \colon (\Gamma + \Theta) \to (\Delta + \Theta))$$

Combining the isomorphisms with families of names, indices, and presheaf of variables, we have ways of expressing context extension of presheaves and families in a multitude of ways.

Corollary 10.3.1 *We have the following derived isomorphisms:*

$$\star(\mathcal{V}_\tau \supset P) \cong \mathcal{N}_\tau \multimap \star P \qquad \mathcal{V}_\tau \supset \blacksquare X \cong \blacksquare(\mathcal{I}_\tau \Rightarrow X) \qquad \Box(\mathcal{I}_\tau \Rightarrow X) \cong \mathcal{N}_\tau \multimap \Box X$$

Proof The first two follow from isomorphisms in the previous two propositions:

$$\star(\mathcal{V}_\tau \supset P) \cong \star(\blacklozenge \mathcal{N}_\tau \supset P) \cong \mathcal{N}_\tau \multimap \star P$$
$$\mathcal{V}_\tau \supset \blacksquare X \cong \blacksquare(\star \mathcal{V}_\tau \Rightarrow X) \cong \blacksquare(\mathcal{I}_\tau \Rightarrow X)$$

The third is a combination of the two:

$$\Box(\mathcal{I}_\tau \Rightarrow X) = \star\blacksquare(\mathcal{I}_\tau \Rightarrow X) \cong \star(\mathcal{V}_\tau \supset \blacksquare X) \cong \mathcal{N}_\tau \multimap \star\blacksquare X = \mathcal{N}_\tau \multimap \Box X$$ □

As presheaves and modules are equivalent, presheaf exponentials out of free presheaves $\blacklozenge X \supset P$ are equivalently described by a $\Diamond$-algebra structure

$$\Diamond(X \multimap \star P) \xrightarrow{s^{\Diamond}_{X,\star P}} (X \multimap \Diamond \star P) \xrightarrow{X \multimap \star \varepsilon_P} X \multimap \star P$$

Thus, $\delta_\tau(P) \cong \blacklozenge \mathcal{N}_\tau \supset P$ is equivalently described by a $\Diamond$-algebra structure on $\delta_\tau(\star P)$, induced

by the distributive law $\Diamond(X \multimap (-)) \to X \multimap \Diamond(-)$.

10.3.2 Pointed strength

By partial sorting the Day convolution operation, we obtain a right action $\mathbf{Fam} \times \mathbf{Fam}_S \to \mathbf{Fam}$ which is both left and right closed with the following operations:

Definition 10.3.4 The left and right Day homs are given by

$$(-) \multimap\!\bullet\ (=) \colon \mathbf{Fam}^{\mathrm{op}} \times \mathbf{Fam}_S \to \mathbf{Fam}_S \qquad \langle -, = \rangle \colon \mathbf{Fam}_S^{\mathrm{op}} \times \mathbf{Fam}_S \to \mathbf{Fam}$$
$$(X \multimap\!\bullet\ \mathcal{Y})_\tau \triangleq X \multimap \mathcal{Y}_\tau \qquad \langle \mathcal{X}, \mathcal{Y} \rangle \triangleq \prod_{\tau \in S} \mathcal{X}_\tau \multimap \mathcal{Y}_\tau$$

⌟

The rest of Section 8.2.2 applies with these definitions, as well as the clone constructions of Chapter 4 which we will use later.

Notation. Day homs will often be written using superscript notation, with sorting kept implicit:

$$W \multimap X \sim X^W \qquad W \multimap\!\bullet\ \mathcal{X} \sim \mathcal{X}^W$$

We maintain $\langle \mathcal{X}, \mathcal{Y} \rangle$ for the right Day hom, and let $\mathcal{Y}^{\mathcal{X}} \in \mathbf{Fam}_S$ refer to the fully sorted hom. ⌟

Definition 10.3.5 The *diagonal transformation* is a family of maps $\kappa_X^W : X \to X^W$ for $W \in \mathbf{Fam}$ and $X \in \mathbf{Mod}$ defined as:

$$\kappa_X^W\ t\ w \triangleq X\langle \iota_2^{\Gamma,\Delta} \rangle t$$

⌟

Proposition 10.3.6 *For all $W \in \mathbf{Fam}$, the Day hom $(-)^W : \mathbf{Fam} \to \mathbf{Fam}$ lifts to modules, and κ is a natural transformation* $\mathrm{Id} \Longrightarrow (-)^W : \mathbf{Mod} \to \mathbf{Mod}$.

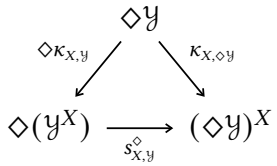
Proof The lifting of $(-)^X$ to $\Diamond$-algebras is induced by the strength $\Diamond(-)^X \to (\Diamond -)^X$ of Proposition 10.3.5. The diagonal transformation $\kappa_X^W : X \to X^W$ is a module homomorphism:

$$\begin{array}{ccccc} \Diamond X & & \xrightarrow{\quad x \quad} & & X \\ {\scriptstyle \Diamond \kappa_Y^W}\downarrow & & & & \downarrow{\scriptstyle \kappa_X^W} \\ \Diamond(X^W) & \xrightarrow[s^{\Diamond}_{W,X}]{} & (\Diamond X)^W & \xrightarrow[x^W]{} & X^W \end{array}$$

commutes by the following calculation, for $t \in X(\Gamma)$, $\rho \colon \Gamma \to \Delta$, $w \in W(\Theta)$:

$$\begin{aligned} \kappa_X^W(\langle \rho \rangle\, t)\, w &= \langle \iota_2^{\Delta,\Theta} \rangle\, (\langle \rho \rangle\, t) && (\kappa \triangleq) \\ &= \langle \iota_2^{\Delta,\Theta} \circ \rho \rangle\, t && (\text{functoriality}) \\ &= \langle (\rho + \Theta) \circ \iota_2^{\Gamma,\Theta} \rangle\, t && (\text{inl naturality}) \\ &= \langle \rho + \Theta \rangle (\langle \iota_2^{\Gamma,\Theta} \rangle\, t) && (\text{functoriality}) \\ &= y(\kappa_X^W\ t\ w, \rho + \Theta) && (\kappa \triangleq) \\ &= y(s^{\Diamond}_{W,X}(\kappa_X^W\ t, \rho)\ w) && (s^{\Diamond} \triangleq) \\ &= y^W(s^{\Diamond}_{W,X}(\Diamond \kappa_X^W\ (t, \rho)))\ w && \square \end{aligned}$$

Remark. Note that the seemingly canonical strength-diagonal compatibility condition



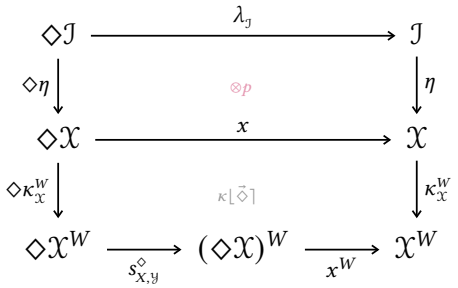
that, combined with naturality of κ, would establish the homomorphism law above, does not commute: the proof gets stuck at equating $(\Gamma + \Theta, X\langle \iota_2^{\Gamma,\Theta}\rangle\, t, \rho + \Theta)$ and $(\Gamma, t, \iota_2^{\Delta,\Theta} \circ \rho)$. These, of course, only equal after application of a multilinear map, such as the $\Diamond$-algebra structure. ⌟

Corollary 10.3.2 *For all $W \in$ **Fam**, the Day hom $(-)^W : \mathbf{Fam}_S \to \mathbf{Fam}_S$ lifts to pointed modules, and κ is a natural transformation* $\mathrm{Id} \Longrightarrow (-)^W : \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$.

Proof The lifting to $\Diamond$-algebras is as before. Given a pointed module $(\mathcal{X}, x, \eta)$, the point for $\mathcal{X}^W$ is the composite:

$$\mathcal{I} \xrightarrow{\eta} \mathcal{X} \xrightarrow{\kappa_{\mathcal{X}}^W} \mathcal{X}^W$$

This preserves points by definition, and is a module homomorphism by the following:



□

Notation. For $g \colon X \to Y^W$ and $w \in W(\Theta)$, we will write the mapping $t \in X(\Gamma) \mapsto g\, t\, w \in Y(\Gamma + \Theta)$ as $g\lfloor w \rfloor \colon X(\Gamma) \to Y(\Gamma + \Theta)$, similarly to how we write the partial application $t \in X(\Gamma) \mapsto f(\Gamma, t, \sigma)$ (for $f \colon X \ominus \mathcal{Y} \to Z$ and $\sigma \in {}^{\Gamma}\mathcal{Y}_{\Delta}$) as $f\{\sigma\} \colon X\Gamma \to Z\Delta$.
This extends to $\mathcal{X}^W$-valued substitution rules $\omega \in {}^{\Gamma}(\mathcal{X}^W)_{\Delta}$ as well: we denote $v \in \mathcal{I}_\alpha\Gamma \mapsto \omega\, v\, w \in {}^{\Gamma}\mathcal{X}_{\Delta+\Theta}$ as $\omega\lfloor w \rfloor \in {}^{\Gamma}\mathcal{X}_{\Delta+\Theta}$. ⌟

Definition 10.3.6 For a pointed family $(\mathcal{X}, \eta)$, the *left and right widening* of a substitution rule $\sigma \in {}^{\Delta}\mathcal{X}_{\Xi}$ by a renaming rule $\rho \colon \Gamma \to \Xi$ and $\varrho \colon \Theta \to \Xi$ are the respective substitution rules

$$\rho \ltimes \sigma \colon {}^{\Gamma+\Delta}\mathcal{X}_{\Xi} \qquad \sigma \rtimes \varrho \colon {}^{\Delta+\Theta}\mathcal{X}_{\Xi}$$
$$\rho \ltimes \sigma \triangleq [\eta \circ \rho,\ \sigma]_{\Xi}^{\Gamma,\Delta} \qquad \sigma \rtimes \varrho \triangleq [\sigma,\ \eta \circ \varrho]_{\Xi}^{\Delta,\Theta}$$

⌟

As widening is defined in terms of copairing, we directly have the following reduction properties by cocartesian universality:

$$(\rho \ltimes \sigma) \circ \mathrm{inl} = \eta \circ \rho \qquad (\rho \ltimes \sigma) \circ \mathrm{inr} = \sigma \qquad \mathrm{inl} \ltimes (\eta \circ \mathrm{inr}) = \eta$$
$$(\sigma \rtimes \varrho) \circ \mathrm{inl} = \sigma \qquad (\sigma \rtimes \varrho) \circ \mathrm{inr} = \eta \circ \varrho \qquad (\eta \circ \mathrm{inl}) \rtimes \mathrm{inr} = \eta$$

Nested widenings can be reassociated:

$$((\rho \ltimes \sigma) \rtimes \varrho) \circ \alpha_{\Gamma,\Delta,\Theta} = \rho \ltimes (\sigma \rtimes \varrho)$$

Naturality for a point-preserving map $f : \mathcal{X} \to \mathcal{Y}$ amounts to:

$$f \circ (\rho \ltimes \sigma) = \rho \ltimes (f \circ \sigma) \qquad f \circ (\sigma \rtimes \varrho) = (f \circ \sigma) \rtimes \varrho$$

If $\mathcal{X}$ is a pointed module, postcomposition with the module action can be pushed into the widening, using the fact that η is a module homomorphism:

$$\langle\varphi\rangle \circ (\rho \ltimes \sigma) = (\varphi \circ \rho) \ltimes (\langle\varphi\rangle \circ \sigma) \qquad \langle\varphi\rangle \circ (\sigma \rtimes \varrho) = (\langle\varphi\rangle \circ \sigma) \rtimes (\varphi \circ \varrho)$$

The following identity will be used several times in situations that the substitution rule to be widened is $(\mathcal{X}^W)$-valued.

Lemma 10.3.1 *For $\sigma \in {}^{\Delta}(\mathcal{X}^W)_{\Xi}$ and renaming rules $\rho : \Gamma \to \Xi$ and $\varrho : \Theta \to \Xi$, the application of a widening $\rho \ltimes \sigma \in {}^{\Gamma+\Delta}(\mathcal{X}^W)_{\Xi}$ or $\sigma \rtimes \varrho \in {}^{\Delta+\Theta}(\mathcal{X}^W)_{\Xi}$ to a convolutional parameter $w \in \Omega$ reduces:*

$$\begin{aligned}(\rho \ltimes \sigma)\lfloor w \rfloor &= (\iota_2^{\Xi,\Omega} \circ \rho) \ltimes \sigma\lfloor w \rfloor && \in {}^{\Gamma+\Delta}(\mathcal{X})_{\Xi+\Omega}\\ (\sigma \rtimes \varrho)\lfloor w \rfloor &= \sigma\lfloor w \rfloor \rtimes (\iota_2^{\Xi,\Omega} \circ \varrho) && \in {}^{\Delta+\Theta}(\mathcal{X})_{\Xi+\Omega}\end{aligned}$$

Proof The identities are established by precomposing them with injections. The first identity, in the $\iota_1^{\Gamma,\Delta}$ case, reduces simply to $\sigma\lfloor w \rfloor$; while in the $\iota_2^{\Gamma,\Delta}\, v$ case we have

$$\begin{aligned}&(\rho \ltimes \sigma)\lfloor w \rfloor\, (\iota_2^{\Gamma,\Delta}\, v)\\ &= (\eta_{\mathcal{X}^W}\, (\rho\, v))\, w && (\ltimes \triangleq)\\ &= \kappa_{\mathcal{X}}\, (\eta_{\mathcal{X}}\, (\rho\, v))\, w && (\eta_{\mathcal{X}^W} \triangleq)\\ &= \langle\iota_2^{\Xi,\Omega}\rangle(\eta_{\mathcal{X}}\, (\rho\, v)) && (\kappa \triangleq)\\ &= \eta_{\mathcal{X}}\, (\iota_2^{\Xi,\Omega}(\rho\, v)) && (\eta \text{ naturality})\end{aligned}$$

The second identity follows similarly, and the analogous reasoning works for the reversed Day hom $\bullet\!\!-$ (see later). □

A form of naturality can be stated for parametrised maps as well; though it looks somewhat innocuous, this is in fact one of the most important identities of our theory.

Lemma 10.3.2 *For all pointed multilinear maps $f : \mathcal{X} \multimap \boxed{\mathcal{Y}} \to \mathcal{Z}$ and all substitution rules $\sigma \in {}^{\Delta}\mathcal{X}_{\Xi}$ and $\varsigma \in {}^{\Xi}\mathcal{Y}_{\Omega}$:*

- *if $\rho : \Gamma \to \Xi$ and $\varrho : \Gamma \to \Omega$ are such that $\varsigma \circ \rho = \eta_{\mathcal{Y}} \circ \varrho \in {}^{\Gamma}\mathcal{Y}_{\Omega}$, then*

$$f\{\varsigma\} \circ (\rho \ltimes \sigma) = \varrho \ltimes (f\{\varsigma\} \circ \sigma) \in {}^{\Gamma+\Delta}\mathcal{Z}_{\Omega}$$

- *if $\varphi : \Theta \to \Xi$ and $\phi : \Theta \to \Omega$ are such that $\varsigma \circ \varphi = \eta_{\mathcal{Y}} \circ \phi \in {}^{\Theta}\mathcal{Y}_{\Omega}$ then*

$$f\{\varsigma\} \circ (\sigma \rtimes \varphi) = (f\{\varsigma\} \circ \sigma) \rtimes \phi \in {}^{\Delta+\Theta}\mathcal{Z}_{\Omega}$$

Proof Let f, σ, ς be as above. For $\rho : \Gamma \to \Xi$, $\varrho : \Gamma \to \Omega$, assume ① $\varsigma \circ \rho = \eta_{\mathcal{Y}} \circ \varrho$. To show the equality of the two substitution rules ${}^{\Gamma+\Delta}\mathcal{Z}_{\Omega}$ from a concatenated context, by universality of coproducts it is sufficient to show that they are equal when precomposed with the injections.

$$
\begin{aligned}
& f\{\varsigma\} \circ (\rho \ltimes \sigma) \circ \iota_2^{\Gamma,\Delta} \\
&= f\{\varsigma\} \circ \eta_{\mathcal{X}} \circ \rho && \text{(widening property)} \\
&= f\{\varsigma\} \circ \mathcal{X}\langle\rho\rangle \circ \eta_{\mathcal{X}} && (\eta_{\mathcal{X}} \text{ naturality}) \\
&= f\{\varsigma \circ \rho\} \circ \eta_{\mathcal{X}} && (f \text{ multilinear}) \\
&= f\{\eta_{\mathcal{Y}} \circ \varrho\} \circ \eta_{\mathcal{X}} && (①) \\
&= f\{\eta_{\mathcal{Y}}\} \circ \mathcal{X}\langle\varrho\rangle \circ \eta_{\mathcal{X}} && (f \text{ multilinear}) \\
&= f\{\eta_{\mathcal{Y}}\} \circ \eta_{\mathcal{X}} \circ \varrho && (\eta_{\mathcal{X}} \text{ naturality}) \\
&= \eta_{\mathcal{Z}} \circ \varrho && (f \text{ pointed}) \\
&= (\varrho \ltimes (f\{\varsigma\} \circ \sigma)) \circ \iota_2^{\Gamma,\Delta} && \text{(widening property)}
\end{aligned}
$$

The other case $f\{\varsigma\} \circ (\rho \ltimes \sigma) \circ \iota_1^{\Gamma,\Delta} = (\varrho \ltimes (f\{\varsigma\} \circ \sigma)) \circ \iota_1^{\Gamma,\Delta}$ is direct by the widening reduction properties. The calculation for right widening is analogous. □

As discussed in Section 10.2.2, $\mathcal{I}/\mathbf{Mod}_S$ is synthetic monoidal over $\mathbf{Fam}_S$. We relate this structure to the Day hom in the following theorem.

Theorem 10.3.1

For all $W \in \mathbf{Fam}$, the lifted functor $(-)^W : \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$ is a synthetic monoidal endofunctor on $\mathcal{I}/\mathbf{Mod}_S$.

Proof See the Appendix on page 336. □

Now, let us consider $\ominus : \mathbf{Fam} \times \mathbf{Fam}_S \to \mathbf{Fam}$ as an action $\ominus : \mathbf{Fam} \times \mathcal{I}/\mathbf{Mod}_S \to \mathbf{Fam}$, which makes $\mathbf{Fam}$ a synthetic $\mathcal{I}/\mathbf{Mod}_S$-modular category.

Proposition 10.3.7 *The diagonal $\kappa^W : \mathrm{Id} \Longrightarrow (-)^W$ is a synthetic monoidal transformation.*

Proof The diagonal preserves the unit operators by definition. We need to show that for all pointed multilinear maps $g : \mathcal{X} \oplus \mathcal{Y} \to \mathcal{Z}$, the following square commutes:

$$
\begin{array}{ccc}
\mathcal{X} \oplus \mathcal{Y} & \xrightarrow{f} & \mathcal{Z} \\
{\scriptstyle \kappa^W_{\mathcal{X}} \oplus \kappa^W_{\mathcal{Y}}} \downarrow & & \downarrow {\scriptstyle \kappa^W_{\mathcal{Z}}} \\
\mathcal{X}^W \oplus \mathcal{Y}^W & \xrightarrow[m[f]]{} & \mathcal{Z}^W
\end{array}
$$

For $t \in \mathcal{X}_\alpha\Gamma$, $\sigma \in {}^{\Gamma}\mathcal{Y}_{\Delta}$ and $w \in W(\Theta)$, we calculate as follows:

$$
\begin{aligned}
& m[f]\{\kappa_{\mathcal{Y}} \circ \sigma\}\lfloor w \rfloor \circ \kappa_{\mathcal{X}} \\
&= f\{(\kappa_{\mathcal{Y}} \circ \sigma)\lfloor w \rfloor \rtimes \mathrm{inr}\} \circ \kappa_{\mathcal{X}}\lfloor w \rfloor && (m[f] \triangleq) \\
&= f\{(\mathcal{Y}\langle\mathrm{inl}\rangle \circ \sigma) \rtimes \mathrm{inr}\} \circ \mathcal{X}\langle\mathrm{inl}\rangle && (\kappa \triangleq) \\
&= f\{((\mathcal{Y}\langle\mathrm{inl}\rangle \circ \sigma) \rtimes \mathrm{inr}) \circ \mathrm{inl}\} && (f \text{ multilinear}) \\
&= f\{\mathcal{Y}\langle\mathrm{inl}\rangle \circ \sigma\} && \text{(widening property)}
\end{aligned}
$$

$$
\begin{aligned}
&= \mathcal{Z}\langle \mathrm{inl} \rangle \circ f\{\sigma\} && (f \text{ multilinear}) \\
&= \kappa_{\mathcal{Z}} \lfloor w \rfloor \circ f\{\sigma\} && (\kappa \triangleq)
\end{aligned}
$$

□

Proposition 10.3.8 *The Day hom on families* $(-)^W : \mathbf{Fam} \to \mathbf{Fam}$ *is a* $(-)^W$*-relative synthetic module functor.*

Proof We will continue writing $(-)^W$ for both $\mathbf{Fam} \to \mathbf{Fam}$ and $\mathcal{I}/\mathbf{Mod}_s \to \mathcal{I}/\mathbf{Mod}_s$ endofunctors, with the context (namely, whether the "base" is $X \in \mathbf{Fam}$ or $\mathcal{X} \in \mathcal{I}/\mathbf{Mod}_s$) making it clear which endofunctor is taken.

Define the strength operator $d^W_{X,\mathcal{Y}} : X^W \ominus \mathcal{Y}^W \to (X \ominus \mathcal{Y})^W$ similarly to $m[-]$ above, but without the use of a parametrised map (as $d[f : X \ominus \mathcal{Y} \to Z] : X^W \ominus \mathcal{Y}^W \to Z^W$ can be derived):

$$d^W_{X,\mathcal{Y}}(\Gamma, l, \omega)\, w \triangleq (\Gamma + \Theta, l\, w, \omega \lfloor w \rfloor \ltimes \iota_1^{\Delta,\Theta})$$

We show that this satisfies the laws of a synthetic modular functor.

- The unit law simplifies to the following:

$$\begin{array}{ccccc}
X^W & \xrightarrow{(\rho_X)^W} & (X \ominus \mathcal{I})^W & \xrightarrow{(X \ominus \eta_{\mathcal{Y}})^W} & (X \ominus \mathcal{Y})^W \\
{\scriptstyle \rho_{X^W}}\downarrow & & & & \uparrow{\scriptstyle d_{X,\mathcal{Y}}} \\
X^W \ominus \mathcal{I} & \xrightarrow[\mathrm{id} \ominus \eta_{\mathcal{Y}}]{} & X^W \ominus \mathcal{Y} & \xrightarrow[\mathrm{id} \ominus \kappa^W_{\mathcal{Y}}]{} & X^W \ominus \mathcal{Y}^W
\end{array}$$

  Taking $l \in (X^W)\Gamma$ and $w \in W(\Theta)$, we calculate as follows:

$$d\{\kappa \circ \eta\} \lfloor w \rfloor\, l = (l\, w, (\kappa \circ \eta) \lfloor w \rfloor \ltimes \mathrm{inr}) = (l\, w, \eta)$$

  where the last step appeared in the right unit law of Theorem 10.3.1.

- For a pointed multilinear map $f : \mathcal{Y} \oplus \mathcal{Z} \to \mathcal{U}$, the associativity law simplifies to

$$\begin{array}{ccccc}
(X^W \ominus \mathcal{Y}^W) \ominus \mathcal{Z}^W & \xrightarrow{d_{X,\mathcal{Y}} \ominus \mathrm{id}} & (X \ominus \mathcal{Y})^W \ominus \mathcal{Z}^W & \xrightarrow{d_{X \ominus \mathcal{Y}, \mathcal{Z}}} & ((X \ominus \mathcal{Y}) \ominus \mathcal{Z})^W \\
{\scriptstyle \alpha_{X^W, \mathcal{Y}^W, \mathcal{Z}^W}}\downarrow & & & & \downarrow{\scriptstyle (\alpha_{X,\mathcal{Y},\mathcal{Z}})^W} \\
X^W \ominus (\mathcal{Y}^W \oplus \mathcal{Z}^W) & & & & (X \ominus (\mathcal{Y} \oplus \mathcal{Z}))^W \\
{\scriptstyle \mathrm{id} \ominus m[f]}\downarrow & & & & \downarrow{\scriptstyle (X \ominus f)^W} \\
X^W \ominus \mathcal{U}^W & & \xrightarrow[d_{X,\mathcal{U}}]{} & & (X \ominus \mathcal{U})^W
\end{array}$$

  Taking $l \in X^W(\Gamma)$, $\omega \in {}^{\Gamma}(\mathcal{V}^W)_{\Delta}$, $\varpi \in {}^{\Delta}(\mathcal{W}^W)_{\Theta}$, the calculation proceeds as follows:

$$
\begin{aligned}
&(\alpha[f]^W)(d\{\varpi\} \circ d\{\omega\}) \lfloor w \rfloor\, l \\
&= \alpha[f](d\{\varpi\} \lfloor w \rfloor (d\{\omega\}\, l)) \\
&= \alpha[f](d\{\omega\} \lfloor w \rfloor\, l, \omega \lfloor w \rfloor \ltimes \mathrm{inr}) && (d \triangleq) \\
&= \alpha[f]((l\, w, \omega \lfloor w \rfloor \ltimes \mathrm{inr}), \omega \lfloor w \rfloor \ltimes \mathrm{inr}) && (d \triangleq)
\end{aligned}
$$

$$
\begin{aligned}
&= (l\,w, f\{\varpi\lfloor w\rfloor \rtimes \mathrm{inr}\} \circ (\omega\lfloor w\rfloor \rtimes \mathrm{inr})) && (\alpha[f] \triangleq)\\
&= (l\,w, (m[f]\{\varpi\} \circ \omega)\lfloor w\rfloor \rtimes \mathrm{inr})\\
&= d\{m[f]\{\varpi\} \circ \omega\}\lfloor w\rfloor\, l && (d \triangleq)
\end{aligned}
$$

where the unlabelled step is taken from the associativity of $m[-]$ in Theorem 10.3.1.

Thus, we have shown that the unsorted functor $(-)^W$ is near-genuine strong relative to $(-)^W : \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$. This of course extends to the sorted $(-)^W : \mathbf{Fam}_S \to \mathbf{Fam}_S$. □

The theory so far was given for the left-to-right direction of the Day hom $\multimap$ and $\bullet\!\!\multimap$, but can be symmetrically developed for the opposite directions $\circ\!\!-$ and $\bullet\!\!-$ by reversing the copairing and injections. Thus, abbreviating $(W \circ\!\!- (-))$ as $(W \circ\!\!-)$, we also have:

Corollary 10.3.3 *The reverse Day exponentiation* $(W \circ\!\!-) : \mathbf{Fam} \to \mathbf{Fam}$ *is synthetic* $\mathcal{I}/\mathbf{Mod}_S$*-strong over the synthetic monoidal* $(W \bullet\!\!-) : \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$.

Since context extension is defined in terms of $\circ\!\!-$, the following result – though *very long* in the making – is satisfyingly immediate.

Theorem 10.3.2

The context extension endofunctor $\delta_\Theta : \mathbf{Fam} \to \mathbf{Fam}$ *is synthetic strong.*

Proof The strength transformation

$$s^{\delta}_{X,\mathcal{Y}} : \delta_\Theta X \ominus \mathcal{Y} \to \delta_\Theta(X \ominus \mathcal{Y})$$

is the pullback of the $(\mathcal{T}\Theta \bullet\!\!-)$-relative strength transformation for $(\mathcal{T}\Theta \circ\!\!-)$ along the synthetic monoidal natural transformation $\kappa : \mathrm{Id} \Longrightarrow (\mathcal{T}\Theta \bullet\!\!-)$ (Proposition 10.3.7):

$$\delta_\Theta X \ominus \mathcal{Y} = (\mathcal{T}\Theta \circ\!\!- X) \ominus \mathcal{Y} \xrightarrow{\mathrm{id}\ominus\kappa^{\mathcal{T}\Theta}_{\mathcal{Y}}} (\mathcal{T}\Theta \circ\!\!- X) \ominus (\mathcal{T}\Theta \bullet\!\!- \mathcal{Y}) \xrightarrow{d^{\mathcal{T}\Theta}_{X,\mathcal{Y}}} \mathcal{T}\Theta \circ\!\!- (X \ominus \mathcal{Y}) = \delta_\Theta(X \ominus \mathcal{Y})$$

The unit and associativity axioms are inherited from the $(\mathcal{T}\Theta \bullet\!\!-)$-relative strength transformation by the pullback. □

The results can be transposed to the closed substitution structure, equipping context extension with a closed strength

$$\delta_\Theta[\![\mathcal{X}, \mathcal{Y}]\!] \to [\![\mathcal{X}, \delta_\Theta \mathcal{Y}]\!] : (\mathcal{I}/\mathbf{Mod}_S)^{\mathrm{op}} \times \mathbf{Fam} \to \mathbf{Fam}$$

satisfying the desired laws of Section 10.2.3. For simplicity, we will refer to functors with such pointed synthetic strength transformations from now on simply as *strong*, distinguishing them from powered functors introduced next.

10.3.3 Convolutional powering

In the preceding section we established that the context extension endofunctor derived from the Day convolutional structure is synthetic strong over the skew-monoidal substitution structure. This is essential for the reconciliation of second-order algebraic and substitution structure, as the strength for the signature endofunctor (which may involve context extension to represent variable binding) is needed to express the algebraic monoid compatibility condition.

When we focus on the second-order features of syntax – namely, metavariables and metasubstitution – we also need to consider the interaction of the syntax endofunctor with the convolutional structure itself, in the form of convolutional powering, and compatibility with synthetic strengths. In the presheaf model of Fiore (2008) the cartesian closed structure of presheaves is rich enough to capture metasubstitution, but in the familial model we are compelled to resort to Day convolution and homs again to faithfully represent the intricate context-manipulation of second-order operations. As an upshot, sorted families are more suitable for representing metavariable families, leading to a simplified theory of metasubstitution.

In this section we focus on the interaction of powering over the Day convolution structure, and pointed strength over the skew substitution structure. We recall the definition of a powered functor below.

Definition 10.3.7 A functor $F\colon \mathbf{Fam} \to \mathbf{Fam}$ is *powered* if it comes with a natural operation

$$p^F_{W,Y}\colon F(W \multimap Y) \to (W \multimap FY)$$

satisfying the laws of a skew-closed (in fact, strong closed) module functor in Definition 5.1.11, or equivalently, the laws of a monoidal closed category:

$$\begin{array}{ccc} F(E \multimap Y) & \xrightarrow{p_{E,Y}} & E \multimap FY \\ & \searrow^{Fi^{\multimap}_{Y}} & \downarrow{\scriptstyle i^{\multimap}_{FY}} \\ & & FY \end{array} \qquad (pi)$$

$$\begin{array}{ccccc} F((U \circledast W) \multimap Y) & & \xrightarrow{p_{U\circledast W,Y}} & & (U \circledast W) \multimap FY \\ {\scriptstyle Fc^{U,W}_{Y}}\downarrow & & & & \downarrow{\scriptstyle c^{U,W}_{FY}} \\ F(W \multimap (U \multimap Y)) & & & & W \multimap (U \multimap FY) \\ & {\scriptstyle p_{W,U\multimap Y}}\searrow & & \nearrow{\scriptstyle W\multimap p_{U,Y}} & \\ & & W \multimap F(U \multimap Y) & & \end{array} \qquad (pc)$$

If a functor possesses both a strength and a powering, we want them to respect each other: in practice, this will correspond to a compatibility of the algebraic, substitution, and metasubstitution structure. The equivalent notion is stated in the cartesian setting by Fiore (2008, Definition 12).

Definition 10.3.8 If $F\colon \mathbf{Fam} \to \mathbf{Fam}$ is equipped both with a powering $p_{W,X}\colon F(X^W) \to (FX)^W$ and a strength $s_{X,\mathcal{Y}}\colon FX \ominus \mathcal{Y} \to F(X \ominus \mathcal{Y})$, the powering and strength are *compatible* if the following diagram commutes:

$$\begin{array}{ccccc} F(X^W) \ominus \mathcal{Y}^W & \xrightarrow{s_{X^W,\mathcal{Y}^W}} & F\big(X^W \ominus \mathcal{Y}^W\big) & \xrightarrow{Fd^W_{X,\mathcal{Y}}} & F\big((X \ominus \mathcal{Y})^W\big) \\ {\scriptstyle p_{W,X}\ominus \mathrm{id}}\downarrow & & & & \downarrow{\scriptstyle p_{W,X\ominus\mathcal{Y}}} \\ (FX)^W \ominus \mathcal{Y}^W & \xrightarrow[d^W_{FX,\mathcal{Y}}]{} & (FX \ominus \mathcal{Y})^W & \xrightarrow[(s_{X,\mathcal{Y}})^W]{} & (F(X \ominus \mathcal{Y}))^W \end{array} \qquad (sdp)$$

For a fixed W, the powering $p_{W,X}\colon F(X^W) \to (FX)^W$ for F acts as an elevator from the functor F to the functor $(-)^W$, and the compatibility diagram above actually exhibits $p_W\colon F((-)^W) \Longrightarrow (F-)^W$ as an elevator in the total category of synthetic strong functors from $(\mathrm{Id}, (F, s))\colon (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}) \to (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam})$ to the synthetic $\mathcal{I}/\mathbf{Mod}_S$-module functor $((W \multimap \bullet), ((W, \multimap), d))\colon (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}) \to (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam})$, a construction spelled out in concrete detail in Section 6.3. We next focus on the context extension endofunctor δ.

Proposition 10.3.9 *The reversed Day hom* $(U \multimap -) : \mathbf{Fam} \to \mathbf{Fam}$ *is powered.*

Proof For $Y, W \in \mathbf{Fam}$, the strength transformation

$$e_Y^{U,W} : U \multimap (W \multimap Y) \to W \multimap (U \multimap Y)$$

is an isomorphism derived from properties of Day homs in Proposition 8.2.1. Calculating via the Yoneda embedding and the internal currying $(X \circledcirc U) \multimap Y \cong (X \multimap (U \multimap Y))$, we have:

$$\begin{aligned}\mathbf{Fam}(X, U \multimap (W \multimap Y)) &\cong \mathbf{Fam}(X \circledcirc U, W \multimap Y)\\ &\cong \mathbf{Fam}(W, (X \circledcirc U) \multimap Y)\\ &\cong \mathbf{Fam}(W, X \multimap (U \multimap Y))\\ &\cong \mathbf{Fam}(X, W \multimap (U \multimap Y))\end{aligned}$$

From first principles, this simply corresponds to an argument swapping with a context reassociation: for $l : (U \multimap (W \multimap Y))(\Gamma)$, $w : W(\Delta)$ and $u : U(\Theta)$,

$$e_Y^{U,W}\, l\, w\, u \triangleq Y(\alpha_{\Theta,\Gamma,\Delta})(l\, u\, w) \in Y(\Theta + (\Gamma + \Delta))$$

where, of course, $\alpha_{\Theta,\Gamma,\Delta}$ is just the identity, as $(\Theta+\Gamma)+\Delta$ and $\Theta+(\Gamma+\Delta)$ are strictly equal in the discrete category of lists. Consequently, the strength axioms also hold on the nose: the unit law becomes $e\, l * u = l\, u *$ (where $*$ is the only element of the set $E[]$), and the associativity law computes as:

$$(U \multimap e)(e(c\, l))\, v\, w\, u = e(e(c\, l)\, v\, w)\, u = e(l\, (w, v))\, u = l\, u\, (v, w) = c(e\, l)\, v\, w\, u$$

If Y happens to be a module, the reassociation of the term context can be performed by a module action on the renaming rule $((\Theta+\Gamma)+\Delta) \to (\Theta+(\Gamma+\Delta))$, so $Y(\alpha^{S^*}_{\Theta,\Gamma,\Delta}) : Y((\Theta+\Gamma)+\Delta) \to Y(\Theta + (\Gamma + \Delta))$ can be equivalently written as $Y\langle \alpha^{\mathbb{F}[S]}_{\Theta,\Gamma,\Delta} \rangle$. □

Corollary 10.3.4 *The context extension endofunctor* $\delta_\Theta : \mathbf{Fam} \to \mathbf{Fam}$ *is powered.*

Proof Follows from the previous proposition, with $U \triangleq \mathfrak{T}\Theta$. □

The following proposition uses the synthetic lifting framework of Section 6.3 to generalise the compatibility law in Diagram (*sdp*) for F the Day hom.

Proposition 10.3.10 *For* $U, W \in \mathbf{Fam}$*, there is an elevator in* $\mathbf{SynMod}$ *from the* $(U \bullet\!\!-)$*-relative* $\mathfrak{I}/\mathbf{Mod}_S$*-module endofunctor* $(U \multimap -) : \mathbf{Fam} \to \mathbf{Fam}$ *to the* $(W \multimap\!\!\bullet)$*-relative* $\mathfrak{I}/\mathbf{Mod}_S$*-module endofunctor* $(W \multimap -) : \mathbf{Fam} \to \mathbf{Fam}$*.*

Proof See the Appendix on page 338. □

Lemma 10.3.3 *For a module* X*, we have the compatibility condition between* e *and* κ*:*

$$\begin{array}{ccc} \mathcal{X}^W & \xrightarrow{\varkappa^U_{\mathcal{X}W}} & (U \bullet\!\!- (W \multimap\!\!\bullet \mathcal{X})) \\ & \searrow^{W \multimap\!\!\bullet (\varkappa^U_{\mathcal{X}})} & \downarrow{\scriptstyle e^{U,W}_{\mathcal{X}}} \\ & & W \multimap\!\!\bullet (U \bullet\!\!- \mathcal{X}) \end{array}$$

Proof Take $l \in \mathcal{X}^W(\Gamma)$, $w \in W(\Delta)$ and $u \in U(\Theta)$, and calculate as follows:

$$\begin{aligned}
& e^{U,W}_{\mathcal{X}}(\varkappa^U_{\mathcal{X}^W}\, l)\, w\, u \\
&= \mathcal{X}\langle \alpha_{\Theta,\Gamma,\Delta}\rangle(\varkappa^U_{\mathcal{X}^W}\, l\, u\, w) && (e^{\triangleq}) \\
&= \mathcal{X}\langle \alpha_{\Theta,\Gamma,\Delta}\rangle(\mathcal{X}^W\langle \iota_1^{\Theta,\Gamma}\rangle\, l\, w) && (\varkappa^{\triangleq}) \\
&= \mathcal{X}\langle \alpha_{\Theta,\Gamma,\Delta}\rangle(\mathcal{X}\langle \iota_1^{\Theta,\Gamma} + \Delta\rangle\, (l\, w)) && \text{(module action for } \mathcal{X}^W) \\
&= \mathcal{X}\langle \alpha_{\Theta,\Gamma,\Delta} \circ (\iota_1^{\Theta,\Gamma} + \Delta)\rangle(l\, w) && \text{(module associativity)} \\
&= \mathcal{X}\langle \iota_1^{\Theta,\Gamma+\Delta}\rangle(l\, w) && \text{(cocartesian coherence)} \\
&= \varkappa^U_{\mathcal{X}}\,(l\, w)\, u && (\varkappa^{\triangleq}) \\
&= (W \multimap \varkappa^U_{\mathcal{X}})\, l\, w\, u && \square
\end{aligned}$$

Remark. Of course, by the symmetry of the operations, a similar property holds for the interaction of κ and e as well. In fact, both $(W \multimap)$ and $(U \circ\!\!-)$ form monads on modules, with κ and $\varkappa$ the respective units; $e^{U,W}_Y : U \circ\!\!- (W \multimap Y) \to W \multimap (U \circ\!\!- Y)$ is then a monad-monad distributive law, with the lemma above being one of the unit preservation conditions. The monad structure of the exponentials will not be needed for our purposes. ⌟

Theorem 10.3.3

The strength and powering for the context extension endofunctor are compatible.

Proof We have the strength transformations

$$\begin{aligned}
d_{X,\mathcal{Y}} &: \delta_\Theta(X) \ominus \delta_\Theta(\mathcal{Y}) \to \delta_\Theta(X \ominus \mathcal{Y}) : \mathbf{Fam} \times \mathcal{I}/\mathbf{Mod}_S \to \mathbf{Fam} \\
s_{X,\mathcal{Y}} &: \delta_\Theta(X) \ominus \mathcal{Y} \to \delta_\Theta(X \ominus \mathcal{Y}) : \mathbf{Fam} \times \mathcal{I}/\mathbf{Mod}_S \to \mathbf{Fam} \\
p_{W,X} &: \delta_\Theta(W \multimap X) \to (W \multimap \delta_\Theta X) : \mathbf{Fam}^{\mathrm{op}} \times \mathbf{Fam} \to \mathbf{Fam}
\end{aligned}$$

As shown in Theorem 10.3.2, the (Id-relative) strength s for δ_Θ is derived from its δ_Θ-relative strength d by pulling back along the synthetic monoidal natural transformation $\kappa^{\mathfrak{F}\Theta} : \mathrm{Id} \Longrightarrow \delta_\Theta$. Proposition 10.3.10 with $U \triangleq \mathfrak{F}\Theta$ showed that p extends to an elevator from $(\delta_\Theta, \delta_\Theta) : (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}) \to (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam})$ to $((W \multimap), (W \multimap)) : (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam}) \to (\mathcal{I}/\mathbf{Mod}_S, \mathbf{Fam})$. Instantiating Lemma 6.3.1 with $F = G = \delta_\Theta$, $M = (W \multimap)$, $K = L = \delta_\Theta$, $K' = L' = \mathrm{Id}$, $\varphi \triangleq p^\delta_W : \delta_\Theta(W \multimap (-)) \Longrightarrow W \multimap \delta_\Theta(-)$, $\varphi' = \mathrm{id}$, $\alpha = \beta = \varkappa^{\mathfrak{F}\Theta} : \mathrm{Id} \Longrightarrow \delta_\Theta$, and the compatibility condition between p and $\varkappa$ proved in Lemma 10.3.3, we have that the elevator $d : d^\delta_{X,\mathcal{Y}} : \delta_\Theta(X) \ominus \delta_\Theta(\mathcal{Y}) \to \delta_\Theta(X \ominus \mathcal{Y})$ induces an elevator relative to the pulled back modules $p_{W,X} : \delta_\Theta(W \multimap X) \to (W \multimap \delta_\Theta X)$. The strength-preservation condition of the elevator is

$$\begin{array}{ccccc}
\delta_\Theta(X^W) \ominus \mathcal{Y}^W & \xrightarrow{s_{X^W,\mathcal{Y}^W}} & \delta_\Theta(X^W \ominus \mathcal{Y}^W) & \xrightarrow{F d_{X,\mathcal{Y}}} & \delta_\Theta((X \ominus \mathcal{Y})^W) \\
\downarrow{\scriptstyle p_{X,W} \ominus \mathrm{id}} & & & & \downarrow{\scriptstyle p_{W,X\ominus\mathcal{Y}}} \\
(\delta_\Theta X)^W \ominus \mathcal{Y}^W & \xrightarrow[d_{\delta_\Theta X,\mathcal{Y}}]{} & (\delta_\Theta X \ominus \mathcal{Y})^W & \xrightarrow[(s_{X,\mathcal{Y}})^W]{} & (\delta_\Theta(X \ominus \mathcal{Y}))^W
\end{array}$$

which is exactly the compatibility axiom Diagram (*sdp*) for the skew and powerings for δ_Θ. □

The preceding theorem will take care of the difficult part of equipping signature endofunctors with compatible skew and powerings. Constructing these strengths for the product and

coproduct functors is much easier, as they do not manipulate the context. We demonstrate the binary case below, with the n-ary case an obvious generalisation. Note that we are not strictly dealing with endofunctors any more as $\times$ and $+$ are bifunctors $\mathbf{Fam} \times \mathbf{Fam} \to \mathbf{Fam}$; the definitions of actions and strengths are easy enough to formally generalise to multi-arity functors (see e.g. the *structural strength* of Borthelle et al. (2020, Definition 2.11)), but since the compatible strengths for products and coproducts are straightforward to construct, this generality is not a worthwhile complication.

Proposition 10.3.11 *The product and coproduct functors* $\times, + \colon \mathbf{Fam} \times \mathbf{Fam} \to \mathbf{Fam}$ *can be equipped with compatible skew and powerings.*

Proof Define the relative pointed strength and convolution strength for $\times$ as follows:

$$s^{\times} \colon (W \times X) \ominus \mathcal{Y} \to (W \ominus \mathcal{Y}) \times (X \ominus \mathcal{Y}) \qquad s^{+} \colon (X + Y) \ominus \mathcal{Z} \to (X \ominus \mathcal{Z}) + (Y \ominus \mathcal{Z})$$
$$s^{\times}\big(\Gamma, (t_1, t_2), \sigma\big) \triangleq \big((\Gamma, t_1, \sigma), (\Gamma, t_2, \sigma)\big) \qquad s^{+}\big(\Gamma, \mathrm{inl}|\mathrm{inr}\, t, \sigma\big) \triangleq \mathrm{inl}|\mathrm{inr}\, (\Gamma, t, \sigma)$$

$$p^{\times} \colon (W \multimap X) \times (W \multimap Y) \to W \multimap (X \times Y) \qquad p^{+} \colon (W \multimap X) + (W \multimap Y) \to W \multimap (X + Y)$$
$$p^{\times}(l_1, l_2)\, w \triangleq (l_1\, w, l_2\, w) \qquad p^{+}(\mathrm{inl}|\mathrm{inr}\, l)\, w \triangleq \mathrm{inl}|\mathrm{inr}\, (l\, w)$$

The simple structural definitions make the strength laws and skew-convolutional compatibility axioms hold by definition. □

The results of this section culminate in the following theorem, restating Theorem 14 of Fiore (2008) in the familial model.

Theorem 10.3.4

> *Every* **Fam**-*endofunctor built by composition from* Id, $+$, $\times$ *and* δ_{Θ} *comes equipped with compatible skew and powerings.*

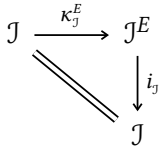
We will call endofunctors constructed according to this theorem *signature endofunctors*. As endofunctors for second-order signatures will give rise to signature endofunctors, this theorem equips them with the structure to axiomatise signature-compatible substitution and metasubstitution structure. In the final section of this chapter, we conclude our translation from the presheaf to the familial model by focusing on initial algebras for endofunctors.

10.3.4 Algebraic monoids

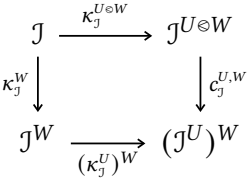
We round up our discussion of convolutional parametrisation by examining the relationship of the Day monoidal closed structure and second-order models for a syntax. The properties proved here will be crucial for the development of the metasubstitution structure, where the convolutional and skew substitution structure interact in intricate ways.

The semi-sorted linear hom $\multimap \bullet \colon \mathbf{Fam}^{\mathrm{op}} \times \mathbf{Fam}_S \to \mathbf{Fam}_S$ exhibits $\mathbf{Fam}_S$ as powered over $\mathbf{Fam}$, with structural $i_{\mathcal{X}} \colon (E \multimap \mathcal{X}) \to \mathcal{X}$ and $c_{\mathcal{X}}^{U,W} \colon (U \otimes W) \multimap \mathcal{X} \to (W \multimap (U \multimap \mathcal{X}))$. Not only does $(W \multimap) \colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ lift to a synthetic monoidal functor on pointed modules for all W, the structural transformations also respect this functorial structure.

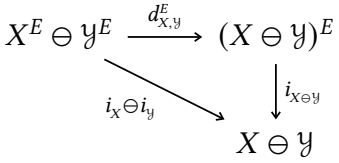
Proposition 10.3.12 *For all* $U, W \in \mathbf{Fam}$*, the transformations* $i_{\mathcal{X}} \colon (E \multimap \mathcal{X}) \to \mathcal{X} \colon \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$ *and* $c_{\mathcal{X}}^{U,W} \colon \mathcal{X}^{U \otimes W} \to (\mathcal{X}^{U})^{W} \colon \mathbf{Fam}^{\mathrm{op}} \times \mathbf{Fam}^{\mathrm{op}} \times \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$ *are synthetic monoidal transformations:*



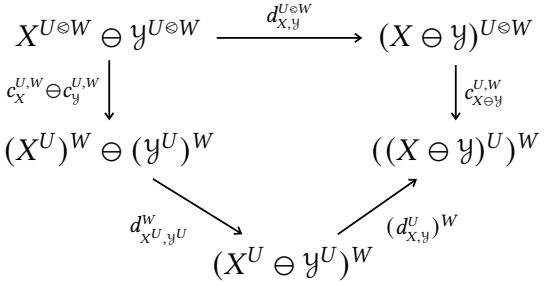
(κi)



(κc)



(di)



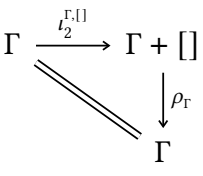
(dc)

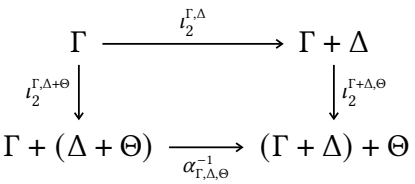
Proof The proofs are nearly immediate if we keep applications of context equalities implicit, but we spell these out for the sake of rigour. For the axioms concerning κ^E, take a $v \in \mathcal{I}_\alpha\Gamma$, $w \in W(\Delta)$ and $u \in U(\Theta)$, and calculate:

$$
\begin{aligned}
& i\,(\kappa\, v) \\
&= \mathcal{I}(\rho_\Gamma)\kappa\, v\, * && (i \triangleq) \\
&= \mathcal{I}(\rho_\Gamma)\mathcal{I}(\iota_2^{\Gamma,[]})\, v && (\kappa \triangleq) \\
&= \mathcal{I}(\rho_\Gamma \circ \iota_2^{\Gamma,[]})\, v && \text{(functoriality)} \\
&= v && \text{(cocartesian coherence)}
\end{aligned}
$$

$$
\begin{aligned}
& c\,(\kappa^{U\oplus W}\, v)\, w\, u \\
&= \mathcal{I}(\alpha^{-1}_{\Gamma,\Delta,\Theta})\kappa^{U\oplus W}\lfloor (u, w)\rfloor\, v && (c \triangleq) \\
&= \mathcal{I}(\alpha^{-1}_{\Gamma,\Delta,\Theta})(\mathcal{I}(\iota_2^{\Gamma,\Delta+\Theta})\, v) && (\kappa \triangleq) \\
&= \mathcal{I}(\alpha^{-1}_{\Gamma,\Delta,\Theta} \circ \iota_2^{\Gamma,\Delta+\Theta})\, v && \text{(functoriality)} \\
&= \mathcal{I}(\iota_2^{\Gamma+\Delta,\Theta} \circ \iota_2^{\Gamma,\Delta})\, v && \text{(cocartesian coherence)} \\
&= \mathcal{I}(\iota_2^{\Gamma+\Delta,\Theta})((\mathcal{I}(\iota_2^{\Gamma,\Delta})\, v)) && \text{(functoriality)} \\
&= \kappa^U\lfloor u\rfloor(\kappa^W\lfloor u\rfloor\, v) \\
&= (\kappa^U)^W\, (\kappa^W\, v)\, w\, u
\end{aligned}
$$

where we use the fact that λ and α are defined in terms of copairing in cocartesian categories, and therefore satisfy the diagrams by universality of coproducts:





For the axiom concerning d and i, take $l \in (X^E)(\Gamma)$ and $\omega \in {}^\Gamma(\mathcal{Y}^E)_\Delta$, and calculate:

$$
\begin{aligned}
& i\,(d\{\omega\}\, l) \\
&= (X \ominus \mathcal{Y})(\rho_\Delta)(d\{\omega\}\lfloor *\rfloor\, l) && (i \triangleq) \\
&= (X \ominus \mathcal{Y})(\rho_\Delta)(\Gamma + [], l\, *, \omega\lfloor *\rfloor \ltimes \iota_1^{\Delta,[]}) && (d \triangleq) \\
&= (\Gamma + [], l\, *, \mathcal{Y}(\rho_\Delta) \circ (\omega\lfloor *\rfloor \ltimes \iota_1^{\Delta,[]})) && \text{(module action for } \ominus) \\
&= (\Gamma + [], l\, *, (\mathcal{Y}(\rho_\Delta) \circ \omega\lfloor *\rfloor) \ltimes (\rho_\Delta \circ \iota_1^{\Delta,[]})) && \text{(widening property)} \\
&= (\Gamma + [], l\, *, \mathcal{Y}(\rho_\Delta) \circ \omega\lfloor *\rfloor \circ \rho_\Gamma) \\
&= (\Gamma, X(\rho_\Gamma)(l\, *), \mathcal{Y}(\rho_\Delta) \circ \omega\lfloor *\rfloor) && \text{(discrete multilinearity)} \\
&= (\Gamma, i\, l, i \circ \omega) && (i \triangleq)
\end{aligned}
$$

where the unlabelled equality is proved by case analysis, using the initial map $¡_\Gamma \colon [] \to \Gamma$:

$$\begin{aligned}
&((\mathcal{Y}(\rho_\Delta) \circ \omega\lfloor * \rfloor) \ltimes (\rho_\Delta \circ \iota_1^{\Delta,[]})) \circ \iota_2^{\Gamma,[]} \\
&= \mathcal{Y}(\rho_\Delta) \circ \omega\lfloor * \rfloor \\
&= \mathcal{Y}(\rho_\Delta) \circ \omega\lfloor * \rfloor \circ \left[\mathrm{id},\ ¡_\Gamma\right]_\Gamma^{\Gamma,[]} \circ \iota_2^{\Gamma,[]} \\
&= \mathcal{Y}(\rho_\Delta) \circ \omega\lfloor * \rfloor \circ \rho_\Gamma \circ \iota_2^{\Gamma,[]}
\end{aligned}
\qquad
\begin{aligned}
&((\mathcal{Y}(\rho_\Delta) \circ \omega\lfloor * \rfloor) \ltimes (\rho_\Delta \circ \iota_1^{\Delta,[]})) \circ \iota_1^{\Gamma,[]} \\
&= \rho_\Delta \circ \iota_1^{\Delta,[]} \\
&= ¡_\Delta \\
&= \mathcal{Y}(\rho_\Delta) \circ \omega\lfloor * \rfloor \circ \rho_\Gamma \circ \iota_1^{\Gamma,[]}
\end{aligned}$$

Finally, for the axiom concerning d and c, take $l \in (X^{U \otimes W})(\Gamma)$, $\omega \in {}^{\Gamma}(\mathcal{Y}^{U \otimes W})_\Delta$, $w \in W(\Theta)$ and $u \in U(\Xi)$.

$$\begin{aligned}
&d\,(d\{c \circ \omega\}\lfloor w \rfloor (c\,l))\,u \\
&= d\{(c \circ \omega)\lfloor w \rfloor \ltimes \iota_1^{\Delta,\Theta}\}\lfloor u \rfloor\,(c\,l\,w) && (d \triangleq) \\
&= \left(c\,l\,w\,u, ((c \circ \omega)\lfloor w \rfloor \ltimes \iota_1^{\Delta,\Theta})\lfloor u \rfloor \ltimes \iota_1^{\Delta+\Theta,\Xi}\right) && (d \triangleq) \\
&= \left(c\,l\,w\,u, ((c \circ \omega)\lfloor w \rfloor \lfloor u \rfloor \ltimes (\iota_2^{\Delta+\Theta,\Xi} \circ \iota_1^{\Delta,\Theta})) \ltimes \iota_1^{\Delta+\Theta,\Xi}\right) && (\text{Lem. } 10.3.1) \\
&= \left(X(\alpha^{-1}_{\Gamma,\Theta,\Xi})(l\,(u,w)), ((\mathcal{Y}(\alpha^{-1}_{\Delta,\Theta,\Xi}) \circ \omega\lfloor (u,w) \rfloor) \ltimes (\iota_2^{\Delta+\Theta,\Xi} \circ \iota_1^{\Delta,\Theta})) \ltimes \iota_1^{\Delta+\Theta,\Xi}\right) && (c \triangleq) \\
&= \left(l\,(u,w), (((\mathcal{Y}(\alpha^{-1}_{\Delta,\Theta,\Xi}) \circ \omega\lfloor (u,w) \rfloor) \ltimes (\iota_2^{\Delta+\Theta,\Xi} \circ \iota_1^{\Delta,\Theta})) \ltimes \iota_1^{\Delta+\Theta,\Xi}) \circ \alpha^{-1}_{\Gamma,\Theta,\Xi}\right) && (\text{discrete multilinearity}) \\
&= \left(l\,(u,w), (\mathcal{Y}(\alpha^{-1}_{\Delta,\Theta,\Xi}) \circ \omega\lfloor (u,w) \rfloor) \ltimes (\iota_1^{\Delta,\Theta} + \Xi)\right) \\
&= \left(l\,(u,w), (\mathcal{Y}(\alpha^{-1}_{\Delta,\Theta,\Xi}) \circ \omega\lfloor (u,w) \rfloor) \ltimes (\alpha^{-1}_{\Delta,\Theta,\Xi} \circ \iota_1^{\Delta,\Theta+\Xi})\right) && (\text{cocartesian coherence}) \\
&= \left(l\,(u,w), \mathcal{Y}(\alpha^{-1}_{\Delta,\Theta,\Xi}) \circ (\omega\lfloor (u,w) \rfloor \ltimes \iota_1^{\Delta,\Theta+\Xi})\right) && (\text{widening property}) \\
&= (X \ominus \mathcal{Y})(\alpha^{-1}_{\Delta,\Theta,\Xi})\left(l\,(u,w), \omega\lfloor (u,w) \rfloor \ltimes \iota_1^{\Delta,\Theta+\Xi}\right) && (\ominus \text{ module action}) \\
&= (X \ominus \mathcal{Y})(\alpha^{-1}_{\Delta,\Theta,\Xi})(d\{\omega\}\,l\,(u,w)) && (d \triangleq) \\
&= c\,(d\{\omega\}\,l)\,w\,u && (c \triangleq)
\end{aligned}$$

The unlabelled equality is also established by three-way case analysis on $v \in \mathcal{I}_\alpha(\Gamma + (\Theta + \Xi))$, with all three cases reducing by widening reductions and cocartesian coherence. □

The following result is the main motivator for the definition of compatible strengths and powerings, and it will be very useful in equipping the term monad with powering.

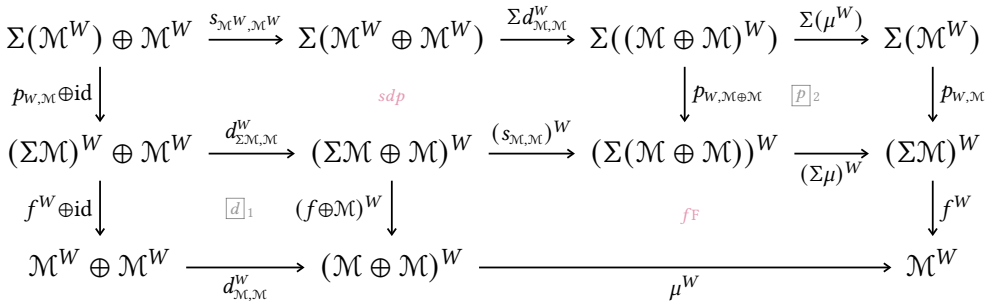
Theorem 10.3.5

If Σ has a compatible strength and powering, the category of Σ-monoids is powered over **Fam**.

Proof The powering we are looking for is the lifted Day hom:

$$(-) \multimap (=) \colon \mathbf{Fam}^{\mathrm{op}} \times \Sigma\text{-}\mathbf{Mon} \to \Sigma\text{-}\mathbf{Mon}$$

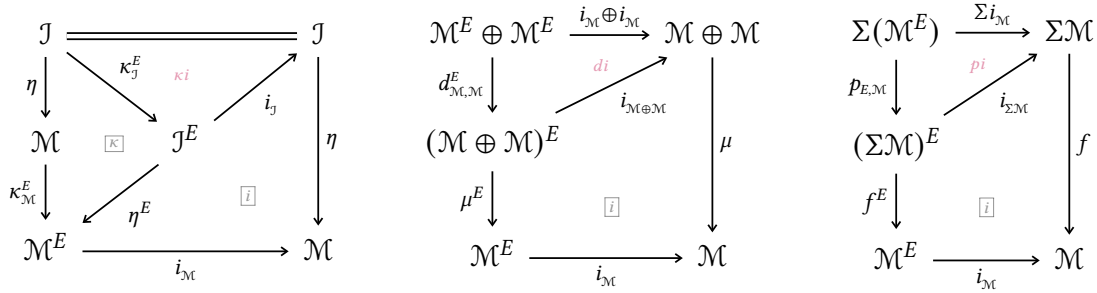
Given $W \in \mathbf{Fam}$, we have shown in Theorem 10.3.1 that $(-)^W$ is a synthetic monoidal functor on pointed modules, and therefore maps invariant (synthetic) monoids in $\mathcal{I}/\mathbf{Mod}_s$ to invariant monoids by Proposition 6.1.2. Since Σ is pointed, $(-)^W$ also lifts to Σ-algebras via the elevator $\Sigma((-)^W) \to (\Sigma -)^W$. These two structures on an object $W \multimap \mathcal{M}$ are themselves compatible thanks to the compatibility of the strength and powering of Σ:



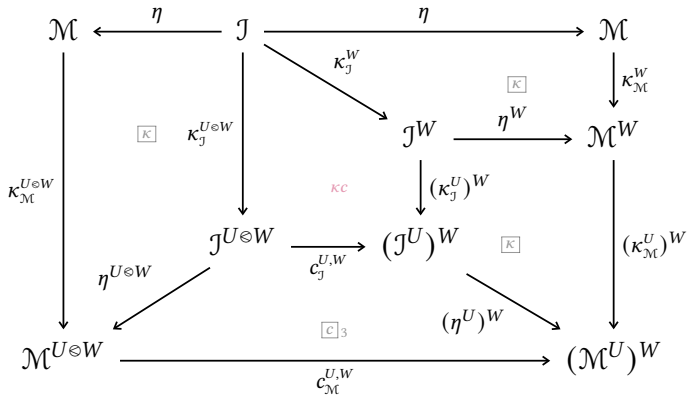
Thus, for all Σ-monoids $\mathcal{M}$, $W \multimap \mathcal{M}$ is also a Σ-monoid. To show that $\multimap$ is an action for Σ-**Mon**, we need to lift the structural transformations to be Σ-monoid homomorphisms:

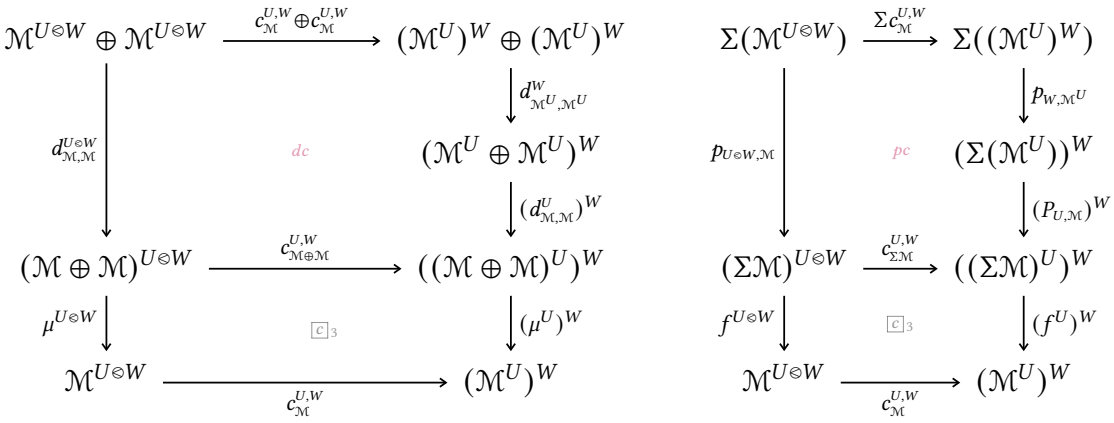
$$i_{\mathcal{M}} : (E \multimap \mathcal{M}) \to \mathcal{M} \in \Sigma\text{-}\mathbf{Mon} \qquad c^{U,W}_{\mathcal{M}} : (U \otimes W) \multimap \mathcal{M} \to (W \multimap (U \multimap \mathcal{M})) \in \Sigma\text{-}\mathbf{Mon}$$

The modular category axioms for these are the same as for the hom in families, so require no further proof. First, the Σ-monoid homomorphism condition for $i_{\mathcal{M}} : (E \multimap \mathcal{M}) \to \mathcal{M}$ consists of the unit-, multiplication-, and algebra-preservation conditions:



The Σ-monoid homomorphism conditions for $c^{U,W}_{\mathcal{M}}$ are as follows:





□

In this section we introduced the familial model as the universe of discourse for a formalisable foundation for second-order abstract syntax. Concepts from previous chapters were instantiated to equip the category of families with a renaming, substitution and convolution structure, and used synthetic monoidal categories to formalise the right notion of pointed strength for syntax endofunctors, and show that algebraic monoids form a powered category over families. We integrate these concepts next in the process of establishing the initiality and freeness theorems for second-order abstract syntax.

CHAPTER 11

Abstract syntax

This chapter applies the abstract mathematical development so far to establish the central claim of this thesis: a second-order signature freely generates a syntactic family of terms supporting substitution, interpretation in models, and metasubstitution. In Section 11.1, we present a high-level overview of the freeness and initiality results, relating the presheaf and familial models. Section 11.2 introduces the *freeness theorem*, identifying the initial syntactic algebra as the free semantic model over a family of metavariables. Building on this, Section 11.3 enriches the term monad with metasubstitution structure via the linear-clone framework from Chapter 4, and demonstrates how this supports an equational logic derived from an equational presentation of the syntax.

11.1 Second-order abstract syntax and its models

The construction of syntax starts from a second-order signature, which lists the constants and operators that the syntax must support, together with their type signatures and binding specifications. The signature induces an endofunctor on $\mathbf{Fam}_S$, algebras for which are sorted families that support the operators of the signature.

11.1.1 Signatures

The notion of an *algebraic signature* is a concise and presentation-independent way to specify first-order algebraic structures, and *binding signatures* are their second-order generalisations, allowing for terms that bind variables in their subterms. To capture simply-typed signatures, we use the notion of a sorted second-order signature given by Fiore and Hur (2010), generalising the untyped binding signatures of Aczel (1978).

Definition 11.1.1 A *second-order signature* $\Sigma = (S, O, |-|)$ is specified by a set of sorts S, a set of operators O, and an arity function $|-| : \left(S^* \times S\right)^* \times S$. ⌟

Notation. For an operator o, the tuple $|\mathsf{o}| = \left(\left[(\overrightarrow{\alpha_1}, \beta_1), (\overrightarrow{\alpha_2}, \beta_2), \ldots, (\overrightarrow{\alpha_n}, \beta_n)\right], \tau\right)$ will be denoted $\mathsf{o} : [\overrightarrow{\alpha_1}]\beta_1, \ldots, [\overrightarrow{\alpha_n}]\beta_n \longrightarrow \tau$, with the bound variable list omitted if empty. ⌟

Example 11.1.1. A first-order algebraic signature is a special case of a second-order signature, without any binding. For example, a *group* has the set of sorts $S = \{*\}$ and operators

$$\mathsf{unit} : * \qquad \mathsf{mult} : *, * \to * \qquad \mathsf{inv} : * \to *$$

The multi-sorted signature of left group actions has the two sorts $S = \{G, A\}$ and operators

$$\mathsf{unit} : G \qquad \mathsf{mult} : G, G \to G \qquad \mathsf{inv} : G \to G \qquad \mathsf{act} : G, A \to A$$

⌟

Example 11.1.2. The second-order signature of the simply-typed lambda calculus Σ_λ with let-bindings has the set of sorts S inductively defined from a set of base types B with the rules

$$\frac{\tau \in B}{\beta \in S} \qquad \frac{\alpha \in S \qquad \beta \in S}{\alpha \to \beta \in S}$$

and has the following families of operators with arities for all $\alpha, \beta \in S$:

$$\mathsf{app}_{\alpha,\beta} : (\alpha \to \beta), \alpha \to \beta \qquad \mathsf{lam}_{\alpha,\beta} : [\alpha]\beta \to (\alpha \to \beta) \qquad \mathsf{let}_{\alpha,\beta} : \alpha, [\alpha]\beta \to \beta$$

⌟

Just like how every algebraic signature determines an endofunctor on $\mathbf{Set}$ (or $\mathbf{Set}^S$) that maps a set X to a sum-of-products form of the operators, a second-order signature also determines an endofunctor on $\mathbf{Fam}_S$ that further incorporates context extension for binding terms.

Definition 11.1.2 The *second-order signature endofunctor* $\Sigma : \mathbf{Fam}_S \to \mathbf{Fam}_S$ associated with a second-order signature $\Sigma = (S, O, |-|)$ is defined as follows:

$$\Sigma(\mathcal{X})_\tau = \coprod_{\mathsf{o} : [\overrightarrow{\alpha_1}]\beta_1, \ldots, [\overrightarrow{\alpha_n}]\beta_n \to \tau} \prod_{1 \le i \le n} \delta_{[\overrightarrow{\alpha_i}]}(\mathcal{X})_{\beta_i}$$

⌟

We will usually conflate signatures and endofunctors for signatures, calling both Σ as above.

Definition 11.1.3 A *(first-order) Σ-algebra* for a signature Σ is an algebra for the $\mathbf{Fam}_S$ endofunctor $(\mathcal{I} + \Sigma)$. That is, it is a family $\mathcal{A} \in \mathbf{Fam}_S$ with a variable embedding $v : \mathcal{I} \to \mathcal{A}$ and algebra structure $a : \Sigma\mathcal{A} \to \mathcal{A}$ comprising, for each operator $\mathsf{o} : [\overrightarrow{\alpha_1}]\beta_1, \ldots, [\overrightarrow{\alpha_n}]\beta_n \to \tau$, a copairing of families of functions

$$o : \delta_{[\overrightarrow{\alpha_1}]}(\mathcal{X})_{\beta_1} \times \cdots \times \delta_{[\overrightarrow{\alpha_n}]}(\mathcal{X})_{\beta_n} \to \mathcal{X}_\tau$$

⌟

Definition 11.1.4 A *(second-order) $(\Sigma, \mathfrak{A})$-algebra* for a signature Σ and a sorted family $\mathfrak{A} \in \mathbf{Fam}_S$ is an algebra for the $\mathbf{Fam}_S$ endofunctor $(\mathcal{I} + \Sigma + \mathfrak{A}\oplus)$. That is, it is a first-order Σ-algebra $(\mathcal{A}, v, a)$ with a *metavariable embedding* $m : \mathfrak{A} \oplus \mathcal{A} \to \mathcal{A}$ that associates elements $\mathfrak{m} \in \mathfrak{A}_\tau\Pi$ of the metavariable family $\mathfrak{A}$ with a *metavariable environment* $\varepsilon \in {}^{\Pi}\mathcal{A}_\Gamma$ to produce an element $m(\Pi, \mathfrak{m}, \varepsilon) \in \mathcal{A}_\tau\Gamma$ that we will usually denote $\mathfrak{m}\{\varepsilon\}$. ⌟

The main freeness theorem we will prove in the next section can now be stated as follows:

Theorem 11.1.1

> *Given a second-order signature Σ and metavariable family $\mathfrak{A}$, the initial $(\Sigma, \mathfrak{A})$-algebra carries the structure of a free Σ-monoid on $\mathfrak{A}$.*

11.1.2 Presheaf and familial models

Theorem 11.1.1 gives a constructive method for calculating free models of signatures: the initial $(\Sigma, \mathfrak{A})$-algebra can be represented as an inductive data type with constructors for variables, operations, and metavariables. The resulting free Σ-monoid $\mathbb{T}\mathfrak{A}$ is a type- and context-indexed family of terms equipped with a substitution operation that respects the syntactic structure. A corresponding result appears as Theorem 2 in Fiore (2008), formulated in the presheaf setting: given a $\mathbf{PSh}_S$-endofunctor Ω and a metavariable presheaf $\mathcal{P}$, the monoid $\mathbf{T}\mathcal{P}$ is the free Ω-monoid on $\mathcal{P}$. Although Theorem 10.2.2 establishes an equivalence between Σ-monoids in $\mathbf{Fam}_S$ and Ω-monoids in $\mathbf{PSh}_S$ (under compatible strong signatures), it does not directly relate $\mathbb{T}\mathfrak{A}$ and $\mathbf{T}\mathcal{P}$. In fact, the initial $(\Omega, \mathcal{P})$-algebra $\mathbf{T}\mathcal{P}$ includes a metavariable constructor $\mathbf{m} \colon \mathcal{P} \otimes \mathbf{T}\mathcal{P} \longrightarrow \mathbf{T}\mathcal{P}$ that is both natural and dinatural, and this induces identifications between terms that remain distinct in $\mathbb{T}\mathfrak{A}$, due to quotienting by the dinaturality condition in the presheaf setting.

A closer examination of metavariables reveals a key simplification in the presheaf setting that helps bridge the gap between $\mathbb{T}\mathfrak{A}$ and $\mathbf{T}\mathcal{P}$. As observed by Hamana (2004) and extended by Fiore (2008), metavariables, in their intended use, are rigid symbols of fixed sort and context – not entities that should vary under term-level renaming. This means they do not naturally form a presheaf. Instead, the presheaf of metavariables is generated from a family in $\mathbf{Fam}_S$ via the free presheaf construction: given a family $\mathfrak{A}$, the presheaf of terms with metavariables in $\mathfrak{A}$ is $\mathbf{T}(\blacklozenge\mathfrak{A})$. As we show in this section, this presheaf has a simpler structure than $\mathbf{T}\mathcal{P}$ for arbitrary $\mathcal{P}$, since the quotienting induced by the metavariable operator $\mathbf{m}$ is absorbed by the free presheaf structure on $\blacklozenge\mathfrak{A}$. In fact, the underlying family of $\mathbf{T}(\blacklozenge\mathfrak{A})$ is precisely $\mathbb{T}\mathfrak{A}$, and we can lift the latter to the former.

To fix notation for the rest of the section, let $\Omega \colon \mathbf{PSh}_S \longrightarrow \mathbf{PSh}_S$ be the signature endofunctor on presheaves and $\mathbf{T} \colon \mathbf{PSh}_S \longrightarrow \mathbf{PSh}_S$ be the free Ω-monoid functor, satisfying the isomorphism

$$[\mathbf{v}, \mathbf{a}, \mathbf{m}] \;:\; \mathbf{T}\mathcal{P} \;\cong\; \mathcal{V} \;+\; \Omega(\mathbf{T}\mathcal{P}) \;+\; \mathcal{P} \otimes \mathbf{T}\mathcal{P}$$

Similarly, let $\Sigma \colon \mathbf{Fam}_S \longrightarrow \mathbf{Fam}_S$ be the signature endofunctor on families, and $\mathbb{T}\mathfrak{A}$ the initial $(\Sigma, \mathfrak{A})$-algebra satisfying the isomorphism

$$[\mathbb{v}, \mathbb{a}, \mathbb{m}] \;:\; \mathbb{T}\mathfrak{A} \;\cong\; \mathcal{I} \;+\; \Sigma(\mathbb{T}\mathfrak{A}) \;+\; \mathfrak{A} \oplus \mathbb{T}\mathfrak{A}$$

First, we relate presheaf and family algebras, assuming the signatures lift.

Proposition 11.1.1 *If* $\star\Omega \cong \Sigma\star$ *then* $\star\mathbf{T}\blacklozenge \cong \mathbb{T}$.

Proof Setting $\mathcal{P} \triangleq \blacklozenge\mathfrak{A}$ for a family $\mathfrak{A}$, the underlying family of the presheaf $\mathbf{T}(\blacklozenge\mathfrak{A})$ satisfies the following using the definition $\mathcal{I} = \star\mathcal{V}$, the lifting isomorphism $\star\Omega \cong \Sigma\star$, and Lemma 9.3.1:

$$\begin{aligned} \star\mathbf{T}(\blacklozenge\mathfrak{A}) \;&\cong\; \star\mathcal{V} \;+\; \star\Omega\big(\mathbf{T}(\blacklozenge\mathfrak{A})\big) \;+\; \star((\blacklozenge\mathfrak{A}) \otimes \mathbf{T}(\blacklozenge\mathfrak{A})) \\ &\cong\; \mathcal{I} \;+\; \Sigma(\star\mathbf{T}(\blacklozenge\mathfrak{A})) \;+\; \mathfrak{A} \oplus \star\mathbf{T}(\blacklozenge\mathfrak{A}) \end{aligned}$$

Thus, $\star\mathbf{T}(\blacklozenge\mathfrak{A})$ satisfies the same isomorphism as $\mathbb{T}\mathcal{X}$, so the two families are isomorphic. □

A more valuable result is that the construction goes the other way around too: the initial presheaf algebra on free presheaves can be induced purely from the initial family.

Proposition 11.1.2 *If* $\star\Omega \cong \Sigma\star$ *then* $\mathbb{T} \colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ *extends to a functor* $\widehat{\mathbb{T}} \colon \mathbf{Fam}_S \to \mathbf{PSh}_S$ *with* $\star\widehat{\mathbb{T}} = \mathbb{T}$ *and* $\widehat{\mathbb{T}}\mathfrak{A}$ *the initial* $(\Omega, \blacklozenge\mathfrak{A})$*-algebra.*

Proof This is an instance of the initial algebra-lifting theorem (Theorem 3.3.1), instantiated with the lifting constructed in the previous section:

$$\begin{array}{ccc} \mathbf{PSh}_S & \xrightarrow{\mathcal{V} + \Omega(-) + \blacklozenge\mathfrak{A}\otimes(-)} & \mathbf{PSh}_S \\ {\scriptstyle\star}\downarrow & & \downarrow{\scriptstyle\star} \\ \mathbf{Fam}_S & \xrightarrow[\mathcal{I} + \Sigma(-) + \mathfrak{A}\oplus(-)]{} & \mathbf{Fam}_S \end{array}$$

that lifts the initial algebra $\mathbb{T}\mathfrak{A}$ to an initial $(\Omega, \blacklozenge\mathfrak{A})$-algebra.

It is worth expanding on the abstract proof, as it uncovers the elegant interaction between initiality and lifting. The presheaf $\widehat{\mathbb{T}}\mathfrak{A}$ is equivalently given by the underlying family $\star\widehat{\mathbb{T}} = \mathbb{T}$ and a module structure on $\mathbb{T}\mathfrak{A}$. The $\Box$-coalgebra structure $\mathbb{T}\mathfrak{A} \to \Box\mathbb{T}\mathfrak{A}$ can be induced by initiality, by equipping $\Box\mathbb{T}\mathfrak{A}$ with the structure of a $(\Sigma, \mathfrak{A})$-algebra:

$$\begin{array}{rcl} \mathcal{I} \xrightarrow{j_{\mathcal{I}}} \Box\mathcal{I} & \xrightarrow{\Box\mathbb{v}} & \Box\mathbb{T}\mathfrak{A} \\ \Sigma\Box\mathbb{T}\mathfrak{A} \xrightarrow{s^{\Sigma}_{\mathcal{I},\mathbb{T}\mathfrak{A}}} \Box\Sigma\mathbb{T}\mathfrak{A} & \xrightarrow{\Box\mathbb{a}} & \Box\mathbb{T}\mathfrak{A} \\ \mathfrak{A} \oplus \Box\mathbb{T}\mathfrak{A} \xrightarrow{s^{\Box}_{\mathfrak{A},\mathbb{T}\mathfrak{A}}} \Box(\mathfrak{A} \oplus \mathbb{T}\mathfrak{A}) & \xrightarrow{\Box\mathbb{m}} & \Box\mathbb{T}\mathfrak{A} \end{array}$$

where $\Sigma\Box \Longrightarrow \Box\Sigma$ is the elevator induced by the pointed strength $\Sigma[\![\mathcal{I}, -]\!] \to [\![\mathcal{I}, \Sigma(-)]\!]$, and $\mathfrak{A} \oplus \Box\mathcal{X} \to \Box(\mathfrak{A} \oplus \mathcal{X})$ is the transpose of the composite

$$\Diamond(\mathfrak{A} \oplus \Box\mathcal{X}) \to \mathfrak{A} \oplus \Diamond\Box\mathcal{X} \xrightarrow{\mathfrak{A}\oplus\varepsilon_{\mathcal{X}}} \mathfrak{A} \oplus \mathcal{X}$$

$\mathbb{T}\mathfrak{A}$ satisfies the equation of an initial $(\Omega, \blacklozenge\mathfrak{A})$-algebra by the previous result. □

This lifting theorem concerns the full syntactic algebras that incorporates variables, terms, and metavariables. Given two endofunctors $\Sigma \colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ and $\Omega \colon \mathbf{PSh}_S \to \mathbf{PSh}_S$ generated from the same second-order signature, we also know that the lifting isomorphism $\star\Omega \cong \Sigma\star$ is satisfied: $\star$ is cartesian and cocartesian, and $\star(\mathcal{V}_\tau \supset \mathcal{P}) \cong \mathcal{N}_\tau \multimap \star\mathcal{P}$ by Corollary 10.3.1.

In summary, starting from a second-order signature, a signature endofunctor on families and its lifting to presheaves is induced; Theorem 10.2.2 establishes the equivalence of models $\Sigma\mathbf{Mon}(\mathbf{Fam}_S) \simeq \Omega\mathbf{Mon}(\mathbf{PSh}_S)$, and Proposition 11.1.2 relates the initial $(\Sigma, \mathfrak{A})$-algebra with the initial $(\Omega, \blacklozenge\mathfrak{A})$-algebra. Fiore (2008, Theorem 2) connects the presheaf side by showing that the initial $(\Omega, \mathcal{P})$-algebra is the free Ω-monoid on $\mathcal{P}$, and we fill in the missing side by showing that the initial $(\Sigma, \mathfrak{A})$-algebra is the free Σ-monoid on $\mathfrak{A}$.

11.2 Metatheory by initiality

The presheaf model offers an abstract, algebraic framework for syntactic metatheory. Rather than defining capture-avoiding substitution manually, Fiore et al. (1999) showed it can be derived automatically from the syntax via initial algebra semantics, a standard tool in programming language theory. While classical approaches like Goguen et al. (1976) model data types as initial algebras for **Set**-endofunctors, shifting to the category of presheaves – variable sets – allows us to define the datatype of context- and type-indexed terms $\mathbf{T}$ as an initial algebra for a $\mathbf{PSh}_S$-endofunctor. This supports initial interpretations into first-order Σ-algebras $\mathbf{T} \to \mathcal{A}$ that preserve context and type. With an appropriate choice of $\mathcal{A}$, such interpretations can implement syntactic operations like substitution or, for second-order signatures, metasubstitution. The uniqueness of initial algebra semantics yields a robust proof principle for reasoning about these operations.

In this section, we adapt this approach to the category of sorted families to prove Theorem 11.1.1, connecting the initial $(\Sigma, \mathfrak{N})$-algebra to the free Σ-monoid on $\mathfrak{N}$. Our method diverges from that of Fiore (2008), whose proof – spelled out by Arkor (2018) and Borthelle et al. (2020) – relies on a monoidal structure that does not extend to our skew-monoidal setting. Their technique, a variant of initial algebra semantics known as *parametrised initiality*, was formalised by Fiore and Saville (2017, Section 4) and abstractly by Matthes and Uustalu (2004). If $K \colon \mathcal{C} \to \mathcal{C}$ is a left adjoint and $(K, \varphi \colon KF \Longrightarrow GK)$ forms a coelevator between $F, G \colon \mathcal{C} \to \mathcal{C}$, given an initial F-algebra $f \colon FA \to A$ and any G-algebra $g \colon GB \to B$ there exists a unique morphism $h \colon KA \to B$ satisfying

$$\begin{array}{ccccc} KFA & \xrightarrow{\varphi_A} & GKA & \xrightarrow{Gh} & GB \\ {\scriptstyle Kf}\downarrow & & & & \downarrow{\scriptstyle g} \\ KA & & \xrightarrow[h]{} & & B \end{array}$$

Instantiating this appropriately, we obtain the standard parametrised initiality result used in the presheaf model of abstract syntax: for all signature endofunctors $\Omega \colon \mathbf{PSh} \to \mathbf{PSh}$, pointed presheaves $\mathcal{Q} \in \mathcal{V}/\mathbf{PSh}_S$ and $(\mathcal{Q} + \Omega + \mathcal{P}\otimes)$-algebras $(\mathcal{A}, [v, a, m])$, there exists a unique parametrised initial algebra interpretation $h^{\mathcal{Q}}_{\mathcal{A}} \colon \mathbf{T}\mathcal{P} \otimes \mathcal{Q} \to \mathcal{A}$ satisfying:

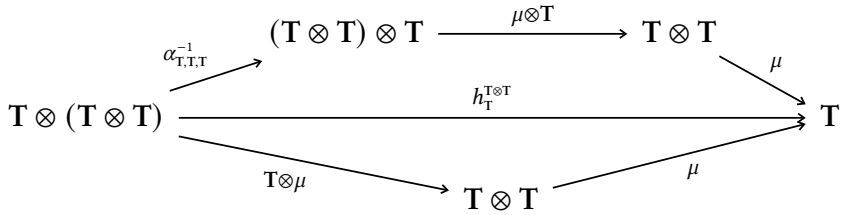
$$\begin{array}{ccc} \mathcal{V} \otimes \mathcal{Q} & \xrightarrow{\lambda_{\mathcal{Q}}} & \mathcal{Q} \\ {\scriptstyle \mathbf{v}\otimes\mathcal{Q}}\downarrow & & \downarrow{\scriptstyle v} \\ \mathbf{T}\mathcal{P} \otimes \mathcal{Q} & \xrightarrow[h^{\mathcal{Q}}_{\mathcal{A}}]{} & \mathcal{A} \end{array} \qquad \begin{array}{ccccc} \Omega(\mathbf{T}\mathcal{P}) \otimes \mathcal{Q} & \xrightarrow{s^{\Omega}_{\mathbf{T}\mathcal{P},\mathcal{Q}}} & \Omega(\mathbf{T}\mathcal{P} \otimes \mathcal{Q}) & \xrightarrow{\Omega h^{\mathcal{Q}}_{\mathcal{A}}} & \Omega\mathcal{A} \\ {\scriptstyle \mathbf{a}\otimes\mathcal{Q}}\downarrow & & & & \downarrow{\scriptstyle a} \\ \mathbf{T}\mathcal{P} \otimes \mathcal{Q} & & \xrightarrow[h^{\mathcal{Q}}_{\mathcal{A}}]{} & & \mathcal{A} \end{array}$$

$$\begin{array}{ccccc} (\mathcal{P} \otimes \mathbf{T}\mathcal{P}) \otimes \mathcal{Q} & \xrightarrow{\alpha_{\mathcal{P},\mathbf{T}\mathcal{P},\mathcal{Q}}} & \mathcal{P} \otimes (\mathbf{T}\mathcal{P} \otimes \mathcal{Q}) & \xrightarrow{\mathcal{P}\otimes h^{\mathcal{Q}}_{\mathcal{A}}} & \mathcal{P} \otimes \mathcal{A} \\ {\scriptstyle \mathbf{m}\otimes\mathcal{Q}}\downarrow & & & & \downarrow{\scriptstyle m} \\ \mathbf{T}\mathcal{P} \otimes \mathcal{Q} & & \xrightarrow[h^{\mathcal{Q}}_{\mathcal{A}}]{} & & \mathcal{A} \end{array}$$

With different choices of $\mathcal{Q}$ and $\mathcal{A}$, different parametrised syntactic operations may be instantiated, and given correctness laws of the form $f = g \colon \mathbf{T}\mathcal{P}\otimes\mathcal{Q} \to \mathcal{A}$, the equality can be indirectly

established by showing that f and g are parametrised algebra homomorphisms in the above sense. The technique can also be adapted to cartesian parametrised maps $\mathbf{T} \times \mathcal{Q} \to \mathcal{Z}$, as is done in the derivation of metasubstitution structure by Fiore (2008).

While the notion of parametrised initiality can be translated to families, the skew-monoidal setting introduces a key limitation: in a parametrised map $h\colon \mathbb{T}\mathfrak{A} \oplus \mathcal{X} \to \mathcal{A}$, variation is restricted to $\mathcal{X}$, requiring $\mathbb{T}\mathfrak{A}$ to appear as the left operand of the tensor. This imposes a right-biased structure on the domain, constraining the form of expressible laws. For example, proving associativity for the substitution monoid on $\mathbb{T}$ (with $\mathfrak{A}$ implicit) would require a domain of the form $\mathbb{T} \oplus (\mathbb{T} \oplus \mathbb{T})$, but this cannot be rearranged to $(\mathbb{T} \oplus \mathbb{T}) \oplus \mathbb{T}$, since the associator in a skew-monoidal category is not invertible. In contrast, the presheaf model allows such rearrangements, enabling the associativity law to be expressed as an equality between $(\mathbf{T} \otimes \mathbf{T})$-parametrised maps:



One resolution is to use the generalised parametrised initiality and allow for a sequence of left-biased parameters: for example, a double-parametrised map $(\mathbb{T} \oplus \mathcal{X}) \oplus \mathcal{Y} \to \mathcal{A}$ would be an instance of h with $K \triangleq (- \oplus \mathcal{X}) \oplus \mathcal{Y}$, which is left adjoint to $[\![\mathcal{X}, [\![\mathcal{Y}, -]\!]]\!]$. More generally, using the notation of Section 5.2.3, an n-parametrised initial map $h\colon (\mathbb{T} \oplus \mathcal{Y}_i)^n \to \mathcal{A}$ satisfies the homomorphism conditions

$$\begin{array}{ccc} (\mathcal{I} \oplus \mathcal{Y}_i)^n & \xrightarrow{(\lambda_{\mathcal{Y}_1} \oplus \mathcal{Y}_{2\le i})^n} & (\mathcal{Y}_i)^n \\ {\scriptstyle (\mathbb{v} \oplus \mathcal{Y}_i)^n}\downarrow & & \downarrow{\scriptstyle v} \\ (\mathbb{T} \oplus \mathcal{Y}_i)^n & \xrightarrow[h_{\mathcal{A}}^{\mathcal{Y}_i}]{} & \mathcal{A} \end{array}$$

$$\begin{array}{ccccccc} (\Sigma\mathbb{T} \oplus \mathcal{Y}_i)^n & \xrightarrow{(s^{\Sigma}_{\mathbb{T},\mathcal{Y}_1} \oplus \mathcal{Y}_{2\le i})^n} & (\Sigma(\mathbb{T} \oplus \mathcal{Y}_1) \oplus \mathcal{Y}_{2\le i})^n & \cdots & (\Sigma(\mathbb{T} \oplus \mathcal{Y}_{j\le k})^k \oplus \mathcal{Y}_{k<i})^n & \cdots & \Sigma(\mathbb{T} \oplus \mathcal{Y}_i)^n \\ & & & & & & \downarrow{\scriptstyle \Sigma h_{\mathcal{A}}^{\mathcal{Y}_i}} \\ {\scriptstyle (\mathbb{a} \oplus \mathcal{Y}_i)^n}\downarrow & & & & & & \Sigma\mathcal{A} \\ & & & & & & \downarrow{\scriptstyle a} \\ (\mathbb{T} \oplus \mathcal{Y}_i)^n & & & & \xrightarrow[h_{\mathcal{A}}^{\mathcal{Y}_i}]{} & & \mathcal{A} \end{array}$$

$$\begin{array}{ccccccc} ((\mathfrak{A} \oplus \mathbb{T}) \oplus \mathcal{Y}_i)^n & \xrightarrow{(\alpha_{\mathfrak{A},\mathbb{T},\mathcal{Y}_i} \oplus \mathcal{Y}_{2\le i})^n} & ((\mathfrak{A} \oplus (\mathbb{T} \oplus \mathcal{Y}_1)) \oplus \mathcal{Y}_{2\le i})^n & \cdots & ((\mathfrak{A} \oplus (\mathbb{T} \oplus \mathcal{Y}_{j\le k})^k) \oplus \mathcal{Y}_{k<i})^n & \cdots & \mathfrak{A} \oplus (\mathbb{T} \oplus \mathcal{Y}_i)^n \\ & & & & & & \downarrow{\scriptstyle \mathfrak{A} \oplus h_{\mathcal{A}}^{\mathcal{Y}_i}} \\ {\scriptstyle (\mathbb{m} \oplus \mathcal{Y}_i)^n}\downarrow & & & & & & \mathfrak{A} \oplus \mathcal{A} \\ & & & & & & \downarrow{\scriptstyle m} \\ (\mathbb{T} \oplus \mathcal{Y}_i)^n & & & & \xrightarrow[h_{\mathcal{A}}^{\mathcal{Y}_i}]{} & & \mathcal{A} \end{array}$$

where the $\cdots$ abbreviate repeated nested applications of the strength and associator. By instantiating K with convolutional parameters, we can derive initiality results for maps of the form $\mathbb{T} \circledcirc W \to \mathcal{X}$; the multi-ary generalisation is not needed, as $\circledcirc$ is strongly associative.

This approach, however, remains cumbersome: we end up with a family of initiality theorems indexed by the number of parameters in the semantic morphisms, which is awkward to formalise since each case must be spelled out separately. Instead, we abandon parametrised initiality and the skew-monoidal structure in favour of the skew-closed structure, recasting each parametrised map $\mathbb{T} \oplus \mathcal{X} \to \mathcal{A}$ as a "standard" initial morphism $\mathbb{T} \to [\![\mathcal{X}, \mathcal{A}]\!]$. This shift sidesteps the one-way associator issue without weakening the theory – every construct in the monoidal presheaf model (pointed strengths, algebraic monoids, etc.) has an equivalent closed formulation. It also simplifies formalisation in a proof assistant, since closed maps translate directly to dependent functions without requiring explicit products or dependent sums. Conceptually, the closed freeness theorem yields a modular, efficient proof that replaces lengthy, repetitive diagram chases (cf. Appendix B of Borthelle et al. (2020)) with concise, compositional reasoning. The remainder of this section presents that proof.

11.2.1 Syntactic algebras

We introduce some auxiliary definitions and lemmas for efficiently equipping objects with second-order $(\Sigma, \mathfrak{A})$-algebra structure. Thus, fix a signature endofunctor $\Sigma \colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ (which, as we know from Theorem 10.3.4, comes with compatible skew pointed strength and powering over the Day hom) and a metavariable family $\mathfrak{A}$.

Notation. The functor $(\mathcal{X} + \Sigma + \mathfrak{A}\oplus)$ will be denoted $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$, and $\Sigma^{\mathcal{I}}_{\mathfrak{A}}$ will be written as $\Sigma_{\mathfrak{A}}$. We will also write $\vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}}$ for the category of $(\mathcal{X} + \Sigma + \mathfrak{A}\oplus)$-algebras and homomorphisms. The structure maps of a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra $\mathcal{A}$ will be denoted and called $v \colon \mathcal{X} \to \mathcal{A}$ for the *variable embedding*, $a \colon \Sigma\mathcal{A} \to \mathcal{A}$ for the *algebra map*, and $m \colon \mathfrak{A} \multimap \mathcal{A}$ for the *metavariable embedding*. ⌟

Lemma 11.2.1 $\vec{\Sigma}^{(=)}_{(-)} \colon \mathbf{Fam}_S^{\mathrm{op}} \times \mathbf{Fam}_S^{\mathrm{op}} \to \mathbf{Cat}$ *is functorial.*

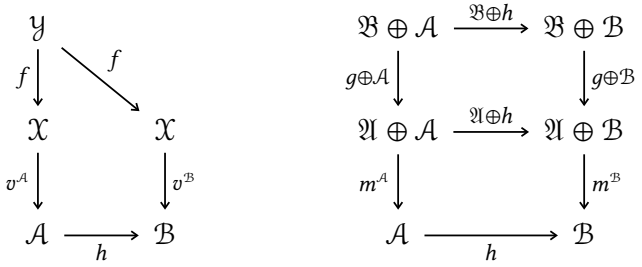
Proof By definition, $\vec{\Sigma}^{(=)}_{(-)}$ maps families $\mathfrak{A}$ and $\mathcal{X}$ to categories of algebras $(\Sigma^{\mathcal{X}}_{\mathfrak{A}}, \Sigma^{\mathcal{X}}_{\mathfrak{A}}\text{-}\mathbf{Alg} = \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}})$. Given $f \colon \mathcal{Y} \to \mathcal{X}$ and $g \colon \mathfrak{B} \to \mathfrak{A}$, the action of the $\mathbf{Cat}$-morphism (i.e. functor) $\vec{\Sigma}^{f}_{g} \colon \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}} \to \vec{\Sigma}^{\mathcal{Y}}_{\mathfrak{B}}$ on objects $(\mathcal{A}, [v, a, m])$ is given by

$$\vec{\Sigma}^{f}_{g}(\mathcal{A}, [v, a, m]) \triangleq (\mathcal{A}, [\mathcal{Y} \xrightarrow{f} \mathcal{X} \xrightarrow{v} \mathcal{A}, a, \mathfrak{B} \xrightarrow{g} \mathfrak{A} \xrightarrow{m} \mathcal{A}])$$

The action of $\vec{\Sigma}^{f}_{g}$ on morphisms $h \colon (\mathcal{A}, [v^{\mathcal{A}}, a^{\mathcal{A}}, n^{\mathcal{A}}]) \to (\mathcal{B}, [v^{\mathcal{B}}, a^{\mathcal{B}}, n^{\mathcal{B}}])$ gives a $\vec{\Sigma}^{\mathcal{Y}}_{\mathfrak{B}}$-algebra homomorphism

$$(\mathcal{A}, [v^{\mathcal{A}} \circ f, a^{\mathcal{A}}, m^{\mathcal{A}} \circ g]) \to (\mathcal{B}, [v^{\mathcal{B}} \circ f, a^{\mathcal{B}}, m^{\mathcal{B}} \circ g])$$

with $h \colon \mathcal{A} \to \mathcal{B}$ as the underlying family map, and the structure map preservation conditions reducing simply to the homomorphism conditions of h:



□

To allow for simple, compositional proofs of objects having $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra structure for different choices of $\mathcal{X}$, we use the Grothendieck construction to associate variable families $\mathcal{X}$ with their respective second-order algebra categories.

Definition 11.2.1 The category $\int \vec{\Sigma}_{\mathfrak{A}}$ is the Grothendieck construction for the functor $\vec{\Sigma}^{-}_{\mathfrak{A}} : (\mathcal{I}/\mathbf{Mod}_S)^{\mathrm{op}} \to \mathbf{Cat}$; explicitly defined as having:

- Objects: pairs $(\mathcal{X}, \mathcal{A})$ with $\mathcal{X}$ a pointed module and $\mathcal{A}$ a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra;
- Morphisms: pairs $(f, h) : (\mathcal{X}, \mathcal{A}) \to (\mathcal{Y}, \mathcal{B})$ with $f : \mathcal{Y} \to \mathcal{X}$ a pointed module homomorphism, and $h : \vec{\Sigma}^{f}_{\mathfrak{A}}\mathcal{A} \to \mathcal{B}$ a $\Sigma^{\mathcal{Q}}_{\mathfrak{A}}$-algebra homomorphism.

Explicitly, given a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra $\mathcal{A}$ and a $\Sigma^{\mathcal{Y}}_{\mathfrak{A}}$-algebra $\mathcal{B}$, $\Sigma^{f}_{\mathfrak{A}}\mathcal{A}$ is also a $\Sigma^{\mathcal{Y}}_{\mathfrak{A}}$-algebra with variable embedding $\mathcal{Y} \xrightarrow{f} \mathcal{X} \xrightarrow{v} \mathcal{A}$, and a morphism $h : \vec{\Sigma}^{f}_{\mathfrak{A}}\mathcal{A} \to \mathcal{B}$ is a $(\Sigma + \mathfrak{A}\oplus)$-algebra homomorphism that preserves the $\mathcal{Y}$-variable embedding:

$$\begin{array}{ccc} \mathcal{X} & \xleftarrow{f} & \mathcal{Y} \\ {\scriptstyle v}\downarrow & & \downarrow{\scriptstyle u} \\ \mathcal{A} & \xrightarrow[h]{} & \mathcal{B} \end{array} \tag{†}$$

⌟

The following proposition allows us to parametrise second-order algebras with a pointed module parameter, and will be the main tool we use to build second-order algebras.

Proposition 11.2.1 *For all $\mathcal{X} \in \mathcal{I}/\mathbf{Mod}_S$, every $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra $\mathcal{A}$ induces a $\Sigma^{[\![\mathcal{X},\mathcal{X}]\!]}_{\mathfrak{A}}$-algebra $[\![\mathcal{X}, \mathcal{A}]\!]$.*

Proof Let $(\mathcal{A}, [v, a, m])$ be a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra. We define the components of the $\Sigma^{[\![\mathcal{X},\mathcal{X}]\!]}_{\mathfrak{A}}$-algebra structure on $[\![\mathcal{X}, \mathcal{A}]\!]$ as follows:

$$\begin{aligned} [\![\mathcal{X}, \mathcal{X}]\!] &\xrightarrow{[\![\mathcal{X}, v]\!]} [\![\mathcal{X}, \mathcal{A}]\!] \\ \Sigma[\![\mathcal{X}, \mathcal{A}]\!] \xrightarrow{s^{\Sigma}_{\mathcal{X},\mathcal{A}}} [\![\mathcal{X}, \Sigma\mathcal{A}]\!] &\xrightarrow{[\![\mathcal{X}, a]\!]} [\![\mathcal{X}, \mathcal{A}]\!] \\ \mathfrak{A} \oplus [\![\mathcal{X}, \mathcal{A}]\!] \xrightarrow{e^{\mathfrak{A}}_{\mathcal{X},\mathcal{A}}} [\![\mathcal{X}, \mathfrak{A} \oplus \mathcal{A}]\!] &\xrightarrow{[\![\mathcal{X}, m]\!]} [\![\mathcal{X}, \mathcal{A}]\!] \end{aligned}$$

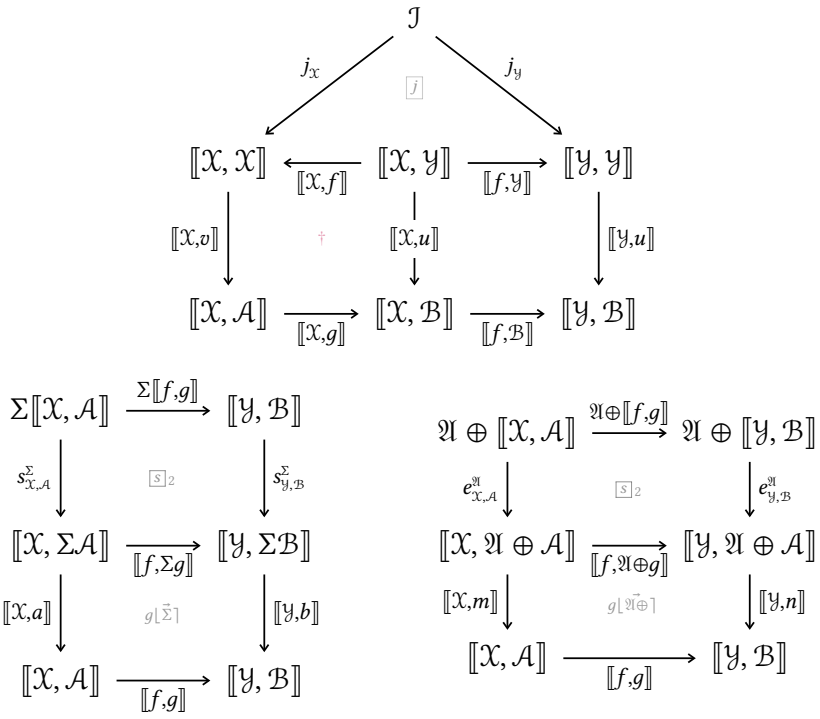
where $e^{\mathfrak{A}}_{\mathcal{X},\mathcal{A}} : \mathfrak{A} \oplus [\![\mathcal{X}, \mathcal{A}]\!] \to [\![\mathcal{X}, \mathfrak{A} \oplus \mathcal{A}]\!]$ is in bijection with the associator $(\mathfrak{A} \oplus \mathcal{X}) \oplus \mathcal{A} \to \mathfrak{A} \oplus (\mathcal{X} \oplus \mathcal{A})$ that makes $\mathfrak{A}\oplus : \mathbf{Fam}_S \to \mathbf{Fam}_S$ a right module strength (see Theorem 5.1.1). □

Proposition 11.2.2 *The internal hom $[\![-, =]\!]$ is the object mapping of a functor $\int \vec{\Sigma}_{\mathfrak{A}} \to \vec{\Sigma}_{\mathfrak{A}}$.*

Proof Given an object $(\mathcal{X}, \mathcal{A} \in \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}}) \in \int \vec{\Sigma}_{\mathfrak{A}}$ for a pointed module $\mathcal{X}$, the internal hom $[\![\mathcal{X}, \mathcal{A}]\!]$ is a $\vec{\Sigma}^{[\![\mathcal{X},\mathcal{X}]\!]}_{\mathfrak{A}}$-algebra, and precomposing the variable embedding $[\![\mathcal{X}, \mathcal{X}]\!] \xrightarrow{[\![\mathcal{X}, v]\!]} [\![\mathcal{X}, \mathcal{A}]\!]$ with the applicator map $j_{\mathcal{X}} : \mathcal{I} \to [\![\mathcal{X}, \mathcal{X}]\!]$ makes $[\![\mathcal{X}, \mathcal{A}]\!]$ a $\Sigma_{\mathfrak{A}}$-algebra. Such reasoning can be efficiently presented by the following sequence of mappings:

$$\mathcal{A} \in \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}} \xmapsto{\text{Prop. 11.2.1}} [\![\mathcal{X}, \mathcal{A}]\!] \in \vec{\Sigma}^{[\![\mathcal{X},\mathcal{X}]\!]}_{\mathfrak{A}} \xmapsto{\text{Lem. 11.2.1}} \vec{\Sigma}^{j_{\mathcal{X}}}_{\mathfrak{A}} [\![\mathcal{X}, \mathcal{A}]\!] \in \vec{\Sigma}_{\mathfrak{A}}$$

Now, given a morphism $(f, h) : (\mathcal{X}, \mathcal{A}) \to (\mathcal{Y}, \mathcal{B})$ in $\int \vec{\Sigma}_{\mathfrak{A}}$, we show that $[\![f, g]\!] : [\![\mathcal{X}, \mathcal{A}]\!] \to [\![\mathcal{Y}, \mathcal{B}]\!]$ is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism. Let $(\mathcal{A}, [v, a, m]) \in \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}}$ and $(\mathcal{B}, [u, b, n]) \in \vec{\Sigma}^{\mathcal{Y}}_{\mathfrak{A}}$ be algebras, with $h : \mathcal{A} \to \mathcal{B}$ a $(\Sigma + \mathfrak{A}\oplus)$-algebra homomorphism that commutes with u and v as in Diagram (†). We show that $[\![f, g]\!]$ preserves variables, algebras, and metavariables.



□

Corollary 11.2.1 *For all pointed $\Box$-coalgebras $\mathcal{X}$, $\Box$ lifts to an endofunctor on $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebras.*

Proof We have the following mappings to show that if $\mathcal{A}$ is a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra, so is $\Box\mathcal{A}$:

$$\mathcal{A} \in \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}} \xmapsto{\text{Prop. 11.2.1}} \Box\mathcal{A} \in \vec{\Sigma}^{\Box\mathcal{X}}_{\mathfrak{A}} \xmapsto{\text{Lem. 11.2.1}} \vec{\Sigma}^{r}_{\mathfrak{A}} \Box\mathcal{A} \in \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}}$$

Given a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra homomorphism $h : \mathcal{A} \to \mathcal{B}$, $(\mathrm{id}, h) : (\mathcal{X}, \mathcal{A}) \to (\mathcal{X}, \mathcal{B})$ is a morphism in $\int \vec{\Sigma}_{\mathfrak{A}}$ with (†) $h \circ v \circ \mathrm{id} = w$ satisfied by the variable-preservation of h, so $[\![\mathrm{id}, h]\!] = \Box h : \Box\mathcal{A} \to \Box\mathcal{B}$ is also a homomorphism of $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebras by Proposition 11.2.2. □

Remark. The corollary above demonstrates the benefit of working in the closed setting: such a lifting property does not work for $\Diamond$-algebras, as the metavariable constructor would have to be of the form that requires the inverse of the skew associator:

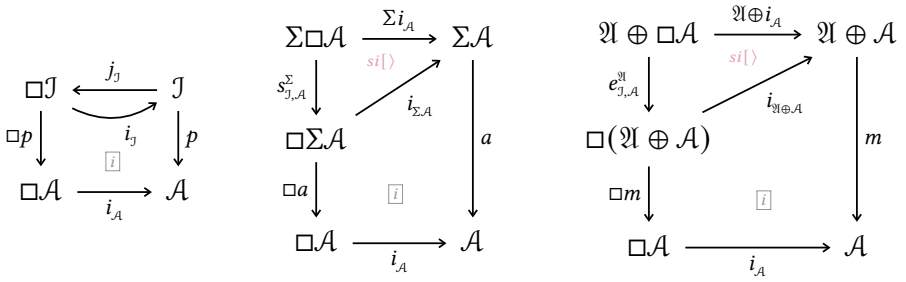
$$\mathfrak{A} \oplus \Diamond\mathcal{A} \cong \mathfrak{A} \oplus (\mathcal{A} \oplus \mathcal{I}) \xrightarrow{?} (\mathfrak{A} \oplus \mathcal{A}) \oplus \mathcal{I} \xrightarrow{m\oplus\mathcal{I}} \mathcal{A} \oplus \mathcal{I}$$

⌟

The structure maps of the closed category are syntactic homomorphisms.

Lemma 11.2.2 *For any $\Sigma_{\mathfrak{A}}$-algebra $\mathcal{A}$, $i_{\mathcal{A}} : \Box\mathcal{A} \to \mathcal{A}$ is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism.*

Proof By Corollary 11.2.1, $\Box\mathcal{A}$ is a $\Sigma_{\mathfrak{A}}$-algebra. The homomorphism conditions are:



□

Lemma 11.2.3 *For any $\Sigma^{\mathcal{Z}}_{\mathfrak{A}}$-algebra $\mathcal{A}$, pointed $\Box$-coalgebras $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ and pointed multilinear maps $f : \mathcal{X} \multimap \boxed{\mathcal{Y}} \to \mathcal{Z}$, the following composite – implementing the synthetic compositor in the synthetic closed $\mathcal{I}$/**Mod**$_S$-modular category* **Fam**$_S$ *– is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism:*

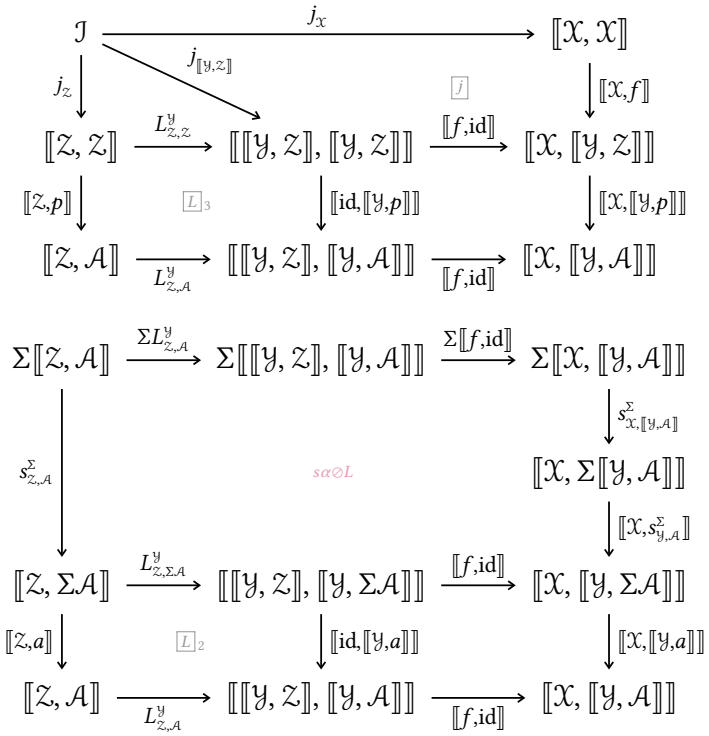
$$L[f]_{\mathcal{A}} : [\![\mathcal{Z}, \mathcal{A}]\!] \xrightarrow{L^{\mathcal{Y}}_{\mathcal{Z},\mathcal{A}}} [\![[\![\mathcal{Y}, \mathcal{Z}]\!], [\![\mathcal{Y}, \mathcal{A}]\!]]\!] \xrightarrow{[\![f,\mathrm{id}]\!]} [\![\mathcal{X}, [\![\mathcal{Y}, \mathcal{A}]\!]]\!]$$

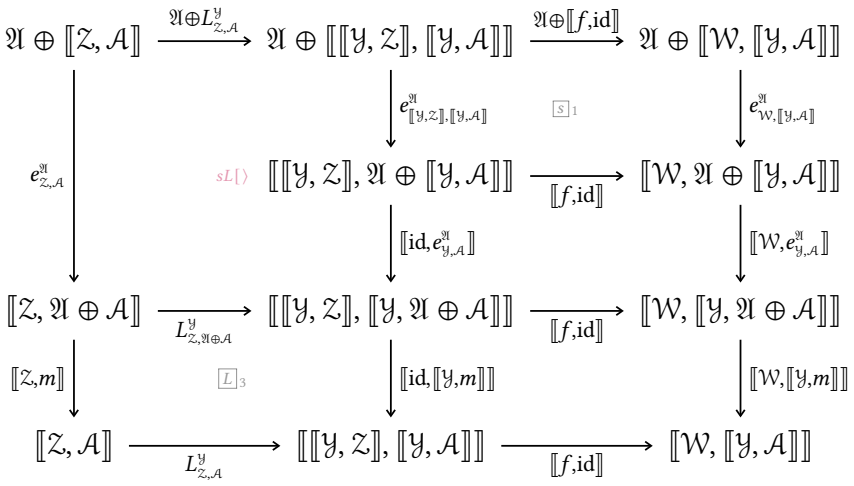
Proof First, we show that the endpoints are $\Sigma_{\mathfrak{A}}$-algebras:

$$\mathcal{A} \in \vec{\Sigma}^{\mathcal{Z}}_{\mathfrak{A}} \xmapsto{\text{Prop. 11.2.2}} [\![\mathcal{Z}, \mathcal{A}]\!] \in \vec{\Sigma}_{\mathfrak{A}}$$

$$\mathcal{A} \in \vec{\Sigma}^{\mathcal{Z}}_{\mathfrak{A}} \xmapsto{\text{Prop. 11.2.1}} [\![\mathcal{Y}, \mathcal{A}]\!] \in \vec{\Sigma}^{[\![\mathcal{Y},\mathcal{Z}]\!]}_{\mathfrak{A}} \xmapsto{\text{Lem. 11.2.1}} \vec{\Sigma}^{f}_{\mathfrak{A}}[\![\mathcal{Y}, \mathcal{A}]\!] \in \vec{\Sigma}^{\mathcal{X}}_{\mathfrak{A}} \xmapsto{\text{Prop. 11.2.2}} [\![\mathcal{X}, \vec{\Sigma}^{f}_{\mathfrak{A}}[\![\mathcal{Y}, \mathcal{A}]\!]]\!] \in \vec{\Sigma}_{\mathfrak{A}}$$

Next, we establish the homomorphism conditions:

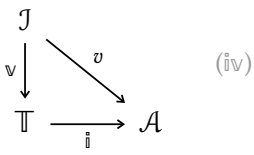




With the main lemmas about second-order algebra homomorphism conditions established, we can move on to the proof of the freeness theorem.

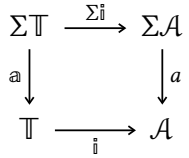
11.2.2 Free substitution structure

In this section we work through the proof of the freeness theorem: the initial $(\Sigma, \mathfrak{A})$-algebra is the free Σ-monoid on $\mathfrak{A}$. We will use initial algebra semantics, using the lemmas from the previous section. To fix notation, we will write $(\mathbb{T}\mathfrak{A}, [\mathbb{v}, \mathbb{a}, \mathbb{m}])$ for the initial $(\Sigma, \mathfrak{A})$-algebra, keeping $\mathfrak{A}$ implicit in the first part of the proof. For a $\Sigma_{\mathfrak{A}}$-algebra $(\mathcal{A}, [v, a, m])$, the unique $\Sigma_{\mathfrak{A}}$-algebra homomorphism into $\mathcal{A}$ will be called the *interpretation map* $\mathbb{i}_{\mathcal{A}} \colon \mathbb{T} \to \mathcal{A}$. Given a pointed module $\mathcal{X}$ and assuming $\mathcal{A}$ is a $\Sigma_{\mathfrak{A}}^{\mathcal{X}}$-algebra, Proposition 11.2.2 shows that $[\![\mathcal{X}, \mathcal{A}]\!]$ is a $\Sigma_{\mathfrak{A}}$-algebra; the homomorphism into it will be called the *traversal map* $\mathbb{t}_{\mathcal{A}}^{\mathcal{X}} \colon \mathbb{T} \to [\![\mathcal{X}, \mathcal{A}]\!]$. The corresponding $\Sigma_{\mathfrak{A}}$-algebra homomorphism conditions simplify to the following diagrams, where $s[\mathbb{t}] \colon \Sigma\mathbb{T} \to [\![\mathcal{X}, \Sigma\mathcal{A}]\!]$ and $e[\mathbb{t}] \colon \mathfrak{A} \oplus \mathbb{T} \to [\![\mathcal{X}, \mathfrak{A} \oplus \mathcal{A}]\!]$ are compositions of the skew-closed strengths for Σ and $\mathfrak{A}\oplus$ with the traversal map in the style of synthetic module functors:



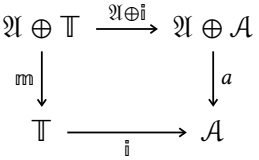
(𝕚𝕧)

$$\mathbb{i} \circ \mathbb{v} = v$$



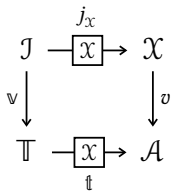
(𝕚𝕒)

$$\mathbb{i} \circ \mathbb{a} = a \circ \Sigma\mathbb{i}$$



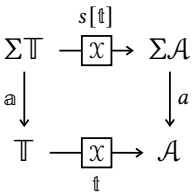
(𝕚𝕞)

$$\mathbb{i} \circ \mathbb{m}\{\varepsilon\} = m\{\mathbb{i} \circ \varepsilon\}$$



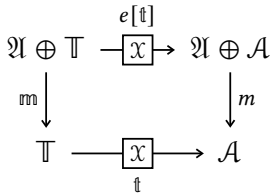
(𝕥𝕧)

$$\mathbb{t}\{\sigma\} \circ \mathbb{v} = v \circ \sigma$$



(𝕥𝕒)

$$\mathbb{t}\{\sigma\} \circ \mathbb{a} = a \circ s[\mathbb{t}]\{\sigma\}$$



(𝕥𝕞)

$$\mathbb{t}\{\sigma\} \circ \mathbb{m}\{\varepsilon\} = m\{\mathbb{t}\{\sigma\} \circ \varepsilon\}$$

Not only do we use initiality to induce interpretation and traversal maps, we also use the uniqueness of the induced maps to establish equality laws. To show that two maps out of $\mathbb{T}$ are equal, it is sufficient to establish that they are both $\vec{\Sigma}_{\Sigma}\mathcal{X}$-algebra homomorphisms; that is, Ⓐ the codomain of the maps is a $\vec{\Sigma}_{\Sigma}\mathcal{X}$-algebra, and Ⓗ the maps preserve variables, the

algebra structure, and metavariables. In particular, every homomorphism $\mathbb{T} \to \mathbb{T}$ must be the identity. The lemmas of the previous section will aid us in proving both of these requirements: Ⓐ by constructing the endpoint by composition of functors between $\Sigma_{\mathfrak{Y}}^{\mathcal{X}}$-algebras (ending with $\mathcal{X} = \mathcal{I}$), and Ⓗ by composing known $\Sigma_{\mathfrak{Y}}$-algebra homomorphisms. As an example, consider the following lemma relating interpretation and traversal maps.

Lemma 11.2.4 *Let $(\mathcal{A}, [v, a, m])$ be a $\Sigma_{\mathfrak{Y}}$-algebra with a pointed map $p \colon \mathcal{X} \to \mathcal{A}$, for a pointed $\Box$-coalgebra $(\mathcal{X}, r, \eta \colon \mathcal{I} \to \mathcal{X})$. Then, the composite below equals the interpretation map $\mathbb{i} \colon \mathbb{T} \to \mathcal{A}$:*

$$\mathbb{T} \xrightarrow{\mathbb{t}} [\![\mathcal{X}, \mathcal{A}]\!] \xrightarrow{[\![\eta, \mathcal{A}]\!]} [\![\mathcal{I}, \mathcal{A}]\!] \xrightarrow{i_{\mathcal{A}}} \mathcal{A}$$

Proof The lemma is proved by initiality.
Ⓐ The codomain $\mathcal{A}$ is a $\Sigma_{\mathfrak{Y}}$-algebra by assumption.
Ⓗ The composite is a $\Sigma_{\mathfrak{Y}}$-algebra homomorphism as so are the components:

$$\begin{array}{ll}
\mathbb{T} & \\
\Big\downarrow{\scriptstyle \mathbb{t}} & \text{initiality} \\
[\![\mathcal{X}, \mathcal{A}]\!] & \\
\Big\downarrow{\scriptstyle [\![\eta, \mathcal{A}]\!]} & \text{Prop. 11.2.2 with (†) } p \circ v = \eta \\
 & \text{by point-preservation of } p \\
[\![\mathcal{I}, \mathcal{A}]\!] & \\
\Big\downarrow{\scriptstyle i_{\mathcal{A}}} & \text{Lemma 11.2.2} \\
\mathcal{A} &
\end{array}$$

Thus, it is equal to the unique homomorphism $\mathbb{i} \colon \mathbb{T} \to \mathcal{A}$ induced by initiality. □

In the presheaf model, the initial $(\Omega, \mathcal{P})$-algebra is a presheaf by definition, with the renaming operation part of the structure. In the familial model, the renaming operation has to be defined separately, and it is a prerequisite to the substitution structure in which the pointed module parameter $\mathcal{X}$ is instantiated with $\mathbb{T}$. Fortunately, the renaming operation $\mathbb{T} \to \Box\mathbb{T}$ can be expressed as an $\mathcal{I}$-parametrised traversal.

Proposition 11.2.3 *$\mathbb{T}$ is a pointed $\Box$-coalgebra.*

Proof We induce the $\Box$-coalgebra structure map $\mathbb{r} \colon \mathbb{T} \to \Box\mathbb{T}$ as the unique traversal $\mathbb{t}^{\mathcal{I}}_{\mathbb{T}}$.

Counit law The counit law $\mathbb{T} \xrightarrow{\mathbb{r}} \Box\mathbb{T} \xrightarrow{i_{\mathbb{T}}} \mathbb{T} = \mathbb{T} \xrightarrow{\mathrm{id}} \mathbb{T}$ follows from Lemma 11.2.4, with $(\mathcal{Y}, y, \eta) \triangleq (\mathcal{I}, j_{\mathcal{I}}, \mathrm{id})$, $\mathcal{A} \triangleq \mathbb{T}$ and $p = \mathbb{v}$.

Comultiplication law The comultiplication law

$$\mathbb{T} \xrightarrow{\mathbb{r}} \Box\mathbb{T} \xrightarrow{\delta_{\mathbb{T}}} \Box\Box\mathbb{T} = \mathbb{T} \xrightarrow{\mathbb{r}} \Box\mathbb{T} \xrightarrow{\Box\mathbb{r}} \Box\Box\mathbb{T}$$

is established by showing that both composites are $\Sigma_{\mathfrak{A}}$-algebra homomorphisms, as Ⓐ is satisfied by two applications of Corollary 11.2.1. Note that $\delta_{\mathcal{X}} : \Box\mathcal{X} \to \Box\Box\mathcal{X}$ is implemented as $L[j_{\mathcal{I}}]_{\mathcal{X}} : [\![\mathcal{I}, \mathcal{X}]\!] \xrightarrow{L^{\mathcal{I}}_{\mathcal{I},\mathcal{X}}} [\![[\![\mathcal{I}, \mathcal{I}]\!], [\![\mathcal{I}, \mathcal{X}]\!]]\!] \xrightarrow{[\![j_{\mathcal{I}}, \mathrm{id}]\!]} [\![\mathcal{I}, [\![\mathcal{I}, \mathcal{X}]\!]]\!]$.

$\mathbb{T}$ — $\mathbb{r}$ — $\Box\mathbb{T}$ — $L[j_{\mathcal{I}}]$ — $\Box\Box\mathbb{T}$

initiality

Lem. 11.2.3 with pointed multilinear $j_{\mathcal{I}}$ (Lem. 5.2.1)

$\mathbb{T}$ — $\mathbb{r}$ — $\Box\mathbb{T}$ — $\Box\mathbb{r}$ — $\Box\Box\mathbb{T}$

initiality

Cor. 11.2.1 with initial homomorphism $\mathbb{r}$

Point Finally, $\mathbb{T}$ is pointed with $\mathbb{v} : \mathcal{I} \to \mathbb{T}$, and the point-renaming compatibility condition

$$\mathcal{I} \xrightarrow{\mathbb{v}} \mathbb{T} \xrightarrow{\mathbb{r}} \Box\mathbb{T} = \mathcal{I} \xrightarrow{j_{\mathcal{I}}} \Box\mathcal{I} \xrightarrow{\Box\mathbb{r}} \Box\mathbb{T}$$

is the $\mathbb{v}$-preservation law $\mathbb{tv}$ of $\mathbb{r}$. □

Proposition 11.2.4 *For a $\Sigma_{\mathfrak{A}}$-algebra $(\mathcal{A}, [v, a, m])$, if $\mathcal{A}$ possesses a pointed $\Box$-coalgebra structure $r : \mathcal{A} \to \Box\mathcal{A}$ such that r is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism, the initial interpretation map $\mathbb{i} : \mathbb{T} \to \mathcal{A}$ is a pointed $\Box$-coalgebra homomorphism.*

Proof $\Box\mathcal{A}$ is a $\Sigma_{\mathfrak{A}}$-algebra by Corollary 11.2.1. We show that both composites of the $\Box$-coalgebra homomorphism condition are $\Sigma_{\mathfrak{A}}$-algebra homomorphisms and are therefore equal:

$$\mathbb{T} \xrightarrow{\mathbb{r}} \Box\mathbb{T} \xrightarrow{\Box\mathbb{i}} \Box\mathcal{A} = \mathbb{T} \xrightarrow{\mathbb{i}} \mathcal{A} \xrightarrow{r} \Box\mathcal{A}$$

$\mathbb{T}$ — $\mathbb{r}$ — $\Box\mathbb{T}$ — $\Box\mathbb{i}$ — $\Box\mathcal{A}$

initiality of $\mathbb{r}$

Cor. 11.2.1 with initial homomorphism $\mathbb{i}$

$\mathbb{T}$ — $\mathbb{i}$ — $\mathcal{A}$ — r — $\Box\mathcal{A}$

initiality of $\mathbb{i}$

assumption

□

Proposition 11.2.5 *For a pointed $\Box$-coalgebra $(\mathcal{X}, \eta, x)$ and $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra homomorphism $(\mathcal{A}, [v, a, m])$, the induced traversal map $\mathbb{t} : \mathbb{T} \to [\![\mathcal{X}, \mathcal{A}]\!]$ is a pointed multilinear map if:*

- *$(\mathcal{A}, \mathcal{I} \xrightarrow{v} \mathcal{A} \xrightarrow{r} \Box\mathcal{A})$ is a pointed $\Box$-coalgebra;*
- *$f : \mathcal{X} \to \mathcal{A}$ is a pointed $\Box$-coalgebra homomorphism;*
- *$r : \mathcal{A} \to \Box\mathcal{A}$ is a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra homomorphism.*

Proof The first multilinear axiom expresses the $\Box$-coalgebra homomorphism condition of $\mathbb{t}$, which follows from Proposition 11.2.4 above with $(\mathcal{A}, r) \triangleq ([\![\mathcal{X}, \mathcal{A}]\!], L[j_{\mathcal{X}}]_{\mathcal{A}} : [\![\mathcal{X}, \mathcal{A}]\!] \to \Box[\![\mathcal{X}, \mathcal{A}]\!])$. As $j_{\mathcal{X}}$ is multilinear, $L[j_{\mathcal{X}}]_{\mathcal{A}}$ is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism by Lemma 11.2.3.

The second multilinear axiom expresses the equality of the composites

$$\mathbb{T} \xrightarrow{\mathbb{t}} [\![\mathcal{X}, \mathcal{A}]\!] \xrightarrow{[\![\mathcal{X}, r]\!]} [\![\mathcal{X}, \Box\mathcal{A}]\!] \;=\; \mathbb{T} \xrightarrow{\mathbb{t}} [\![\mathcal{X}, \mathcal{A}]\!] \xrightarrow{L[x]_{\mathcal{A}}} [\![\mathcal{X}, \Box\mathcal{A}]\!]$$

which we establish by showing Ⓐ for $[\![\mathcal{X}, \Box\mathcal{A}]\!]$ and Ⓗ for the morphisms:

$$\mathcal{A} \in \vec{\Sigma}^{\mathcal{X}}_{\mathcal{Y}} \xmapsto{\text{Cor. 11.2.1}} \Box\mathcal{A} \in \vec{\Sigma}^{\mathcal{X}}_{\mathcal{Y}} \xmapsto{\text{Prop. 11.2.2}} [\![\mathcal{X}, \Box\mathcal{A}]\!] \in \vec{\Sigma}_{\mathcal{Y}}$$

$$\begin{array}{ccc}
\mathbb{T} & & \\
{\scriptstyle \mathbb{t}}\downarrow & & \text{initiality} \\
[\![\mathcal{X}, \mathcal{A}]\!] & & \\
{\scriptstyle [\![\mathcal{X}, r]\!]}\downarrow & & \text{Prop. 11.2.2 with} \\
 & & (\dagger)\ r \circ q = \Box q \circ p \text{ as} \\
 & & q \in \Box\text{-}\mathbf{Coalg}(\mathcal{X}, \mathcal{A}) \\
[\![\mathcal{X}, \Box\mathcal{A}]\!] & &
\end{array}
\qquad
\begin{array}{ccc}
\mathbb{T} & & \\
{\scriptstyle \mathbb{t}}\downarrow & & \text{initiality} \\
[\![\mathcal{X}, \mathcal{A}]\!] & & \\
{\scriptstyle L[x]_{f}A}\downarrow & & \text{Lem. 11.2.3 with pointed} \\
 & & \text{multilinear } p \\
 & & \text{(Lem. 5.2.1)} \\
[\![\mathcal{X}, \Box\mathcal{A}]\!] & &
\end{array}$$

The map $\mathbb{t} \colon \mathbb{T} \to [\![\mathcal{X}, \mathcal{A}]\!]$ is pointed if $\mathbb{t}\{\eta\} \circ \mathbb{v} = v$. By Diagram (𝕥𝕧), $\mathbb{t}\{\eta\} \circ \mathbb{v} = f \circ \eta$, and $f \circ \eta = v$ since f is point-preserving. □

With these preliminaries, equipping $\mathbb{T}$ with an algebraic monoid structure – axiomatising the structural simultaneous substitution operation – follows a very similar pattern.

Proposition 11.2.6 *$\mathbb{T}$ is an invariant Σ-monoid.*

Proof The unit is the variable constructor $\mathbb{v} \colon \mathcal{I} \to \mathbb{T}$, while the substitution $\mathbb{s} \colon \mathbb{T} \to [\![\mathbb{T}, \mathbb{T}]\!]$ is induced as the unique traversal map $\mathbb{t}^{\mathbb{T}}_{\mathbb{T}}$, using $\mathbb{T}$ both as a pointed $\Box$-coalgebra (Proposition 11.2.3) and a $\Sigma^{\mathbb{T}}_{\mathcal{Y}}$-algebra $(\mathbb{T}, [\mathrm{id}_{\mathbb{T}}, \mathbb{a}, \mathbb{m}])$.

Left unit The left unit law is expressed by Diagram (𝕥𝕧) of $\mathbb{s}$:

$$\begin{array}{ccc}
\mathcal{I} & \xrightarrow{j_{\mathbb{T}}} & [\![\mathbb{T}, \mathbb{T}]\!] \\
{\scriptstyle \mathbb{v}}\downarrow & & \downarrow{\scriptstyle [\![\mathbb{T}, \mathrm{id}]\!]} \\
\mathbb{T} & \xrightarrow[\mathbb{s}]{} & [\![\mathbb{T}, \mathbb{T}]\!]
\end{array}$$

Right unit The right unit law requires showing that the following composite is the identity:

$$\mathbb{T} \xrightarrow{\mathbb{s}} [\![\mathbb{T}, \mathbb{T}]\!] \xrightarrow{[\![\mathbb{v}, \mathbb{T}]\!]} [\![\mathcal{I}, \mathbb{T}]\!] \xrightarrow{i_{\mathbb{T}}} \mathbb{T}$$

This is a consequence of Lemma 11.2.4 which shows that the composite must be equal to the unique point-preserving interpretation $\mathbb{i}_{\mathbb{T}} \colon \mathbb{T} \to \mathbb{T}$, which, by uniqueness, is the identity on $\mathbb{T}$.

Pointed multilinearity We show that $\mathbb{s}$ is a pointed multilinear map using Proposition 11.2.5:

- $\mathcal{A} = \mathcal{X} \triangleq \mathbb{T}$ is a pointed $\Box$-coalgebra by Proposition 11.2.3;
- $f \triangleq \mathrm{id} \colon \mathbb{T} \to \mathbb{T}$ is a pointed $\Box$-coalgebra homomorphism;

- $r \triangleq \mathbb{r} : \mathbb{T} \to \Box\mathbb{T}$ is a $\Sigma_{\mathfrak{A}}^{\mathbb{T}}$-algebra homomorphism because it is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism, $\Box\mathbb{T}$ is a $\Sigma_{\mathfrak{A}}^{\mathbb{T}}$-algebra with $v \triangleq \mathbb{r} : \mathbb{T} \to \Box\mathbb{T}$, and $\mathbb{r}$ preserves $\mathrm{id}_{\mathbb{T}}$ and $\mathbb{r}$.

Associativity The associativity law is the equality of the following composites, which we establish by showing that both are $\Sigma_{\mathfrak{A}}$-algebra homomorphisms:

$$\mathbb{T} \xrightarrow{\mathbb{s}} [\![\mathbb{T},\mathbb{T}]\!] \xrightarrow{[\![\mathbb{T},\mathbb{s}]\!]} [\![\mathbb{T},[\![\mathbb{T},\mathbb{T}]\!]]\!] \;=\; \mathbb{T} \xrightarrow{\mathbb{s}} [\![\mathbb{T},\mathbb{T}]\!] \xrightarrow{L[\mathbb{s}]_{\mathbb{T}}} [\![\mathbb{T},[\![\mathbb{T},\mathbb{T}]\!]]\!]$$

$$\begin{array}{ccc} \mathbb{T} & & \\ {\scriptstyle \mathbb{s}}\downarrow & & \text{initiality} \\ [\![\mathbb{T},\mathbb{T}]\!] & & \\ {\scriptstyle [\![\mathbb{T},\mathbb{s}]\!]}\downarrow & & \text{Prop. 11.2.2 with } \mathbb{s} \\ [\![\mathbb{T},[\![\mathbb{T},\mathbb{T}]\!]]\!] & & \text{in } \vec{\Sigma}_{\mathfrak{A}}^{\mathbb{T}} \text{ as } \mathbb{s} \circ \mathrm{id} = \mathbb{s} \end{array} \qquad \begin{array}{ccc} \mathbb{T} & & \\ {\scriptstyle \mathbb{s}}\downarrow & & \text{initiality} \\ [\![\mathbb{T},\mathbb{T}]\!] & & \\ {\scriptstyle L[\mathbb{s}]_{\mathbb{T}}}\downarrow & & \text{Lem. 11.2.3 with} \\ & & \text{pointed multilinear} \\ & & \mathbb{s} \text{ from above} \\ [\![\mathbb{T},[\![\mathbb{T},\mathbb{T}]\!]]\!] & & \end{array}$$

Compatibility with renaming structure The invariant monoid compatibility law is:

$$\mathbb{T} \xrightarrow{\mathbb{s}} [\![\mathbb{T},\mathbb{T}]\!] \xrightarrow{[\![\mathbb{v},\mathbb{T}]\!]} [\![\mathcal{I},\mathbb{T}]\!] \;=\; \mathbb{T} \xrightarrow{\mathbb{r}} [\![\mathcal{I},\mathbb{T}]\!]$$

The first one is the composite of the initial homomorphism $\mathbb{s}$, and the action $[\![\mathbb{v},\mathbb{T}]\!]$, which is homomorphic by Proposition 11.2.2 with $f \triangleq \mathbb{v} : \mathcal{I} \to \mathbb{T}$ and (†) $\mathrm{id} \circ \mathrm{id} \circ \mathbb{v} = \mathbb{v}$. The second one is the initial homomorphism into $\Box\mathbb{T}$, so by uniqueness, the two composites must be equal, and $\mathbb{T}$ is an invariant substitution monoid.

Compatibility with algebra structure The Σ-algebra structure map for $\mathbb{T}$ is $\mathbb{a} : \Sigma\mathbb{T} \to \mathbb{T}$. The Σ-monoid compatibility condition

$$\begin{array}{ccc} \Sigma\mathbb{T} & \xrightarrow{\widehat{\mathbb{T}}\,s[\mathbb{t}]} & \Sigma\mathbb{T} \\ {\scriptstyle \mathbb{a}}\downarrow & & \downarrow{\scriptstyle a} \\ \mathbb{T} & \xrightarrow[\mathbb{t}]{\widehat{\mathbb{T}}} & \mathbb{T} \end{array}$$

with $\widehat{\mathbb{T}} \in \mathcal{I}/\Box\text{-}\mathbf{Coalg}$ the pointed $\Box$-coalgebra with carrier $\mathbb{T}$ and structure map given by $\mathbb{T} \xrightarrow{\mathbb{s}} [\![\mathbb{T},\mathbb{T}]\!] \xrightarrow{[\![\mathbb{v},\mathrm{id}]\!]} \Box\mathbb{T}$ derived from the monoid structure. This is not precisely the $\mathbb{a}$-preservation law (𝕥𝕒) of $\mathbb{s}$, in which the parameter $\mathbb{T}$ is a $\Box$-coalgebra with $\mathbb{r}$. However, since we've established previously that $\mathbb{T}$ is an invariant monoid, both algebra structures are equivalent, so $\widehat{\mathbb{T}} \cong (\mathbb{T},\mathbb{v},\mathbb{r})$ and the components of the corresponding pointed strengths coincide. □

The proposition above achieves our main practical goal: it derives and verifies the structural simultaneous substitution operation directly from the inductive structure of the syntax. The "wall" of renaming–lifting–substitution compatibility lemmas from Section 2.1.3 is now in place – not by bypassing the work, but by proving each property where it necessarily arises,

whether as a naturality condition, a multilinearity axiom, or a monoid law. The underlying structure of families, modules, and monoids guided the development; no law had to be introduced ad hoc. To conclude, we turn to the proof of the freeness theorem.

Remark. From now on, the metavariable parameter of $\mathbb{T}$ will be made explicit; that is, we will write $\mathbb{T}\mathfrak{A}$ for the initial $\Sigma_{\mathfrak{A}}$-algebra. ⌟

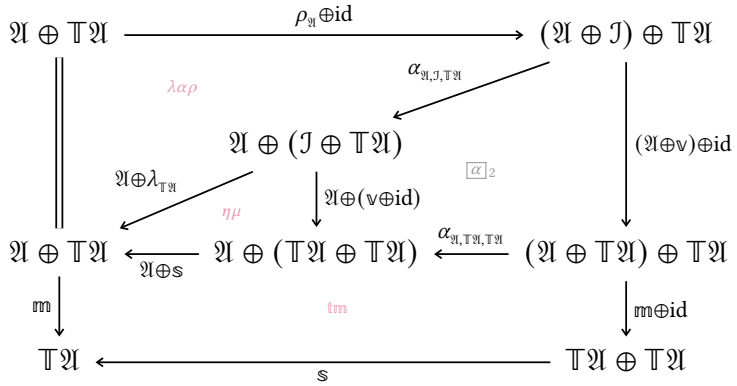
Definition 11.2.2 Every metavariable parameter $\mathfrak{A}$ has an associated term in $\mathbb{T}\mathfrak{A}$, given by the *unit* $\mathfrak{u}\colon \mathfrak{A} \to \mathbb{T}\mathfrak{A}$ of $\mathbb{T}$, a natural transformation with components:

$$\mathfrak{u}\colon \mathfrak{A} \xrightarrow{\rho_{\mathfrak{A}}} \mathfrak{A} \oplus \mathcal{I} \xrightarrow{\mathfrak{A}\oplus \mathfrak{v}} \mathfrak{A} \oplus \mathbb{T}\mathfrak{A} \xrightarrow{\mathfrak{m}} \mathbb{T}\mathfrak{A} \qquad ⌟$$

Lemma 11.2.5 *The metavariable constructor decomposes into the unit and substitution:*

$$\begin{array}{ccc} \mathfrak{A} \oplus \mathbb{T}\mathcal{M} & \xrightarrow{\mathfrak{u}\oplus \mathrm{id}} & \mathbb{T}\mathfrak{A} \oplus \mathbb{T}\mathcal{M} \\ & \searrow^{\mathfrak{m}} & \downarrow{\scriptstyle \mathfrak{s}} \\ & & \mathbb{T}\mathcal{M} \end{array} \qquad (\mathtt{sum})$$

Proof By expanding $\mathfrak{u}$, we have the following diagram:



Alternatively, we can express the unit in terms of the curried metavariable operator:

$$\mathfrak{u}\colon \mathfrak{A} \xrightarrow{\mathfrak{m}} [\![\mathbb{T}\mathfrak{A}, \mathbb{T}\mathfrak{A}]\!] \xrightarrow{[\![\mathfrak{v},\mathrm{id}]\!]} [\![\mathcal{I}, \mathbb{T}\mathfrak{A}]\!] \xrightarrow{i_{\mathbb{T}\mathfrak{A}}} \mathbb{T}\mathfrak{A}$$

and use a similar diagram to prove that $\mathfrak{A} \xrightarrow{\mathfrak{m}} [\![\mathbb{T}\mathfrak{A}, \mathbb{T}\mathfrak{A}]\!]$ equals $\mathfrak{m} \xrightarrow{\mathfrak{u}} \mathbb{T} \xrightarrow{\mathfrak{s}} [\![\mathbb{T}\mathfrak{A}, \mathbb{T}\mathfrak{A}]\!]$. □

Theorem 11.2.1

> $\mathbb{T}\mathfrak{A}$ *is the free* Σ*-monoid on* $\mathfrak{A}$.

Proof Proposition 11.2.6 showed that $\mathbb{T}\mathfrak{A}$ is a Σ-monoid on any sorted family $\mathfrak{A}$. To establish freeness, we need to show that given any Σ-monoid $(\mathcal{M}, \eta, \mu, a)$, any assignment $\omega\colon \mathfrak{A} \to \mathcal{M}$ can be extended to a unique Σ-monoid homomorphism $\mathfrak{e}_\omega\colon \mathbb{T}\mathfrak{A} \to \mathcal{M}$ along $\mathfrak{u}\colon \mathfrak{A} \to \mathbb{T}\mathfrak{A}$:

$$\begin{array}{ccc} \mathfrak{A} & \xrightarrow{\mathfrak{u}} & |\mathbb{T}\mathfrak{A}| \\ & \searrow^{\omega} & \downarrow{\scriptstyle |\mathfrak{e}_\omega|} \\ & & |\mathcal{M}| \end{array} \qquad (\mathtt{eu})$$

$$\mathfrak{e}_\omega \circ \mathfrak{m}\{\mathfrak{v}\} = \omega$$

As usual, this is done by equipping $\mathcal{M}$ with a $\Sigma_{\mathfrak{A}}$-algebra structure and inducing $\mathbb{e}_\omega$ by initiality. The variable embedding and Σ-algebra structures of $\mathcal{M}$ are given by η and a respectively, while the metavariable embedding $m\colon \mathfrak{A} \multimap \mathcal{M} \to \mathcal{M}$ is defined as the composite

$$\mathfrak{A} \xrightarrow{\omega} \mathcal{M} \overset{\mu}{\multimap} \mathcal{M} \to \mathcal{M}$$

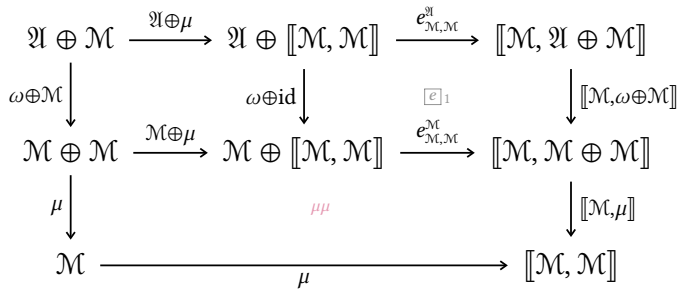
The extension $\mathbb{e}_\omega : \mathbb{T}\mathfrak{A} \to \mathcal{M}$ is then the initial $\Sigma_{\mathfrak{A}}$-algebra homomorphism into $(\mathcal{M}, [\eta, a, \mu \circ \omega])$. Left to show is that $\mathbb{e}_\omega$ is a Σ-monoid homomorphism that factors ω and the unique such map, using the following lemmas.

Lemma 11.2.6 *$[\![\mathcal{M}, \mathcal{M}]\!]$ is a $\Sigma_{\mathfrak{A}}$-algebra.*

Proof $\mathcal{M}$ is a $\Sigma_{\mathfrak{A}}$-algebra with $(\mathrm{id}_{\mathcal{M}}, a, \mu \circ \omega)$, so it is a $\Sigma_{\mathfrak{A}}^{\mathfrak{A}}$-algebra by Proposition 11.2.2. □

Lemma 11.2.7 *The monoid multiplication $\mu\colon \mathcal{M} \to [\![\mathcal{M}, \mathcal{M}]\!]$ is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism.*

Proof $[\![\mathcal{M}, \mathcal{M}]\!]$ is a $\Sigma_{\mathfrak{A}}$-algebra by above. The homomorphism conditions for μ are given directly by the left unit law, the Σ-monoid compatibility condition, and the associativity law of μ transposed in terms of e:



□

Lemma 11.2.8 *The extension $\mathbb{e}_\omega : \mathbb{T}\mathfrak{A} \to \mathcal{M}$ is a pointed $\Box$-coalgebra homomorphism.*

Proof $\mathcal{M}$ is a $\Box$-coalgebra with the structure $r : \mathcal{M} \xrightarrow{\mu} [\![\mathcal{M}, \mathcal{M}]\!] \xrightarrow{[\![\eta, \mathcal{M}]\!]} \Box\mathcal{M}$, which is furthermore a $\Sigma_{\mathfrak{A}}$-algebra homomorphism as so is μ from above, and $[\![\eta, \mathcal{M}]\!]$ by Proposition 11.2.2 with $f \triangleq \eta$ and (†) $\mathrm{id} \circ \mathrm{id} \circ \eta = \eta$ satisfied directly. Then, by Proposition 11.2.4, $\mathbb{e}_\omega : \mathbb{T}\mathfrak{A} \to \mathcal{M}$ is a pointed $\Box$-coalgebra homomorphism. □

Σ-monoid homomorphism We need to show that $\mathbb{e}_\omega$ preserves variables, the Σ-algebra structure, and substitution. The first two follow from the fact that $\mathbb{e}_\omega$ is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism. For preservation of substitution, we equate the following two composites:

$$\mathbb{T}\mathfrak{A} \xrightarrow{\mathbb{s}} [\![\mathbb{T}\mathfrak{A}, \mathbb{T}\mathfrak{A}]\!] \xrightarrow{[\![\mathrm{id}, \mathbb{e}_\omega]\!]} [\![\mathbb{T}\mathfrak{A}, \mathcal{M}]\!] = \mathbb{T}\mathfrak{A} \xrightarrow{\mathbb{e}_\omega} \mathcal{M} \xrightarrow{\mu} [\![\mathcal{M}, \mathcal{M}]\!] \xrightarrow{[\![\mathbb{e}_\omega, \mathcal{M}]\!]} [\![\mathbb{T}\mathfrak{A}, \mathcal{M}]\!]$$

$\mathbb{T}\mathfrak{A}$ — initiality of $\mathbb{s}$

$\downarrow \mathbb{s}$

$[\![\mathbb{T}\mathfrak{A}, \mathbb{T}\mathfrak{A}]\!]$ — Prop. 11.2.2 with $\mathbb{e}_\omega$ in $\vec{\Sigma}^{\mathbb{T}}_{\mathfrak{A}}$ and (†) $\mathbb{e}_\omega \circ \mathrm{id} = \mathbb{e}_\omega$

$\downarrow [\![\mathbb{T}\mathfrak{A}, \mathbb{e}_\omega]\!]$

$[\![\mathbb{T}\mathfrak{A}, \mathcal{M}]\!]$

$\mathbb{T}$ — initiality of $\mathbb{e}_\omega$

$\downarrow \mathbb{e}_\omega$

$\mathcal{M}$ — Lemma 11.2.7

$\downarrow \mu$

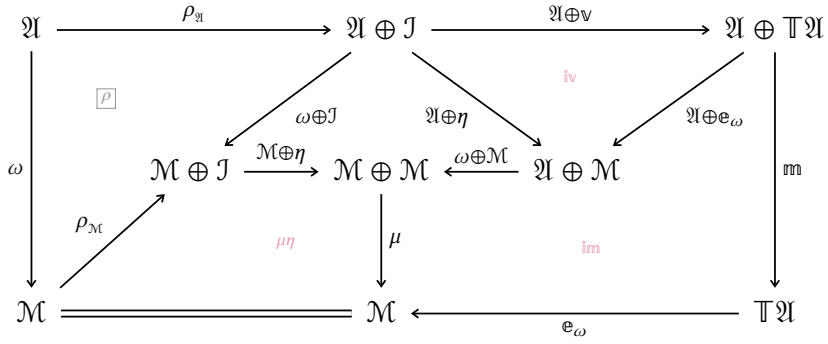
$[\![\mathcal{M}, \mathcal{M}]\!]$ — Prop. 11.2.2 with $\mathbb{e}_\omega \in \Box\text{-}\mathbf{Coalg}(\mathbb{T}\mathfrak{A}, \mathcal{M})$ by Lem. 11.2.8 and (†) $\mathrm{id} \circ \mathrm{id} \circ \mathbb{e}_\omega = \mathbb{e}_\omega$

$\downarrow [\![\mathbb{e}_\omega, \mathcal{M}]\!]$

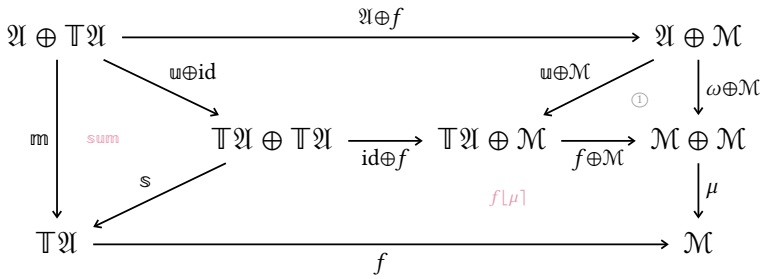
$[\![\mathbb{T}\mathfrak{A}, \mathcal{M}]\!]$

The composites are equal by uniqueness, so $\mathbb{e}_\omega : \mathbb{T}\mathfrak{A} \to \mathcal{M}$ is a $\Sigma_{\mathfrak{A}}$-algebra homomorphism.

Factoring We next show that ω factors through $\mathbb{e}_\omega$ with the following diagram in $\mathbf{Fam}_S$:



Uniqueness Finally we show that $\mathbb{e}_\omega$ equals any Σ-monoid homomorphism $f : \mathbb{T}\mathfrak{A} \to \mathcal{M}$ that factorises ω: ① $f \circ \mathbb{u} = \omega$. This is done by proving that f must be a $\Sigma_{\mathfrak{A}}$-homomorphism and is therefore equal to the initial unique map $\mathbb{e}_\omega$. As before, the preservation of variables and the Σ-algebra structure follow from the analogous axioms of Σ-monoid homomorphisms. The preservation of metavariables is as follows:



□

In practice, the free monoid theorem implies the following: given a term $t \in \mathbb{T}\mathfrak{A}_\alpha(\Gamma)$ and a Σ-monoid $\mathcal{M}$ – that is, a model of the syntax equipped with substitution – any interpretation of metavariables $\omega \colon \mathfrak{A} \to \mathcal{M}$ induces a homomorphic semantic interpretation $\mathbb{e}_\omega \colon \mathbb{T}\mathfrak{A} \to \mathcal{M}$. This homomorphism is structurally recursive, preserving variables and the Σ-algebra structure, and it satisfies the semantic substitution lemma: syntactic substitution corresponds to monoid multiplication in the model. Traditionally, these properties require a long list of intricate lemmas involving renaming, weakening, lifting, and more. In the algebraic familial model, however, once we verify that $\mathcal{M}$ is a valid model, the denotational interpretation follows automatically – no additional machinery is needed.

Theorem 11.2.2

The free Σ-monoid assignment $\mathfrak{A} \mapsto \mathbb{T}\mathfrak{A}$ is the object mapping of the free Σ-monoid functor $\mathbb{T} \colon \mathbf{Fam}_S \to \Sigma\mathbf{Mon}$.

Proof The functorial action maps a function $f \colon \mathfrak{A} \to \mathfrak{B}$ to a Σ-monoid homomorphism $\mathbb{T}f \colon \mathbb{T}\mathfrak{A} \to \mathbb{T}\mathfrak{B}$. This is induced by freeness: since $\mathbb{T}\mathfrak{B}$ is a Σ-monoid and we have the assignment $\omega \triangleq \mathfrak{A} \xrightarrow{f} \mathfrak{B} \xrightarrow{\mathbb{u}} \mathbb{T}\mathfrak{B}$, there is a unique homomorphic extension $\mathbb{e}_{\mathbb{u} \circ f} \colon \mathbb{T}\mathfrak{A} \to \mathbb{T}\mathfrak{B}$ which we define to be $\mathbb{T}f$. □

We will also write $\mathbb{T} \colon \Sigma\mathbf{Mon} \to \Sigma\mathbf{Mon}$ for the induced Σ-monoid endofunctor, mapping $(\mathcal{M}, a, \eta, \mu)$ to the Σ-monoid $(\mathbb{T}\mathcal{M}, \mathbb{a}, \mathbb{v}, \mathbb{s})$ on the underlying family.

Corollary 11.2.2 *By freeness, we have a natural transformation $\mathbb{o} \colon \mathbb{T} \Longrightarrow \mathrm{Id}_{\Sigma\mathbf{Mon}} \colon \Sigma\mathbf{Mon} \to \Sigma\mathbf{Mon}$, with components $\mathbb{o}_{\mathcal{M}} \colon \mathbb{T}\mathcal{M} \to \mathcal{M}$ given by the extension*

$$\begin{array}{ccc} \mathcal{M} & \xrightarrow{\mathbb{u}_{\mathcal{M}}} & \mathbb{T}\mathcal{M} \\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle \mathbb{o}_{\mathcal{M}} \triangleq \mathbb{e}_{\mathrm{id}}} \\ & & \mathcal{M} \end{array}$$

In particular, we obtain the *free Σ-monoid monad* $\mathbb{T} \colon \mathbf{Fam}_S \to \mathbf{Fam}_S = \mathfrak{A} \mapsto |\mathbb{T}\mathfrak{A}|$ on families, with unit $\mathbb{u}_{\mathfrak{A}} \colon \mathfrak{A} \to \mathbb{T}\mathfrak{A}$ and multiplication $\mathbb{j}_{\mathfrak{A}} \triangleq \mathbb{o}_{\mathbb{T}\mathfrak{A}} \colon \mathbb{T}(\mathbb{T}\mathfrak{A}) \to \mathbb{T}\mathfrak{A}$. Consequently, every Σ-monoid $\mathcal{M}$ is an algebra for the monad $\mathbb{T}$, with structure map $\mathbb{o} \triangleq \mathbb{o}_{\mathcal{M}} \colon \mathbb{T}\mathcal{M} \to \mathcal{M}$ satisfying the unit law by the factoring property, and the multiplication law by the naturality of $\mathbb{o}$:

$$\begin{array}{ccc} \mathbb{T}(\mathbb{T}\mathcal{M}) & \xrightarrow{\mathbb{j}_{\mathcal{M}} = \mathbb{o}_{\mathbb{T}\mathcal{M}}} & \mathbb{T}\mathcal{M} \\ {\scriptstyle \mathbb{T}\mathbb{o}}\downarrow & & \downarrow{\scriptstyle \mathbb{o}} \\ \mathbb{T}\mathcal{M} & \xrightarrow[\mathbb{o}_{\mathcal{M}}]{} & \mathcal{M} \end{array}$$

Thus, as expected, the freeness theorem yields a free Σ-monoid functor and induces a functor $\Sigma\mathbf{Mon} \to \mathbb{T}\text{-}\mathbf{Alg}$. Though we won't rely on this, one can show that the categories are equivalent by proving the forgetful functor $\Sigma\mathbf{Mon} \to \mathbf{Fam}_S$ is monadic, using techniques based on equational systems from Fiore and Hur (2009). Instead, we turn to the second-order aspects of the syntax: metasubstitution and sound second-order equational reasoning.

11.3 Second-order syntax

The original presheaf model of Fiore et al. (1999) addressed syntax with variables and binders, supporting both single-variable and simultaneous substitution. Hamana (2004) extended this to include parametrised metavariables and characterised free Σ-monoids, marking a move toward a general theory of *second-order syntax*. However, key practical limitations remained unresolved until Fiore (2008), who introduced two crucial refinements: an emphasis on *metasubstitution* – mapping metavariables to terms with other metavariables – and its *internalisation*, which generalises Hamana's monadic approach to allow metavariables that reference variables beyond their declared parameters. Building on results from earlier sections, we translate internal metasubstitution into the family setting and show its central role in enabling sound second-order equational reasoning.

11.3.1 Metasubstitution

The free Σ-monoid theorem directly gives rise to a form of metasubstitution implemented as the Kleisli extension; this was first proposed (in the monadic form) by Hamana (2004).

Definition 11.3.1 The *external metasubstitution operation* is the Kleisli extension

$$\mathbf{Fam}_S(\mathfrak{A}, \mathbb{T}\mathfrak{B}) \to \mathbf{Fam}_S(\mathbb{T}\mathfrak{A}, \mathbb{T}\mathfrak{B}) \in \mathbf{Set}$$

⌟

Given a metavariable $\mathfrak{m} \in \mathfrak{A}_\tau\Pi$, a metasubstitution rule $\zeta : \mathfrak{A} \to \mathbb{T}\mathfrak{B}$ has to be type-and context-preserving, so may only map $\mathfrak{m}$ to terms in the same context Π. In particular, a metavariable without parameters must be mapped to a closed term. For example, given $\mathfrak{m} \in \mathfrak{A}_\alpha[\beta, \gamma]$ and $\mathfrak{n} \in \mathfrak{A}_\beta[]$ and the term $x : \alpha, y : \beta, z : \gamma \vdash \mathfrak{m}\{\mathfrak{n}\{\}, z\} : \mathbb{T}\mathfrak{A}_\alpha[]$, we cannot instantiate $\mathfrak{n}$ with y as y is not well-typed in the empty parameter context of $\mathfrak{n}$, and neither can we instantiate $\mathfrak{m}$ with anything that refers to x, y or z.

The solution is to decouple the type and context of a metasubstitution rule from the ambient context, allowing the instantiating terms to refer both to metavariable parameters as well as any variables in the context. In the presheaf model this is made possible by internalising the metasubstitution operation not as a mapping of hom-sets, but a natural transformation of presheaf exponentials. Writing $\langle \mathcal{P}, \mathcal{Q} \rangle \in \mathbf{PSh}$ for $\prod_{\tau \in S} \mathcal{P}_\tau \supset \mathcal{Q}_\tau$, *internal metasubstitution operation* in presheaves is

$$\mathbf{ms}_{\mathcal{P},\mathcal{Q}} : \langle \mathcal{P}, \mathbf{T}\mathcal{Q} \rangle \to \langle \mathbf{T}\mathcal{P}, \mathbf{T}\mathcal{Q} \rangle \in \mathbf{PSh}$$

with two variations (uncurried and curried) being

$$\mathbf{T}\mathcal{P} * \langle \mathcal{P}, \mathbf{T}\mathcal{Q} \rangle \to \mathbf{T}\mathcal{Q} \qquad \mathbf{T}\mathcal{P} \to \langle \mathcal{P}, \mathbf{T}\mathcal{Q} \rangle \supset \mathbf{T}\mathcal{Q}$$

Fiore (2008) builds the former by equipping the term monad $\mathbf{T}$ with a cartesian strength $\mathbf{cs}_{\mathcal{P},\mathcal{Q}} : \mathbf{T}\mathcal{P} * Q \to \mathbf{T}(\mathcal{P} * Q)$, using which an internal meta-renaming and meta-substitution operation is derived:

$$\mathbf{mr}_{\mathcal{P},\mathcal{Q}} : \mathbf{T}\mathcal{P} * \langle \mathcal{P}, \mathcal{Q} \rangle \xrightarrow{\mathbf{cs}_{\mathcal{P},\langle \mathcal{P},\mathcal{Q} \rangle}} \mathbf{T}(\mathcal{P} * \langle \mathcal{P}, \mathcal{Q} \rangle) \xrightarrow{\mathbf{T}\varepsilon^{\mathcal{P}}_{\mathcal{Q}}} \mathbf{T}\mathcal{Q}$$

$$\mathbf{ms}_{\mathcal{P},\mathcal{Q}} \colon \mathbf{T}\mathcal{P} * \langle \mathcal{P}, \mathbf{T}\mathcal{Q} \rangle \xrightarrow{\mathbf{mr}_{\mathcal{P},\mathcal{Q}}} \mathbf{TT}\mathcal{Q} \xrightarrow{\mathbf{j}_{\mathcal{Q}}} \mathbf{T}\mathcal{Q}$$

The cartesian strength for $\mathbf{T}$ is dependent on a defined cartesian strength on Ω, and when this cartesian strength is compatible with the pointed strength, the term monad $\mathbf{T}$ is cartesian strong (Fiore, 2008, Theorem 13).

To translate the above to the family setting, we analyse how the constructions simplify for free metavariable presheaves $\mathcal{P} \triangleq \blacklozenge\mathcal{A}$. Once again, we use the isomorphism $\star(\blacklozenge X \supset Q) \cong X \multimap \star Q$ and $\star\mathbf{T}\blacklozenge \cong \mathbb{T}$ to simplify $\langle \blacklozenge\mathcal{A}, \mathbf{T}\blacklozenge\mathcal{B} \rangle$ and $\langle \mathbf{T}\blacklozenge\mathcal{A}, \mathbf{T}\blacklozenge\mathcal{B} \rangle$ to

$$\prod_{\tau \in S} \mathcal{A}_\tau \multimap \mathbb{T}\mathcal{B}_\tau \quad \text{and} \quad \prod_{\tau \in S} \mathbb{T}\mathcal{A}_\tau \multimap \mathbb{T}\mathcal{B}_\tau$$

respectively. Writing $\langle\!\langle \mathcal{X}, \mathcal{Y} \rangle\!\rangle \triangleq \prod_{\tau \in S} \mathcal{X}_\tau \multimap \mathcal{Y}_\tau$ for the left closed action (enrichment) of $\mathbf{Fam}_S$ over $\multimap$, we have the following definition.

Definition 11.3.2 The *internal metasubstitution operation* is the **Fam**-morphism

$$\mathbb{ms}_{\mathcal{A},\mathcal{B}} \colon \langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{B} \rangle\!\rangle \to \langle\!\langle \mathbb{T}\mathcal{A}, \mathbb{T}\mathcal{B} \rangle\!\rangle \in \mathbf{Fam}$$

also equivalently written using the right action (powering) $\multimap\!\bullet \colon \mathbf{Fam}^{\mathrm{op}} \times \mathbf{Fam}_S \to \mathbf{Fam}_S$ as

$$\mathbb{ms}_{\mathcal{A},\mathcal{B}} \colon \mathbb{T}\mathcal{A} \to \langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{B} \rangle\!\rangle \multimap\!\bullet\, \mathbb{T}\mathcal{B} \qquad \lrcorner$$

Of the two approaches, we prefer the second, as it is formulated entirely in terms of the Day-closed structure and makes the term $\mathbb{T}\mathcal{A}$ explicit in the domain. Several construction strategies are available. Since $\mathbb{T}\mathcal{A}$ is the initial $\Sigma_{\mathcal{A}}$-algebra, we can define metasubstitution by giving $\langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{A} \rangle\!\rangle \multimap\!\bullet\, \mathbb{T}\mathcal{B}$ a $\Sigma_{\mathcal{A}}$-algebra structure and appealing to initiality. Alternatively, treating $\mathbb{T}\mathcal{A}$ as the free Σ-monoid, we can show that $\langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{A} \rangle\!\rangle \multimap\!\bullet\, \mathbb{T}\mathcal{B}$ is itself a Σ-monoid and define an embedding $\omega \colon \mathcal{A} \to \langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{A} \rangle\!\rangle \multimap\!\bullet\, \mathbb{T}\mathcal{B}$.

We adopt a more abstract route using the framework from Chapter 4, based on clones and powered monad morphisms. This approach automatically ensures the correctness properties of metasubstitution, avoiding the need to reprove them via initiality or freeness. To summarise the relevant results of the chapter: if $\mathbb{T}$ is a powered monad, then any $\mathbb{T}$-algebra $\mathbb{T}\mathcal{M} \to \mathcal{M}$ corresponds bijectively to a powered monad morphism $\mathbb{T} \Longrightarrow (\!|\mathcal{A}|\!)$, where the abstract clone $(\!|\mathcal{A}|\!)$ is given by $(\!|\mathcal{A}|\!)(\mathcal{X}) \triangleq (\!|\mathcal{A}, \mathcal{A}|\!)(\mathcal{X}) = \langle\!\langle \mathcal{X}, \mathcal{A} \rangle\!\rangle \multimap\!\bullet\, \mathcal{A}$. The induced operation, known as *meta-extension*, internalises the free extension: it interprets a term in $\mathbb{T}\mathcal{A}$ in a model $\mathcal{M}$, given an internal assignment of $\mathcal{A}$-metavariables into $\mathcal{M}$:

$$\mathbb{me}_{\mathcal{A},\mathcal{M}} \colon \mathbb{T}\mathcal{A} \to (\!|\mathcal{M}|\!)\mathcal{A} = \mathbb{T}\mathcal{A} \to \langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{M} \rangle\!\rangle \multimap\!\bullet\, \mathbb{T}\mathcal{M}$$

Metasubstitution is then the meta-extension with $\mathcal{M} = \mathbb{T}\mathcal{N}$.

To make use of the framework, we first prove that $\mathbb{T}$ is a powered monad: ① we equip it with a transformation $\mathbb{p}_{W,\mathcal{X}} \colon \mathbb{T}(W \multimap\!\bullet\, \mathcal{X}) \to W \multimap\!\bullet\, \mathbb{T}\mathcal{X}$, ② prove its compatibility with the closed structure $i_{\mathcal{X}} \colon E \multimap\!\bullet\, \mathcal{X} \to \mathcal{X}$ and $c_{\mathcal{X}}^{U,W} \colon (U \circledcirc W) \multimap\!\bullet\, \mathcal{X} \to (U \multimap\!\bullet\, (W \multimap\!\bullet\, \mathcal{X}))$, and ③ with the monad structure $\mathbb{u} \colon \mathrm{Id} \Longrightarrow \mathbb{T}$ and $\mathbb{j} \colon \mathbb{T}\mathbb{T} \Longrightarrow \mathbb{T}$. Conveniently enough, all the work for ① and ③ has already been laid out in Section 3.4, as demonstrated next.

Proposition 11.3.1 *For all $W : \mathbf{Fam}$, $\varphi_W : \mathbb{T}(W \multimap (-)) \Longrightarrow (W \multimap \mathbb{T}(-))$ is a family of monad-functor distributive laws, and the family is natural in W.*

Proof We know from Theorem 10.3.5 that $W \multimap (-) : \mathbf{Fam}_S \to \mathbf{Fam}_S$ lifts to Σ-monoids along the forgetful functor $\Sigma\mathbf{Mon} \to \mathbf{Fam}_S$. This, by Theorem 3.4.1, induces a monad-functor distributive law $\mathbb{T}(W \multimap (-)) \Longrightarrow W \multimap \mathbb{T}(-) : \mathbf{Fam}_S \to \mathbf{Fam}_S$, satisfying the compatibility conditions

$$\begin{array}{ccc} W \multimap \mathcal{X} & & \\ {\scriptstyle \mathbb{u}_{W\multimap\mathcal{X}}}\swarrow \quad \searrow{\scriptstyle W\multimap\mathbb{u}_{\mathcal{X}}} & & (\mathbb{pu}) \\ \mathbb{T}(W \multimap \mathcal{X}) \xrightarrow[(\varphi_W)_{\mathcal{X}}]{} W \multimap \mathbb{T}\mathcal{X} & & \end{array} \qquad \begin{array}{ccccc} & & \mathbb{T}(W \multimap \mathbb{T}\mathcal{X}) & & \\ & {\scriptstyle \mathbb{T}((\varphi_W)_{\mathcal{X}})}\nearrow & & \searrow{\scriptstyle (\varphi_W)_{\mathbb{T}\mathcal{X}}} & \\ \mathbb{TT}(W \multimap \mathcal{X}) & & & & W \multimap \mathbb{TT}\mathcal{X} \quad (\mathbb{pj}) \\ {\scriptstyle \mathbb{j}_{W\multimap\mathcal{X}}}\downarrow & & & & \downarrow{\scriptstyle W\multimap\mathbb{j}_{\mathcal{X}}} \\ \mathbb{T}(W \multimap \mathcal{X}) & & \xrightarrow[(\varphi_W)_W]{} & & W \multimap \mathbb{T}\mathcal{X} \end{array}$$

For $f : W \to U \in \mathbf{Fam}$, the natural transformation $f \multimap (-) : (U \multimap) \Longrightarrow (W \multimap)$ lifts to a Σ-monoid homomorphism also by Theorem 10.3.5, so by Proposition 3.4.1 the free monad $\mathbb{T}$ distributes over $f \multimap (-)$, giving the naturality of $\varphi_W : \mathbb{T}(W \multimap (-)) \Longrightarrow W \multimap \mathbb{T}(-)$ in the W component:

$$\begin{array}{ccc} \mathbb{T}(U \multimap \mathcal{X}) & \xrightarrow{(\varphi_U)_{\mathcal{X}}} & U \multimap \mathbb{T}\mathcal{X} \\ {\scriptstyle \mathbb{T}(f\multimap\mathcal{X})}\downarrow & & \downarrow{\scriptstyle f\multimap\mathbb{T}\mathcal{X}} \\ \mathbb{T}(W \multimap \mathcal{X}) & \xrightarrow[(\varphi_W)_{\mathcal{X}}]{} & W \multimap \mathbb{T}\mathcal{X} \end{array}$$

□

Theorem 11.3.1
$\mathbb{T}$ is a powered monad.

Proof The powering components are given by the natural family of distributive laws above:

$$\mathbb{p}_{W,\mathcal{X}} \triangleq (\varphi_W)_{\mathcal{X}} : \mathbb{T}(W \multimap \mathcal{X}) \to (W \multimap \mathbb{T}\mathcal{X}) : \mathbf{Fam}^{\mathrm{op}} \times \mathbf{Fam}_S \to \mathbf{Fam}_S$$

Its compatibility with the monad structure of $\mathbb{T}$ is also proved above. Left is the establishment of the powering laws (compatibility with i and c), which we do via freeness.

$$\mathbb{T}(E \multimap \mathcal{Y}) \xrightarrow{\mathbb{p}_{E,\mathcal{Y}}} E \multimap \mathbb{T}\mathcal{Y} \xrightarrow{i_{\mathbb{T}\mathcal{Y}}} \mathbb{T}\mathcal{Y} \;=\; \mathbb{T}(E \multimap \mathcal{Y}) \xrightarrow{\mathbb{T}i_{\mathcal{Y}}} \mathbb{T}\mathcal{Y}$$

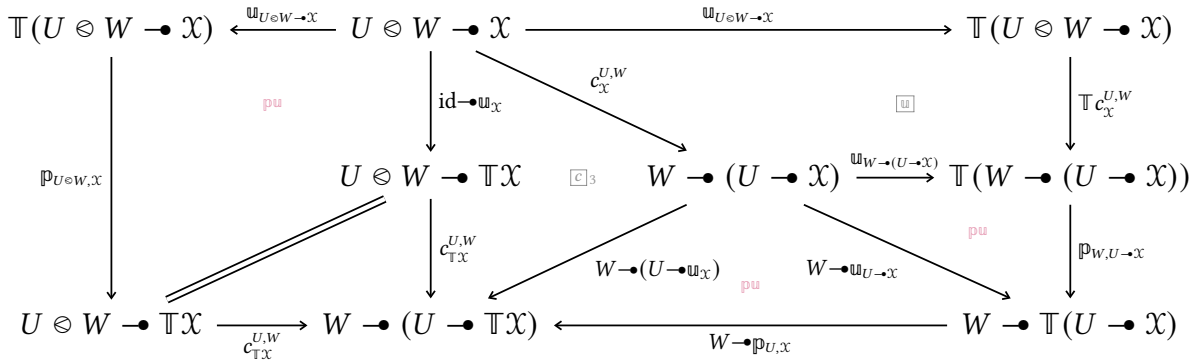
The first composite is a Σ-monoid homomorphism: $\mathbb{p}$ is a homomorphic extension and $i_{\mathbb{T}\mathcal{Y}}$ by Theorem 10.3.5. The second arrow is a functorial map of $\mathbb{T} : \mathbf{Fam}_S \to \Sigma\mathbf{Mon}$ (Theorem 11.2.2) and is therefore homomorphic. Both maps factor through $\omega \triangleq \mathbb{u}_{\mathcal{Y}} \circ i_{\mathcal{Y}} : E \multimap \mathcal{Y} \to \mathbb{T}\mathcal{Y}$:

$$\begin{array}{ccccc} \mathbb{T}(E \multimap \mathcal{Y}) & \xleftarrow{\mathbb{u}_{E\multimap\mathcal{Y}}} & E \multimap \mathcal{Y} & \xrightarrow{\mathbb{u}_{E\multimap\mathcal{Y}}} & \mathbb{T}(E \multimap \mathcal{Y}) \\ {\scriptstyle \mathbb{p}_{E,\mathcal{Y}}}\downarrow & \mathbb{pu} \quad {\scriptstyle E\multimap\mathbb{u}_{\mathcal{Y}}}\searrow & & & \\ E \multimap \mathbb{T}\mathcal{Y} & & {\scriptstyle i_{\mathcal{Y}}}\downarrow & \boxed{\mathbb{u}} & \downarrow{\scriptstyle \mathbb{T}i_{\mathcal{Y}}} \\ {\scriptstyle i_{\mathbb{T}\mathcal{Y}}}\downarrow & \boxed{i} & & & \\ \mathbb{T}\mathcal{Y} & \xleftarrow[\mathbb{u}_{\mathcal{Y}}]{} & \mathcal{Y} & \xrightarrow[\mathbb{u}_{\mathcal{Y}}]{} & \mathbb{T}\mathcal{Y} \end{array}$$

Both composites of the law between $\mathbb{p}$ and $c^{U,W}_{\mathcal{X}} : (U \circledcirc W \multimap \mathcal{X}) \to W \multimap (U \multimap \mathcal{X})$ are Σ-monoid homomorphisms:

$$\begin{array}{ccc}
\mathbb{T}(U \circledcirc W \multimap \mathcal{X}) & & \\
{\scriptstyle \mathbb{p}_{U\circledcirc W,\mathcal{X}}}\downarrow & \text{freeness} & \\
U \circledcirc W \multimap \mathbb{T}\mathcal{X} & & \\
{\scriptstyle c^{U,W}_{\mathbb{T}\mathcal{X}}}\downarrow & \text{Thm. 10.3.5} & \\
W \multimap (U \multimap \mathbb{T}\mathcal{X}) & &
\end{array}
\qquad
\begin{array}{ccc}
\mathbb{T}(U \circledcirc W \multimap \mathcal{X}) & & \\
{\scriptstyle \mathbb{T}c^{U,W}_{\mathcal{X}}}\downarrow & \text{Thms. 10.3.5 and 11.2.2} & \\
\mathbb{T}(U \multimap (W \multimap \mathcal{X})) & & \\
{\scriptstyle \mathbb{p}_{W,U\multimap\mathcal{X}}}\downarrow & \text{freeness} & \\
W \multimap \mathbb{T}(U \multimap \mathcal{X}) & & \\
{\scriptstyle W\multimap\mathbb{p}_{U,\mathcal{X}}}\downarrow & \text{Thm. 10.3.5 and freeness} & \\
W \multimap (U \multimap \mathbb{T}\mathcal{X}) & &
\end{array}$$

Both maps factor through $(U \circledcirc W \multimap \mathcal{X}) \xrightarrow{\mathrm{id}\multimap\mathbb{u}_{\mathcal{X}}} (U \circledcirc W \multimap \mathbb{T}\mathcal{X}) \xrightarrow{c^{U,W}_{\mathbb{T}\mathcal{X}}} (W \multimap (U \multimap \mathbb{T}\mathcal{X}))$:



Thus, $\mathbb{p}$ is a powering over $\multimap$, compatible with the monad structure of $\mathbb{T}$. □

Instantiating Theorem 4.3.1 with the strong monad $\mathbb{T}$, we get the following.

Definition 11.3.3 For all Σ-monoids $\mathcal{M}$, the components of the powered monad map $\mathbb{T} \Longrightarrow (\!|\mathcal{M}|\!)$ thus induced give the *internal meta-extension* operation

$$\mathbb{me}_{\mathcal{A},\mathcal{M}} : \mathbb{T}\mathcal{A} \to (\langle\!\langle \mathcal{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M}) \quad \sim \quad \langle\!\langle \mathcal{A}, \mathcal{M} \rangle\!\rangle \to \langle\!\langle \mathbb{T}\mathcal{A}, \mathcal{M} \rangle\!\rangle$$

With $\mathcal{M} \triangleq \mathbb{T}\mathcal{B}$, the meta-extension specialises to the *internal metasubstitution* operation

$$\mathbb{ms}_{\mathcal{A},\mathcal{B}} : \mathbb{T}\mathcal{A} \to (\langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{B} \rangle\!\rangle \multimap \mathbb{T}\mathcal{B}) \quad \sim \quad \langle\!\langle \mathcal{A}, \mathbb{T}\mathcal{B} \rangle\!\rangle \to \langle\!\langle \mathbb{T}\mathcal{A}, \mathbb{T}\mathcal{B} \rangle\!\rangle$$

⌟

It's worth unpacking the metasubstitution operation, since deriving it through multiple abstract layers can obscure its practical meaning. The powering of $\mathbb{T}$ is defined via the map

$$W \multimap [\![\mathcal{X}, \mathcal{Y}]\!] \to [\![W \multimap \mathcal{X}, W \multimap \mathcal{Y}]\!]$$

which is the closed analogue of the map $m[-] : (W \multimap \mathcal{X}) \oplus (W \multimap \mathcal{Y}) \to W \multimap (\mathcal{X} \oplus \mathcal{Y})$ from Theorem 10.3.1. Precomposing this with the monoid multiplication $\mathcal{M} \to [\![\mathcal{M}, \mathcal{M}]\!]$, and lifting the signature endofunctor Σ through the same powering, allows us to promote $W \multimap (-)$ to an endofunctor on Σ-monoids, as shown in Theorem 10.3.5. The meta-extension (and metasub-

stitution) operation is then freely induced by endowing the internal hom $\langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \to \mathcal{M}$ with Σ-monoid structure. The resulting $\Sigma_{\mathfrak{A}}$-algebra homomorphism conditions shown below give a structural definition of the metasubstitution operation $\mathbb{me}$, ensuring it preserves variables, constructors, and substitution in a way that aligns with the underlying syntax.

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{\eta} & \mathcal{M} \\ {\scriptstyle \mathbb{v}}\downarrow & & \downarrow{\scriptstyle \kappa_{\mathcal{M}}^{\langle\!\langle \mathfrak{A},\mathcal{M}\rangle\!\rangle}} \\ \mathbb{T}\mathfrak{A} & \xrightarrow[\mathbb{me}_{\mathfrak{A},\mathcal{M}}]{} & \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M} \end{array} \qquad \begin{array}{ccccc} \Sigma(\mathbb{T}\mathfrak{A}) & \xrightarrow{\Sigma \mathbb{me}_{\mathfrak{A},\mathcal{M}}} & \Sigma(\langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M}) & \xrightarrow{p^{\Sigma}_{\langle\!\langle \mathfrak{A},\mathcal{M}\rangle\!\rangle,\mathcal{M}}} & \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \Sigma\mathcal{M} \\ {\scriptstyle \mathbb{a}}\downarrow & & & & \downarrow{\scriptstyle \mathrm{id} \multimap a} \\ \mathbb{T}\mathfrak{A} & & \xrightarrow[\mathbb{me}_{\mathfrak{A},\mathcal{M}}]{} & & \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M} \end{array}$$

$$\begin{array}{ccccc} \mathfrak{A} \oplus \mathbb{T}\mathfrak{A} & \xrightarrow{\beta^{\mathcal{M}}_{\mathfrak{A}} \oplus \mathbb{me}_{\mathfrak{A},\mathcal{M}}} & (\langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M}) \oplus (\langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M}) & \xrightarrow{d^{\langle\!\langle \mathfrak{A},\mathcal{M}\rangle\!\rangle}_{\mathcal{M},\mathcal{M}}} & \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap (\mathcal{M} \oplus \mathcal{M}) \\ {\scriptstyle \mathbb{a}}\downarrow & & & & \downarrow{\scriptstyle \mathrm{id} \multimap \mu} \\ \mathbb{T}\mathfrak{A} & & \xrightarrow[\mathbb{me}_{\mathfrak{A},\mathcal{M}}]{} & & \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M} \end{array}$$

The diagrams correspond to the following explicit recursive definitions:

$$\begin{aligned} \mathbb{me}\,\lfloor\zeta\rfloor\,(\mathbb{v}\,v) &\triangleq \eta\,(\iota_2^{\Gamma,\Delta}\,v) \\ \mathbb{me}\,\lfloor\zeta\rfloor\,(\mathbb{a}\,t) &\triangleq a\,(p\,\lfloor\zeta\rfloor\,(\Sigma\,\mathbb{me}\,t)) \\ \mathbb{me}\,\lfloor\zeta\rfloor\,(\mathbb{m}\{\varepsilon\}\mathfrak{a}) &\triangleq \mu\,\{(\mathbb{me}\,\lfloor\zeta\rfloor \circ \varepsilon) \rtimes \iota_1^{\Gamma,\Delta}\}\,(\zeta\,\mathfrak{a}) \end{aligned}$$

We work through the case of metasubstituting into metavariables. Take a parametrised metavariable $\mathfrak{a} \in \mathfrak{A}_\alpha\Pi$, an environment $\varepsilon\colon \Pi \dashrightarrow \boxed{\mathbb{T}\mathfrak{A}} \to \Gamma$, and a metasubstitution rule $\zeta \in \langle\!\langle \mathfrak{A}, \mathbb{T}\mathfrak{B} \rangle\!\rangle\Delta = \prod_{\tau,\Theta} \mathfrak{A}_\tau\Theta \to \mathbb{T}\mathfrak{B}_\tau(\Theta + \Delta)$. Metasubstituting $(\mathbb{m}\{\varepsilon\}\mathfrak{a}) \in \mathbb{T}\mathfrak{A}_\alpha\Gamma$ with ζ starts by looking up the metavariable $\mathfrak{a}$ in the rule, which gives a term $\zeta\,\mathfrak{a} \in \mathbb{T}\mathfrak{B}_\alpha(\Pi + \Delta)$ (instantiating τ with α and Θ with Π). However, $\mathbb{ms}\,\lfloor\zeta\rfloor\,(\mathbb{m}\{\varepsilon\}\mathfrak{a})$ must be in $\mathbb{T}\mathfrak{B}_\alpha(\Gamma + \Delta)$, so the context of $\zeta\,\mathfrak{a}$ needs to be adjusted by a substitution rule ${}^{\Pi+\Delta}(\mathbb{T}\mathfrak{B})_{\Gamma+\Delta}$. This is exactly where $\varepsilon \in {}^{\Pi}\mathbb{T}\mathfrak{A}_\Gamma$ comes in, which maps every parameter in Π to a $\mathbb{T}\mathfrak{A}$-term in Γ. To bring the metavariable contexts to the right shape, the terms in ε are recursively metasubstituted, giving a substitution rule $\mathcal{I}_\alpha\Pi \xrightarrow{\varepsilon} \mathbb{T}\mathfrak{A}_\alpha\Gamma \xrightarrow{\mathbb{ms}\,\lfloor\zeta\rfloor} \mathbb{T}\mathfrak{B}_\alpha(\Delta + \Gamma)$. Finally, this is widened by the injection $\iota_1^{\Gamma,\Delta}\colon \mathcal{I}_\alpha\Gamma \to \mathcal{I}_\alpha(\Delta + \Gamma)$ to give the final substitution rule

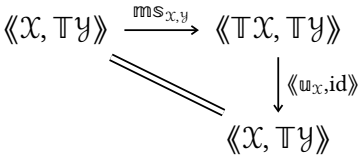
$$(\mathbb{me}\,\lfloor\zeta\rfloor \circ \varepsilon) \rtimes \iota_1^{\Gamma,\Delta} \in {}^{\Pi+\Delta}(\mathbb{T}\mathfrak{B})_{\Gamma+\Delta}$$

Applying this with the object-level substitution operation to $\zeta\,\mathfrak{a}\colon \mathbb{T}\mathfrak{B}_\alpha(\Pi + \Delta)$ gives us a term in $\mathbb{s}\,\{(\mathbb{me}\,\lfloor\zeta\rfloor \circ \varepsilon) \rtimes \iota_1^{\Gamma,\Delta}\}\,(\zeta\,\mathfrak{a}) \in \mathbb{T}\mathfrak{B}_\alpha(\Gamma + \Delta)$, as required.

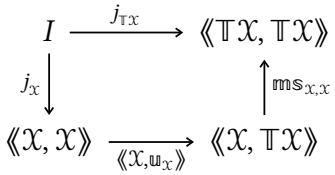
We now have the metasubstitution and meta-extension operations in hand, but their role and governing laws remain to be placed in an abstract setting. Their representation as internalised extensions offers a key insight: the term monad $\mathbb{T}$ is not merely a monad on $\mathbf{Fam}_S$, but a monad *enriched* in $\mathbf{Fam}$, with hom-objects given by $\langle\!\langle -, = \rangle\!\rangle\colon \mathbf{Fam}_S^{\mathrm{op}} \times \mathbf{Fam}_S \to \mathbf{Fam}$. The equivalence of enrichment and powering was the subject of Theorem 4.3.2, and instantiated with the powered monad $\mathbb{T}$, we obtain the enriched unit and extension operations

$$\mathbb{u}\colon \mathcal{X} \to \mathbb{T}\mathcal{X} \qquad \mathbb{ms}\colon \langle\!\langle \mathcal{X}, \mathbb{T}\mathcal{Y} \rangle\!\rangle \to \langle\!\langle \mathbb{T}\mathcal{X}, \mathbb{T}\mathcal{Y} \rangle\!\rangle$$

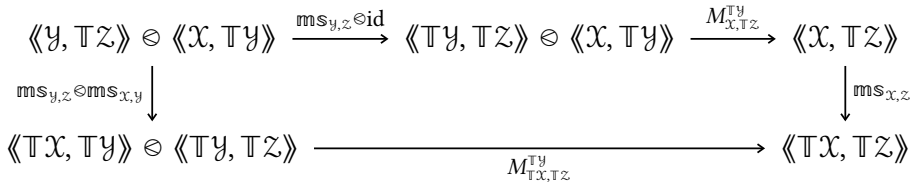
satisfying the following axioms:



$$\mathbb{ms}\,\lfloor\zeta\rfloor\,(\mathbb{u}\,a) = \zeta(a)$$

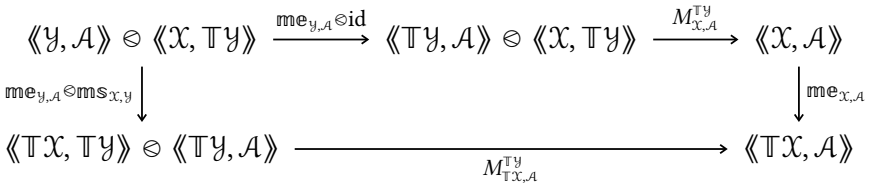


$$\mathbb{ms}\,\lfloor\mathbb{u}\rfloor\,t = t$$



$$\mathbb{ms}\,\lfloor\xi\rfloor\,(\mathbb{ms}\,\lfloor\zeta\rfloor\,t) = \mathbb{ms}\,\lfloor\mathbb{ms}\lfloor\xi\rfloor\circ\zeta\rfloor\,t$$

The equational laws clearly encapsulate the intended unit and associativity laws that metasubstitution should satisfy. Similarly, the associativity of meta-extension is the law



$$\mathbb{me}\,\lfloor\kappa\rfloor\,(\mathbb{ms}\,\lfloor\zeta\rfloor\,t) = \mathbb{ms}\,\lfloor\mathbb{me}\lfloor\kappa\rfloor\circ\zeta\rfloor\,t \tag{‡}$$

that is used in the soundness of the equational theory of the next section.

With this final characterisation of the second-order meta-extension and metasubstitution operations, our study of second-order abstract syntax is complete. A direct application of metasubstitution, meta-extension, and their laws concludes this chapter.

11.3.2 Equational logic

Among the first applications of the algebraic theory of second-order abstract syntax were *term equational systems* and *second-order equational logic*. The general theory was developed by Fiore and Hur (2007, 2008, 2009, 2010), Hur (2010), and Fiore (2013). We distil the core elements of this research to a simple and implementation-friendly equational system that allows the specification of second-order axioms between terms with variable binders and parametrised metavariables, and synthesises a logic for second-order equational reasoning. The soundness of the logic is established using the metasubstitution laws introduced in the previous section.

Notation. Elements of the set $\mathbb{T}\mathfrak{A}_\alpha\Gamma$ for $\mathfrak{A} \in \mathbf{Fam}_S$, $\alpha \in S$ and $\Gamma \in \mathbb{F}[S]$ will also be denoted $\mathfrak{A} \triangleright \Gamma \vdash t : \alpha$. ⌟

Definition 11.3.4 Let $A \subseteq \sum_{\mathfrak{A}\in\mathbf{Fam}_S} \sum_{\Gamma\in\mathbb{F}[S]} \sum_{\alpha\in\Gamma} (\mathfrak{A} \triangleright \Gamma \vdash t : \alpha) \times (\mathfrak{A} \triangleright \Gamma \vdash t : \alpha)$ be a relation between (meta)variable contexts, terms and sorts, with elements denoted $\mathfrak{A} \triangleright \Gamma \vdash t = s : \alpha$.

Define an *equational theory* with axioms A with the following rules and axioms:

$$\textsc{ax}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash t = s : \alpha}{\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha} \qquad \textsc{refl}\ \frac{}{\mathfrak{A} \triangleright \Gamma \vdash t \equiv t : \alpha} \qquad \textsc{textsf}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha}{\mathfrak{A} \triangleright \Gamma \vdash s \equiv t : \alpha}$$

$$\textsc{trans}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha \qquad \mathfrak{A} \triangleright \Gamma \vdash s \equiv u : \alpha}{\mathfrak{A} \triangleright \Gamma \vdash t \equiv u : \alpha} \qquad \textsc{ren}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha \qquad \rho : \Gamma \to \Delta}{\mathfrak{A} \triangleright \Delta \vdash \mathbb{r}\, t\, \rho \equiv \mathbb{r}\, s\, \rho : \alpha}$$

$$\textsc{msub}\ \frac{\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha \qquad \zeta, \xi \in \langle\!\langle \mathfrak{A}, \mathbb{T}\mathfrak{B} \rangle\!\rangle \Delta \qquad \forall \mathfrak{a} \in \mathfrak{A}_\tau \Pi.\ \mathfrak{B} \triangleright \Pi + \Delta \vdash \zeta\, \mathfrak{a} \equiv \xi\, \mathfrak{a} : \tau}{\mathfrak{B} \triangleright \Gamma + \Delta \vdash \mathbb{ms}\, t\, \zeta \equiv \mathbb{ms}\, s\, \xi : \alpha}$$

The rules and axioms above define an inductive proof system for equating two second-order terms. A second-order axiom is a template for an equality rule, with metavariables standing for terms that can be instantiated arbitrarily (as long as the type and parameter context match). For example, in the second-order theory of the λ-calculus (temporarily extended with arithmetic, to make for more interesting examples), the axioms representing β and η axioms would be stated as follows, with metavariable families $\mathfrak{A}$ and $\mathfrak{B}$ defined appropriately:

$$\begin{aligned} \mathfrak{a} : \mathfrak{A}_\beta[\alpha], \mathfrak{b} : \mathfrak{A}_\alpha[] \triangleright [] \vdash \mathsf{app}\,(\mathsf{lam}\,(x.\,\mathfrak{a}\{x\}), \mathfrak{b}) &\equiv \mathfrak{a}\{\mathfrak{b}\} : \beta \\ \mathfrak{g} : \mathfrak{B}_{\alpha\to\beta}[] \triangleright [] \vdash \mathsf{lam}\,(x.\,\mathfrak{g}\,x) &\equiv \mathfrak{g} \quad : \alpha \to \beta \end{aligned}$$

Any instance of β-equivalence may be derived by instantiating the axiom with the MSUB rule with terms of the appropriate form. For example, given terms

$$\emptyset \triangleright x : \mathbb{N} \to \mathbb{N}, y : \mathbb{N} \vdash x \cdot (y + 5) : \mathbb{N} \qquad \emptyset \triangleright y : \mathbb{N} \vdash y - 1 : \mathbb{N}$$

we construct the metasubstitution rule $\zeta \triangleq \{\mathfrak{a}\lfloor x \rfloor \mapsto x \cdot (y + 5), \mathfrak{b} \mapsto y - 1\} \in \langle\!\langle \mathfrak{A}, \emptyset \rangle\!\rangle [y : \mathbb{N}]$, and applying it to both sides of the β axiom gives the derived equality:

$$\emptyset \triangleright y : \mathbb{N} \vdash \mathsf{app}\,(\mathsf{lam}\,(x.\,x \cdot (y + 5)), y - 1) \equiv (y - 1) \cdot (y + 5) : \mathbb{N}$$

The rule REN, not part of the original equational logic of Fiore and Hur (2010), is required to prove the general congruence property, where the rewritten subexpressions are open terms. As a variation of the example above, take

$$\emptyset \triangleright y : \mathbb{N} \vdash y \cdot (\mathsf{app}\,(\mathsf{lam}\,(x.\,x + y), 3)) : \mathbb{N}$$

The abstracted term $\mathfrak{b} : \mathfrak{B}_\mathbb{N}[\mathbb{N}] \triangleright [y : \mathbb{N}] \vdash y \cdot \mathfrak{b}\{y\} : \mathbb{N}$ and metasubstitution rules $\zeta, \xi \in \langle\!\langle \mathfrak{A}, \emptyset \rangle\!\rangle [y : \mathbb{N}]$ now live in the same nonempty context, and applying $\mathbb{ms}$ will combine the contexts to $[y : \mathbb{N}, y : \mathbb{N}]$; or more generally, to $\Gamma + \Gamma$. Collapsing the repeated context elements is done with the rule REN, with the diagonal renaming $[\mathrm{id}, \mathrm{id}] : \Gamma + \Gamma \to \Gamma$. Since metasubstitution always extends the context of terms, such a renaming cannot be captured as a special case of metasubstitution, hence its addition to the equational logic; this is not a problem in the presheaf model, as the presheaf exponentials can represent more complex forms of context manipulation, not just concatenation.

While the logic allows for equational reasoning within the syntax, this is not connected to any models yet. Given a set of axioms, we may consider models – Σ-monoids – in which the axioms become equal when interpreted.

Definition 11.3.5 A Σ-monoid $\mathcal{M}$ is *a model for a second-order axiom* $\mathfrak{A} \triangleright \Gamma \vdash t = s : \alpha$, denoted $\mathcal{M} \vDash \mathfrak{A} \triangleright \Gamma \vdash t = s : \alpha$, if $\mathbb{me}\, t = \mathbb{me}\, s \in (\!|\mathcal{M}|\!)\mathfrak{A}$; explicitly, for all $\zeta : \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \Delta$, we have $\mathbb{me}\, \lfloor \zeta \rfloor\, t = \mathbb{me}\, \lfloor \zeta \rfloor\, s \in \mathcal{M}_\alpha(\Gamma + \Delta)$. $\mathcal{M}$ *satisfies* $\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha$, denoted $\mathcal{M} \vDash \mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha$, if $\mathbb{me}\, t = \mathbb{me}\, s \in (\!|\mathcal{M}|\!)\mathfrak{A}$. ⌟

Definition 11.3.6 For a set of axioms A, a *(Σ, A)-monoid* $\mathcal{M}$ is a Σ-monoid such that for every axiom in A, $\mathcal{M} \vDash \mathfrak{A} \triangleright \Gamma \vdash t = s : \alpha$. ⌟

The theory of metasubstitution and meta-extension developed earlier makes the soundness of the equational logic quite easy to prove.

Theorem 11.3.2

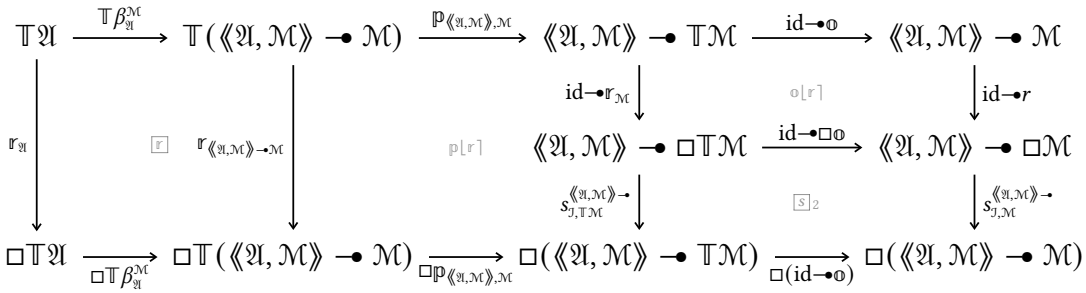
Let A be a set of axioms over the signature Σ. If $\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha$ is derivable for two terms $t, s \in \mathbb{T}\mathfrak{A}_\alpha\Gamma$, then $\mathcal{M} \vDash \mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha$ for all (Σ, A)-monoids $\mathcal{M}$.

Proof Let $t, s \in \mathbb{T}\mathfrak{A}_\alpha(\Gamma)$ be two terms and assume $\mathfrak{A} \triangleright \Gamma \vdash t \equiv s : \alpha$. We prove the soundness theorem by induction on the derivation of the equality.

Case AX $\mathcal{M}$ is a (Σ, A)-monoid, so it satisfies the axiom by assumption.

Case REFL, SYM, TRANS Follow by the equivalence relation property of equality.

Case REN Let $\zeta : \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \Theta$ a metasubstitution rule and assume inductively that $\mathbb{me}\, a\, \zeta = \mathbb{me}\, b\, \zeta$. We first show that $\mathbb{me} : \mathbb{T}\mathfrak{A} \to \langle\!\langle \mathfrak{A}, \mathcal{M} \rangle\!\rangle \multimap \mathcal{M}$ is a $\Box$-coalgebra homomorphism using the diagram below, where the naturality and preservation conditions concerning $\mathbb{r}$ are derived from those of $\mathbb{s}$, and the fact that $\mathbb{o}$, $\mathbb{p}$ and $\mathbb{T}f$ are Σ-monoid homomorphisms:



With this, meta-extension of a renamed term calculates as follows:

$$\begin{aligned} \mathbb{me}\, (\mathbb{r}\, a\, \rho)\, \kappa &= s\{\rho\}(r \circ \mathbb{me}\, a\, \kappa) \\ &= s\{\rho\}(r \circ \mathbb{me}\, b\, \kappa) && \text{(induction)} \\ &= \mathbb{me}\, (\mathbb{r}\, b\, \rho)\, \kappa \end{aligned}$$

Case MSUB Let $\zeta, \xi : \langle\!\langle \mathfrak{A}, \mathbb{T}\mathfrak{B} \rangle\!\rangle \Delta$ be metasubstitution rules, and $\kappa : \langle\!\langle \mathfrak{B}, \mathcal{M} \rangle\!\rangle \Theta$ a meta-extension rule. Assume inductively that $\mathbb{ms}\, t = \mathbb{ms}\, s$ and for all $\mathfrak{b} \in \mathfrak{B}_\tau(\Pi)$, $\zeta\, \mathfrak{b} = \xi\, \mathfrak{b}$. We use the

associativity property of metasubstitution shown in Diagram (‡) to calculate as follows:

$$\begin{aligned}
\mathtt{me}\,\lfloor\kappa\rfloor\,(\mathtt{ms}\,\lfloor\zeta\rfloor\,t) &= \mathtt{ms}\,\lfloor\mathtt{me}\lfloor\kappa\rfloor\circ\zeta\rfloor\,t && \text{(Diagram (‡))}\\
&= \mathtt{ms}\,\lfloor\mathtt{me}\lfloor\kappa\rfloor\circ\xi\rfloor\,s && \text{(induction)}\\
&= \mathtt{me}\,\lfloor\kappa\rfloor\,(\mathtt{ms}\,\lfloor\xi\rfloor\,s) && \text{(Diagram (‡))}
\end{aligned}$$

Thus, for every model, satisfaction of axioms extends to satisfaction of any equation provable from the axioms. □

This chapter unified the abstract developments from earlier sections to construct the familial model of second-order abstract syntax. Using initial algebra semantics, we derived homomorphic interpretation maps across models parametrised by renaming, substitution, and metasubstitution. We proved a freeness theorem linking initial second-order algebras with free models, and established correctness and compatibility laws via the uniqueness of initial semantics and free extensions. Finally, we placed internalised metasubstitution and meta-extension in an enriched setting, and used this structure to formulate a sound, syntax-generic equational logic for second-order reasoning.

Summary of Part III

Part III introduced the family-based model of second-order abstract syntax, deriving it from and relating it to the presheaf model. We addressed the limitations of the presheaf setting – such as the absence of presheaf actions and the need for quotienting – by formulating the model in more general contexts: skew-monoidal categories and the Day convolution structure. Beginning in Chapters 8 and 9, we developed the general theory of presheaves, showing how enriched structure on the base category yields stronger substitution principles in the presheaf category. The restricted nature of discrete presheaves, or type- and context-indexed families, prompted a careful treatment of renaming, naturality, and parametrisation in Chapter 10. This, in turn, enabled a reformulation of the key theorems of the presheaf model that underpin the development of second-order abstract syntax in Chapter 11.

Next, we demonstrate how the abstract theory of the familial model directly informs the implementation of a generic syntax-formalisation library, and how it can be used for syntax-generic metatheory.

PART IV

APPLICATIONS

The categorical theory of second-order abstract syntax developed so far has been entirely mathematical, though always with a practical goal in mind: to distil core constructions and theorems into a robust foundation for mechanising second-order calculi. Our aim was not to formally verify the familial model itself, but to use its insights to build a formalisation framework that eliminates the tedious and error-prone boilerplate common in mechanised programming language research.

In this final part of the thesis, we outline the key components of our Agda implementation of a second-order formalisation framework. Chapter 12 provides a guided overview of the library, focusing on the main challenges of encoding abstract categorical ideas in a concrete setting. We then turn to applications in Chapter 13, demonstrating how the framework supports both generic metatheory and the implementation of specific syntaxes.

CHAPTER 12

Computer formalisation

The theory developed thus far represents only part of the contribution of this research. The practical complement is the development of a syntax-generic formalisation framework for second-order calculi in Agda, called `agda-soas`. This framework implements the core elements of the mathematical theory and serves as a proving ground for its claims of practicality and formalisability. It is not intended to mechanise the familial model in its full abstraction and generality. Rather, this principled, selective approach enabled the creation of a streamlined and relatively simple system, underpinned by a robust mathematical foundation. Unlike many existing formalisation frameworks, `agda-soas` handles nearly all boilerplate upfront, using syntax-generic initial algebra semantics; only minimal syntax-specific metatheory is required, and even that is automatically generated from a concise syntax-description language.

Since the theoretical underpinnings of the library have already been presented, we avoid repeating the formal definitions except for illustrative purposes. Sections 12.1 and 12.2 provide examples of how the abstract categorical theory is realised as clean, idiomatic Agda code. Of particular interest is the interface between the generic metatheory and user-defined syntax signatures, the subtleties of which are discussed in Section 12.3.

This chapter builds on our POPL 2022 distinguished paper (Fiore and Szamozvancev, 2022), with supplementary materials available at the paper's website. The `agda-soas` library, available on GitHub, currently lags slightly behind the full theory presented in this thesis, as it does not yet incorporate the enriched metasubstitution framework.

12.1 Familial model

We start by laying down the basic foundations of the familial model: contexts and families in Section 12.1.1, and the generic axiomatisation of renaming and substitution in Section 12.1.2.

12.1.1 Contexts, families, and variables

The definitions of contexts and variables are well-known from intrinsically-typed treatments of syntax. Rather than using named variable-type pairs, contexts are lists of types that come

from a fixed set S, and variables are typed and scoped de Bruijn indices into the context: new points to the first element of the context, while old(v) points to the variable v in an extended context. This corresponds to the coproduct structure of the presheaf of variables.

```
data Ctx : Set where
  ∅   : Ctx
  _·_ : (α : S) → (Γ : Ctx) → Ctx
```

```
data ℐ : S → Ctx → Set where
  new :          ℐ α (α · Γ)
  old : ℐ β Γ → ℐ β (α · Γ)
```

Families and sorted families are indexed families of sets, expressed in Agda as functions in the universe level Set_1, containing types and the type of types Set:

```
Fam : Set₁
Fam = Ctx → Set
```

```
Famₛ : Set₁
Famₛ = S → Fam
```

```
_⇾_ : Famₛ → Famₛ → Set
𝒳 ⇾ 𝒴 = {α : S}{Γ : Ctx} → 𝒳 α Γ → 𝒴 α Γ
```

The concatenation of contexts is used in the definition of context extension, as well as the Day homs. The derivation of the latter from the former is possible, but would result in an unnecessary indirection step – we will often prefer to define things concretely, where the derivation from a more general construct is of mathematical interest only.

```
_+_ : Ctx → Ctx → Ctx
∅       + Δ = Δ
(α · Γ) + Δ = α · (Γ + Δ)
```

```
δ : Ctx → Fam → Famₛ
δ Θ X Γ = X (Θ + Γ)
```

```
_⊸_ : Fam → Fam → Fam
(X ⊸ Y) Γ = {Δ : Ctx} → X Δ → Y (Γ + Δ)
```

```
_⟜_ : Fam → Fam → Fam
(X ⟜ Y) Γ = {Δ : Ctx} → X Δ → Y (Δ + Γ)
```

Renaming and substitution rules are type-preserving families of maps from variables to variables or families, with a change of context:

```
_–[_]→_ : Ctx → Famₛ → Ctx → Set
Γ –[ 𝒳 ]→ Δ = {α : S} → ℐ α Γ → 𝒳 α Δ
```

```
_⇝_ : Ctx → Ctx → Set
Γ ⇝ Δ = Γ –[ ℐ ]→ Δ
```

The cocartesian structure of contexts and renamings is exhibited by the definitions of the injections and copairing, both done by induction on the variable syntax.

```
inl : (Δ : Ctx) → Γ ⇝ (Γ + Δ)
inl Δ new     = new
inl Δ (old v) = old (inl Δ v)
```

```
inr : (Γ : Ctx) → Δ ⇝ (Γ + Δ)
inr ∅       v = v
inr (α · Γ) v = old (inr Γ v)
```

```
copair : (𝒳 : Famₛ) → (Γ –[ 𝒳 ]→ Θ) → (Δ –[ 𝒳 ]→ Θ) → (Γ + Δ) –[ 𝒳 ]→ Θ
copair 𝒳 {∅}     σ ς v       = ς v
copair 𝒳 {α · Γ} σ ς new     = σ new
copair 𝒳 {α · Γ} σ ς (old v) = copair 𝒳 (σ ∘ old) ς v
```

A useful special case of copairing is adding a single term to a substitution, which corresponds to the standard *cons* operation in the theory of explicit substitutions (Abadi et al., 1991).

```
add : (𝒳 : Famₛ) → 𝒳 α Δ → Γ –[ 𝒳 ]→ Δ → (α · Γ) –[ 𝒳 ]→ Δ
add 𝒳 t σ = copair 𝒳 (λ{ new → t }) σ
```

To bridge the gap between the categorical theory and existing syntax formalisation efforts, we include a glossary of terms to connect `agda-soas` to other proposed frameworks, most notably the state of the art work by Allais et al. (2021) on the `generic-syntax` library.

Families McBride (2005) recognises the value of abstracting intrinsically-typed variables and terms into intrinsically-typed sets, but gives them the vague name of *stuff*, denoted ⧫. Allais et al. (2021) calls S-sorted families S -Scoped and introduces some syntactic sugar for constructing and manipulating sorted (and unsorted families). The Scope operator corresponds to context extension.

Substitution rules One of the main innovations of McBride (2005) is to generalise the notion of a renaming and substitution rule into a type-preserving map from variables over Γ to "stuff" over Δ. Allais et al. (2021) writes (Γ -Env) $\mathcal{X}$ Δ for such transformations, and calls renaming maps Thinnings. The $\bullet$ operator corresponds to our add, and select is simply the composition of a renaming and substitution rule.

12.1.2 Renaming and substitution

Renaming We seek a way to equip families with a functorial action of the form

$$\mathsf{r} : \{\Gamma\ \Delta : \mathsf{Ctx}\} \to (\Gamma \rightsquigarrow \Delta) \to X\ \Gamma \to X\ \Delta$$

that, syntactically, corresponds to a renaming operation. Rearranging this to the form

$$\mathsf{r} : \{\Gamma : \mathsf{Ctx}\} \to X\ \Gamma \to \big(\{\Delta : \mathsf{Ctx}\} \to (\Gamma \rightsquigarrow \Delta) \to X\ \Delta\big)$$

we note that the codomain is a function of α and Γ, and refactoring this as an operator on families gives us the *cofree presheaf comonad*:

$$\begin{aligned}&\Box : \mathsf{Fam} \to \mathsf{Fam}\\ &\Box\ X\ \Gamma = \{\Delta : \mathsf{Ctx}\} \to (\Gamma \rightsquigarrow \Delta) \to X\ \Delta\end{aligned}$$

The coalgebra structure map for this family expands precisely to the type of r above, with the coalgebra axioms corresponding to the expected functoriality laws: preservation of identity and composition of renamings. In Agda, such structures are typically axiomatized using parametrised records. We follow the standard approach of separating *structures* – which equip a given object with additional structure – from *bundles* – which package the object together with its structure. This pattern is familiar from libraries such as `agda-stdlib` and `agda-categories` (Hu and Carette, 2021, Section 4.2).

```
record Coalg (X : Fam) : Set where
  field  r    : X ⇾ □ X
         r-id : {t : X Γ} → r[ id ] t ≡ t
         r-∘  : {ρ : Γ ↝ Δ}{ϱ : Δ ↝ Θ}{t : X Γ} → r[ ϱ ∘ ρ ] t ≡ r[ ϱ ] (r[ ρ ] t)
```

For syntactic simplicity, we use the synonym r[ρ] t for r t ρ. Since context transformations such as weakening, contraction, etc. correspond to renaming maps, the associated structural rules for a □-coalgebra X can be derived via renaming:

```
wkl : X Γ → X (Γ + Δ)          wkr : X Δ → X (Γ + Δ)          contr : X (Γ + Γ) → 𝒳 Γ
wkl t = r[ inl Δ ] t           wkr t = r[ inr Γ ] t           contr t = r[ copair 𝒥 id id ] t
```

A morphism of families that preserves the renaming operation is a □-coalgebra homomorphism The natural notion of a transformation between □-coalgebras is a *homomorphism*: a map $X \to Y$ that preserves the coalgebra structures of X and Y.

```
record Coalg⇒ (X□ : Coalg X)(Y□ : Coalg Y) (f : X → Y) : Set where
  field ⟨r⟩ : {ρ : Γ ⇝ Δ}{t : X Γ} → f (X.r[ ρ ] t) ≡ Y.r[ ρ ] (f t)
```

By turning the structures defined here into bundles, we can define the category of coalgebras and coalgebra homomorphisms; while we won't detail it here, the `agda-soas` library has some generic functionality to define categories of objects with extra structure with minimal effort.

```
Coalgebra : Set₁                 Coalgebra⇒ : Coalgebra → Coalgebra → Set
Coalgebra = Σ Set Coalg          Coalgebra⇒ (X , X□) (Y , Y□) = Σ (X → Y) (Coalg⇒ X□ Y□)

ℂoalgebras : Category
ℂoalgebras = record { Obj = Coalgebra ; _⇒_ = Coalgebra⇒ ; ... }
```

The record PCoalg for pointed coalgebras (equipped with a point $\eta : \mathcal{I} \to \mathcal{X}$ that commutes with renaming) and homomorphisms between them are defined similarly.

Substitution The axiomatisation of substitution structure follows the same pattern. We define the *internal substitution hom* as a generalisation of the cofree coalgebra, parametrising a family with an arbitrary substitution rule:

```
⦅_,_⦆ : Fam_S → Fam_S → Fam_S
⦅ 𝒳 , 𝒴 ⦆ α Γ = {Δ : Ctx} → (Γ −[ 𝒳 ]→ Δ) → 𝒴 α Δ
```

The closed structure of families under this hom is proved easily: the structural transformations i : ⦅ 𝒥 , 𝒳 ⦆ → 𝒳, j : 𝒥 → ⦅ 𝒳 , 𝒳 ⦆ and L : ⦅ 𝒴 , 𝒵 ⦆ → ⦅ ⦅ 𝒳 , 𝒴 ⦆ , ⦅ 𝒳 , 𝒵 ⦆ ⦆ are defined by function application and composition. A family equipped with a substitution operation is a monoid in this closed category, with the laws (defined in terms of i, j, L) expanding as:

```
record Mon (ℳ : Fam_S) : Set where
  field η : 𝒥 → ℳ
        μ : ℳ → ⦅ ℳ , ℳ ⦆

        η-μ : {σ : Γ −[ ℳ ]→ Δ}{v : 𝒥 α Γ} → μ[ σ ] (η v) ≡ σ v
        μ-η : {t : ℳ α Γ} → μ[ η ] t ≡ t
        μ-μ : {σ : Γ −[ ℳ ]→ Δ} {ς : Δ −[ ℳ ]→ Θ} {t : ℳ α Γ} →
              μ[ ς ] (μ[ σ ] t) ≡ μ[ μ[ ς ] ∘ σ ] t
```

The multiplication μ represents simultaneous substitution, replacing every variable in context Γ with an $\mathcal{M}$-term in Δ. In practice (e.g. in β-reduction) one often uses one- or two-variable substitution for the last variable or variables in the context, which is derived using add:

```
[_/] : 𝓜 α Γ → 𝓜 β (α · Γ) → 𝓜 β Γ
[ s /] t = μ[ add 𝓜 s η ] t

[_,_/]₂ : 𝓜 α Γ → 𝓜 β Γ → 𝓜 τ (α · β · Γ) → 𝓜 τ Γ
[ s₁ , s₂ /]₂ t = μ[ add 𝓜 s₁ (add 𝓜 s₂ η) ] t
```

The associativity axiom μ-μ generalises the associativity laws of single-variable substitution, often encountered in normalisation theorems: the following instance is derivable for every monoid, where $[\_/]'$ is single-variable substitution for the second variable of the context:

```
subst-lemma : (t : 𝓜γ (β · α · Γ))(s : 𝓜 β (α · Γ))(r : 𝓜 α Γ) →
              [ r /] ([ s /] t) ≡ [ [ r /] s /] ([ r /]' t)
```

Monoid homomorphisms preserve the unit and multiplication. When $\mathcal{M}$ is a family associated with an inductively defined syntax, $\mathcal{N}$ is a model of the syntax, and f is a map $\mathcal{M} \to\!\triangleright \mathcal{N}$, the preservation of multiplication

```
⟨μ⟩ : {σ : Γ –[ 𝓜 ]→ Δ}{t : 𝓜 α Γ} → f (𝓜.μ[ σ ] t) ≡ 𝓝.μ[ f ∘ σ ] (f t)
```

expresses the *semantic substitution lemma*: the interpretation of substitution in the syntax is the composition of interpretations in the model. We give its familiar form for single-variable substitution as an example below. The fact that f commutes with add is established by function extensionality, case analysis on the variable, and the preservation axiom of the monoidal unit $\langle\eta\rangle : \{v : \mathcal{I}\ \alpha\ \Gamma\} \to f\ (\mathcal{M}.\eta\ v) \equiv \mathcal{N}.\eta\ v$.

```
sub-lemma : (s : 𝓜 α Γ)(t : 𝓜 β (α · Γ)) → f (𝓜.[ s /] t) ≡ 𝓝.[ f s /] (f t)
sub-lemma s t = trans ⟨μ⟩ (cong (𝓝.μ (f t)) (ext λ{ new → refl ; (old y) → ⟨η⟩}))
```

Every monoid $\mathcal{M}$ has an underlying pointed $\Box$-coalgebra instance $\mathcal{M}^{\Box}_{*}$: PCoalg. As the freeness proof induces substitution structure using an existing renaming structure (since we need to weaken when substituting under binders), we introduced the notion of a *invariant monoid* to ensure that the existing renaming structure is compatible with the one derived from substitution. The main property of invariant monoid we need is that the strength transformation (defined later) using either pointed coalgebra structure is equivalent.

```
record InvMon (𝓜 : Fam_S) : Set where
  field   𝓜^□_* : PCoalg 𝓜
          𝓜^M : Mon    𝓜
          η-compat :                {v : 𝓘 α Γ}  → η^□_* v  ≡ η^M v
          μ-compat : {ρ : Γ ⇝ Δ} {t : 𝓜 α Γ} → r t ρ ≡ μ t (η^M ∘ ρ)

  str-eq : str 𝓜^□_* 𝓨 ≡ str 𝓜^M.□_* 𝓨
```

Linearity As discussed throughout the mathematical development, parametrised maps in families are more limited than their presheaf counterparts. The simpler monoidal and closed substitution structures in families lack the renaming-invariance properties captured by dinaturality in natural transformations. To address this, we introduce a set of axioms collectively encapsulating *pointed multilinearity* – here simply *linearity* – which encode the necessary coherence laws for maps $\mathcal{X} \to\!\triangleright \langle\!|\, \mathcal{Y}\,,\, \mathcal{Z} \,|\!\rangle$, assuming all three families are pointed coalgebras:

```
record Linear (f : 𝒳 →▷ ⟪ 𝒴 , 𝒵 ⟫) : Set where
  field  r-f : {σ : Γ –[ 𝒴 ]→ Δ}{ϱ : Δ ⇝ Θ} {t : 𝒳 α Γ} → 𝒵.r[ ϱ ] (f t σ) ≡ f t (𝒴.r[ ϱ ] ∘ σ)
         f-r : {ρ : Γ ⇝ Δ}{ς : Δ –[ 𝒴 ]→ Θ} {t : 𝒳 α Γ} → f (𝒳.r[ ρ ] t) ς ≡ f t (ς ∘ ρ)
         f-η :                               {v : ℐ α Γ} → f (𝒳.η v) 𝒴.η   ≡ 𝒵.η v
```

These axioms, spelled out explicitly in Section 10.2.1, capture the structure needed to recover key properties of presheaf models within the family setting. One immediate consequence is that the codomain $\langle\!|\, \mathcal{Y}\,,\, \mathcal{Z} \,|\!\rangle$ becomes a pointed $\Box$-coalgebra, and the map f qualifies as a pointed coalgebra homomorphism. Without the linearity axioms, this generally fails: the internal hom $\langle\!|\, \mathcal{Y}\,,\, \mathcal{Z} \,|\!\rangle$ is not pointed in general, even if both $\mathcal{Y}$ and $\mathcal{Z}$ are. As shown in Lemma 5.2.1, several relevant transformations – such as $\mathsf{j} : \mathcal{I} \to\!\triangleright \langle\!|\, \mathcal{X}\,,\, \mathcal{X} \,|\!\rangle$, $\mathsf{r} : \mathcal{X} \to\!\triangleright \langle\!|\, \mathcal{I}\,,\, \mathcal{X} \,|\!\rangle$ and $\mu : \mathcal{X} \to\!\triangleright \langle\!|\, \mathcal{X}\,,\, \mathcal{X} \,|\!\rangle$ – are all linear, with their laws deriving from the axioms of closed categories, coalgebras, and monoids.

We again compare the definitions above to prior work.

Renaming Our use of coalgebra structure to capture renaming was originally inspired by Allais et al. (2021), whose Thinnable X operator expands to the coalgebra $X \to\!\triangleright \Box X$. While their library defines the operations making $\Box$ a comonad, comonad and coalgebra laws are neither stated nor used. The library also includes a free presheaf monad $\Diamond$ and various structural operators, though these are not discussed in the paper.

The VarLike record is a variant of a pointed coalgebra: it equips a sorted family $\mathcal{X}$ with coalgebra structure (a thinning operation), and a distinguished term new : $\mathcal{X}\ \alpha\ (\alpha \cdot \Gamma)$, from which the point base $(\Gamma$ -Env$)\ \mathcal{X}\ \Gamma = \mathcal{I} \to\!\triangleright \mathcal{X}$ and various weakening operations are defined inductively over lists or renamings.

A similar idea appears in McBride's (2005) *kit* construction, which packages a translation $tm : \blacklozenge\ \alpha\ \Gamma \to \mathbb{T}\ \alpha\ \Gamma$ of "stuff" into syntactic terms, a variable embedding $vr : \mathcal{I}\ \alpha\ \Gamma \to \blacklozenge\ \alpha\ \Gamma$, and a weakening map $wk : \blacklozenge\ \alpha\ \Gamma \to \blacklozenge\ \alpha\ (\beta \cdot \Gamma)$. This concept is further developed in Benton et al. (2012, Section 7).

Substitution Both McBride and Allais et al. derive substitution as a special case of a generic traversal (see later), but again, the structure is not axiomatised and families with substitution structure are not given as much emphasis as families with renaming structure.

Laws and linearity Whereas our work makes coherence axioms an inseparable part of the structure, Allais et al. decouple operations from laws and prove the latter at an ad hoc basis using a separate reasoning framework based on simulation and fusion lemmas. Axioms of coalgebras (ren-id, ren^2), monoids (sub-id, sub^2), and linear maps (rensub, subren) are stated and proved for syntactic terms rather than arbitrary families, with a proof technique that

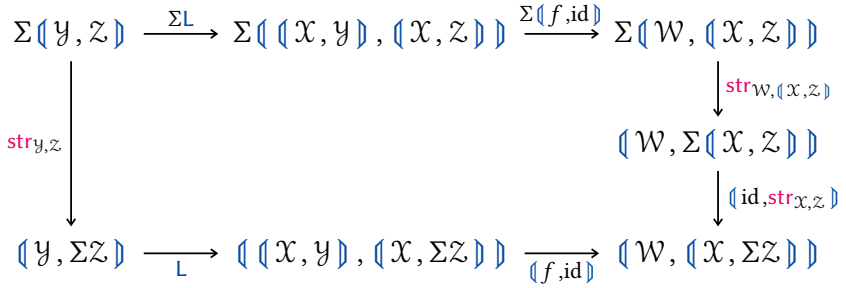
feels quite independent from abstract syntax. By contrast, our approach builds the axioms into the fabric of the structures from the outset, allowing us to appeal to syntactic principles like initiality to establish correctness and compatibility laws systematically, without requiring an external reasoning framework.

12.1.3 Models

Given a family endofunctor $\Sigma : \mathsf{Fam_S} \to \mathsf{Fam_S}$ capturing the signature of a second-order syntax, we defined a model for the syntax to be a substitution monoid with compatible Σ-algebra structure. As we've seen, to express the compatibility condition, we need to be able to push substitution rules into constructors and under binders using the *pointed strength* of Section 10.3.2. The strength for a functor Σ is axiomatised as a record as follows:

$$
\begin{aligned}
&\mathsf{record}\ \mathsf{Strong}\ (\Sigma : \mathsf{Fam_S} \to \mathsf{Fam_S}) : \mathsf{Set}\ \mathsf{where}\\
&\quad \mathsf{field}\quad \mathsf{str} \quad : (\mathcal{X}^{\square}_{*} : \mathsf{PCoalg})(\mathcal{Y}\colon \mathsf{Fam_S}) \to \Sigma\,(\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!) \twoheadrightarrow (\!|\,\mathcal{X}\,,\,\mathsf{F}\,\mathcal{Y}\,|\!)\\
&\qquad\quad \mathsf{str\text{-}n_1} : (f^{\square}_{*}\!\Rightarrow : \mathsf{PCoalg}\!\Rightarrow\, \mathcal{W}^{\square}_{*}\ \mathcal{X}^{\square}_{*}\ f)\ (h : \Sigma\,(\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!)\ \alpha\ \Gamma)\ (\sigma : \Gamma \mathrel{-}[\,\mathcal{W}\,]\!\to \Delta) \to\\
&\qquad\qquad \mathsf{str}\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ h\ (f \circ \sigma) \equiv \mathsf{str}\ \mathcal{W}^{\square}_{*}\ \mathcal{Y}\ (\Sigma_1\ (\lambda\ h'\ \sigma' \to h'\ (f \circ \sigma'))\ h)\ \sigma\\
&\qquad\quad \mathsf{str\text{-}n_2} : (g : \mathcal{Y} \twoheadrightarrow \mathcal{Z})(h : \Sigma\,(\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!)\ \alpha\ \Gamma)(\sigma : \Gamma \mathrel{-}[\,\mathcal{X}\,]\!\to \Delta) \to\\
&\qquad\qquad \mathsf{str}\ \mathcal{X}^{\square}_{*}\ \mathcal{Z}\ (\Sigma_1\ (\lambda\ h'\ \sigma' \to g\ (h'\ \sigma'))\ h)\ \sigma \equiv \Sigma_1\ g\ (\mathsf{str}\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ h\ \sigma)\\
&\qquad\quad \mathsf{str\text{-}i} \quad : (h : \Sigma\,(\!|\,\mathcal{I}\,,\,\mathcal{Y}\,|\!)\ \alpha\ \Gamma) \to \mathsf{str}\ \mathcal{I}^{\square}_{*}\ \mathcal{Y}\ h\ \mathsf{id} \equiv \Sigma_1\ (\mathsf{i}\ \mathcal{Y})\ h\\
&\qquad\quad \mathsf{str\text{-}L} \quad : \{f : \mathcal{W} \twoheadrightarrow (\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!)\}\ (f^{\Sigma} : \mathsf{Linear}\ \mathcal{W}^{\square}_{*}\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}^{\square}_{*}\ f)\\
&\qquad\qquad (h : \Sigma\,(\!|\,\mathcal{Y}\,,\,\mathcal{Z}\,|\!)\ \alpha\ \Gamma)(\sigma : \Gamma \mathrel{-}[\,\mathcal{W}\,]\!\to \Delta)(\varsigma : \Delta \mathrel{-}[\,\mathcal{X}\,]\!\to \Theta) \to\\
&\qquad\qquad\quad \mathsf{str}\ \mathcal{Y}^{\square}_{*}\ \mathcal{Z}\ h\ (\lambda\ v \to f\ (\sigma\ v)\ \varsigma)\\
&\qquad\qquad \equiv \mathsf{str}\ \mathcal{X}^{\square}_{*}\ \mathcal{Z}\ (\mathsf{str}\ \mathcal{W}^{\square}_{*}\ (\!|\,\mathcal{X}\,,\,\mathcal{Z}\,|\!)\ (\Sigma_1\ (\lambda\ h\ \sigma\ \varsigma \to h\ (\lambda\ v \to f\ (\sigma\ v)\ \varsigma)\ h)\ \sigma)\ \varsigma
\end{aligned}
$$

Though they look rather complicated, the equations correspond the naturality and strength axioms of synthetic strengths. In particular, str-L expresses the compatibility of strength with the parametrised compositor $\mathsf{L}[f] : (\!|\,\mathcal{Y}\,,\,\mathcal{Z}\,|\!) \to (\!|\,\mathcal{W}\,,\,(\!|\,\mathcal{X}\,,\,\mathcal{Z}\,|\!)\,|\!)$, a closed form of Diagram ($s\alpha\oslash L$).



The most important instance of strong functors is the context extension δ, whose strength uses the well-known auxiliary lift operation on substitution rules over pointed coalgebras $\mathcal{X}$:

$$
\begin{aligned}
&\mathsf{lift} : (\Theta : \mathsf{Ctx}) \to (\Gamma \mathrel{-}[\,\mathcal{X}\,]\!\to \Delta) \to (\Theta + \Gamma) \mathrel{-}[\,\mathcal{X}\,]\!\to (\Theta + \Delta)\\
&\mathsf{lift}\ \emptyset \qquad \sigma\ v \qquad\quad = \sigma\ v\\
&\mathsf{lift}\ (\tau \cdot \Theta)\ \sigma\ \mathsf{new} \quad = \mathcal{X}.\eta\ \mathsf{new}\\
&\mathsf{lift}\ (\tau \cdot \Theta)\ \sigma\ (\mathsf{old}\ v) = \mathcal{X}.\mathsf{r}\ (\mathsf{lift}\ \Theta\ \sigma\ v)\ \mathsf{old}
\end{aligned}
$$

The Strong instance for context extension requires lift and various derived lemmas thereof, all ultimately corresponding to the properties listed in Section 2.1.3.

```
δ:Str : (Ξ : Ctx) → Str (δ:Functor Ξ)
δ:Str Ξ = record
  { str    = λ 𝒳*□ 𝒴 h σ    → h (lift 𝒳 σ)
  ; str-n₁ = λ f*□⇒ h σ     → cong h (lift-n₁ Ξ f*□⇒ σ)
  ; str-n₂ = λ f    h σ     → refl
  ; str-i  = λ 𝒴    h       → cong h (lift-i Ξ 𝒴)
  ; str-L  = λ 𝒴 fΣ h σ ς   → cong h (lift-L Ξ 𝒴 f σ ς) }
```

To show how lifting, closed structure, and linear maps interact, we spell out the proof of lift-L:

```
lift-L : (Ξ : Ctx) {f : 𝒲 ↛ ⟪ 𝒳 , 𝒴 ⟫} (fΣ : Linear 𝒲*□ 𝒳 𝒴 f)
         (σ : Γ –[ 𝒲 ]→ Δ) (ς : Δ –[ 𝒳 ]→ Θ)(v : ℐ α (Ξ + Γ)) →
         lift 𝒴 (λ u → f (σ u) ς) v  ≡  f (lift 𝒲 σ v) (lift 𝒳 ς)

lift-L ∅       fΣ σ ς v       = refl

lift-L (β · Ξ) fΣ σ ς new     = begin
  𝒴.η new                  ≡˘⟨ f-η ⟩
  f (𝒲.η new) 𝒳.η          ≡⟨ eq-at-new refl ⟩
  f (𝒲.η new) (lift 𝒳 ς) ■

lift-L (β · Ξ) fΣ σ ς (old v) = begin
  𝒴.r[ old ] (lift 𝒴 (λ u → f (σ u) ς) v)  ≡⟨ lift-L Ξ fΣ σ ς v⟩
  𝒴.r[ old ] (f (lift 𝒲 σ v) (lift 𝒳 ς))  ≡⟨ r-f ⟩
  f (lift 𝒲 σ v) (𝒳.r[ old ] ∘ lift 𝒳 ς)  ■
```

The proof proceeds by induction on the extension context and the variable. When looking at the top variable, we use the fact that linear maps are point-preserving, and propagate the property of two substitution rules equalling at the first variable to the equality of applications of the map to the point of $\mathcal{X}$ at the first variable:

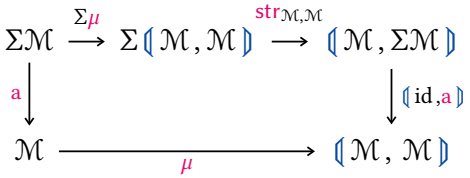
```
eq-at-new : {σ : (α · Γ) –[ 𝒴 ]→ (α · Θ)} {ς : (α · Δ) –[ 𝒴 ]→ (α · Θ)} →
            σ new ≡ ς new → f (𝒳.η new) σ ≡ f (𝒳.η new) ς
```

Indeed, this key property is inlined in the proof of Lemma 10.3.2 and proved using the linearity axioms. The recursive case of lift-L again proceeds using axioms of linear maps, justifying the need for the abstraction. To show how lift-L generalises several lifting lemmas in one fell swoop, we apply it to the linear maps j ℐ, j 𝒳, r and μ to obtain the familiar compatibility conditions of lift with composition of renaming and substitution rules, listed in Section 2.1.3.

```
lift 𝓘 (ϱ ∘ ρ)       ≡ lift 𝓘 ϱ ∘ lift 𝓘 ρ
lift 𝓧 (ς ∘ ρ)       ≡ lift 𝓧 ς ∘ lift 𝓘 ρ
lift 𝓧 (r[ ϱ ] ∘ σ) ≡ r[ lift 𝓘 ϱ ] ∘ lift 𝓧 σ
lift 𝓧 (μ[ ς ] ∘ σ) ≡ μ[ lift 𝓧 ς ] ∘ lift 𝓧 σ
```

With the appropriate definition of strength, we can define our algebraic models: monoids with compatible F-algebra structure. For the initial model – the syntax – the compatibility condition axiomatises the structurally recursive definition of substitution on syntactic terms.

```
record _-Mon (Σ : Fam_S → Fam_S) (𝓜 : Fam_S) : Set where
  field 𝓜ᴹ : Mon 𝓜
        a  : Σ 𝓜 –→ 𝓜
        μ⟨a⟩ : {σ : Γ –[ 𝓜 ]→ Δ}{t : Σ 𝓜 α Γ} →
               μ (a t) σ ≡ a (str 𝓜*□ 𝓜 (Σ₁ μ t) σ)
```



While no previous research on formalisation frameworks axiomatises models with substitution like we do, the associated pattern nevertheless appear.

Lifting and strength The lift operation, which extends the domain and codomain of a substitution rule by introducing additional variables, has long been recognised as a crucial abstraction – dating back at least to Altenkirch and Reus (1999) in the context of monadic syntax. In their setting, the lift operation appears naturally as part of the monad laws, with the unit and associativity properties ensuring its correctness; however, these properties are not explicitly abstracted or formalised as a separate concept. In contrast, Bird and Paterson (1999[a]) explore a more basic variant of lifting, using a swapping operation sw : T 𝓧 + 1 → T (𝓧 + 1) in their work on nested datatypes and generalised folds.

Later developments make more explicit use of lifting as a generic traversal mechanism. McBride (2005) and Benton et al. (2012) adopt the lift pattern to systematically define type-preserving and structurally generic transformations. In more recent work, Allais et al. (2021) incorporate a variant under the name extend$_S$, specialised to λ-terms, and a generalised form for arbitrary families is given as lift. These usages highlight the continuing relevance of lifting not just as a programming convenience, but as a structurally important component in the formal treatment of syntax and substitution.

Laws The laws for lift-like operations naturally arise as part of proving substitution laws for the family or monad of terms. Altenkirch and Reus (1999) explicitly state the monad axioms and the interaction laws between lifting, functoriality, and substitution, though they stop short of isolating the lifting lemmas in a formal setting of their own. In contrast, Bird and Paterson (1999[a], Section 3.3) axiomatise their swap-like operation sw as a distributive law when establishing monadic structure, taking a more categorical route.

A particularly detailed account is given in Benton et al. (2012), whose practical guide to intrinsically-typed formalisation in type theory lays out an extensive suite of lemmas required to endow the family of terms with lawful substitution structure. Their comprehensive to-do list of identity–renaming–variable–lifting–substitution interaction laws in Section 4

(and summarised in Section 2.1.3) vividly illustrates the laborious work typically needed to formalise syntax and substitution from scratch.

Our framework offers a sharp contrast: not only does the str-L law generalise all four key lifting-associativity properties (cf. the corresponding instantiations `LiftRcR`, `LiftScR`, `LiftRcS`, and `LiftScS`), but it does so generically for any signature endofunctor, with no additional user effort. This dramatically reduces boilerplate and shifts the burden of proof from the user to the general theory.

A similar goal motivates the work of Allais et al. (2021), who introduce the generic Simulation and Fusion constructions. These are designed to derive correctness proofs of traversal equivalences and compositions from minimal assumptions. Inspired by Kripke semantics and logical relations, their framework potentially supports reasoning about a wide variety of traversal behaviours (see their Section 7).

However, the examples they do provide concern only standard syntactic identities involving renaming and substitution – exactly the kind of properties we show can be proved purely syntactically, using initiality and structured substitution. Moreover, while powerful, the simulation and fusion abstractions are quite complex and tend to obscure the underlying syntactic regularities – especially those related to lifting – which in our framework emerge directly and naturally. As a result, their approach may come across less as a principled semantic design and more as an elaborate workaround for missing syntactic structure.

A common challenge in formalising lawful substitution structures is the tangled dependency between definitions and axioms, which hinders modular, bottom-up development. Ideally, simpler operations would be fully defined and axiomatised before being used to define more complex ones. In practice, however, axioms for a "primitive" operation often depend on the definition of "derived" ones. For example, Bird and Paterson (1999[a]) define sw as a distributive law between the functor $(-) + 1$ and the term monad $\mathbb{T}$. But sw is used in the definition of join, which itself is needed to even state the monad–functor interaction laws required to prove the monad laws for $\mathbb{T}$. This circular dependency makes clean separation difficult.

Similar issues arise in intrinsically-typed formalisations, where renaming, lifting, and substitution are deeply interdependent. To manage this, many developments adopt a two-phase approach: define operations first, then prove the laws. But this clashes with our goal of packaging related concepts – operations and axioms – into coherent Agda records.

Agda's records don't permit interleaved definitions: one can't define some fields in one record (e.g. lifting), use them in another (e.g. substitution), and return to the original record later (e.g. lifting-substitution laws). We address this by formulating strength associativity over arbitrary linear maps, eventually instantiated with application, renaming, or substitution. This generality allows us to untangle the axiomatisation, supporting modular, idiomatic formalisation without premature commitments to concrete operations.

12.2 Initial algebra semantics

The true value of the familial model lies in placing intrinsically-typed formalisation of syntaxes on a rigorous mathematical foundation. This, in turn, enables the use of categorical techniques – such as those provided by the `agda-categories` library – as practical tools for

reasoning. Our framework is a strong demonstration of this idea: we prove the free algebraic monoid theorem by assuming the existence of an initial $\Sigma_{\mathfrak{N}}$-algebra via a parametrised Agda module, and then carry out the proof from Section 11.2 almost verbatim using the library's axiomatisation of initial objects.

12.2.1 Free algebraic monoid structure

To be syntax-generic, we fix a signature endofunctor Σ and a metavariable family $\mathfrak{N}$. A Σ-algebra structure map on $\mathcal{A}$ captures the constructors of a second-order syntax, whereas the variable and metavariable constructors are encoded as $\mathtt{v} : \mathcal{I} \to\!\triangleright \mathcal{A}$ and $\mathtt{m} : \mathfrak{N} \to\!\triangleright \langle\!|\, \mathcal{A}, \mathcal{A} \,|\!\rangle$. For example, the term $\mathtt{m}[t, s] : \mathcal{A}\,\tau\,\Gamma$ for a metavariable $\mathfrak{a} : \mathfrak{N}\,\tau\,[\alpha, \beta]$, and terms $t : \mathcal{A}\,\alpha\,\Gamma$ and $s : \mathcal{A}\,\beta\,\Gamma$ is represented by $\mathtt{m}\ \mathfrak{a}\ \lambda\{\ \mathtt{new} \to t\ ;\ \mathtt{old}\ \mathtt{new} \to s\ \}$. Families that support this structure will be called syntactic algebras or $\Sigma_{\mathfrak{N}}$-*algebras*, with the expected notion of homomorphism.

```
record SynAlg (𝒜 : Fam𝕊) : Set where
  field  a : Σ 𝒜 →▹ 𝒜
         v : 𝓘 →▹ 𝒜
         m : 𝔑 →▹ ⟨| 𝒜 , 𝒜 |⟩
```

```
record SynAlg⇒ (𝒜ˢ : SynAlg 𝒜) (ℬˢ : SynAlg ℬ) (f : 𝒜 →▹ ℬ) : Set where
  field  ⟨a⟩ : {t : Σ 𝒜 α Γ}                  → f (𝒜.a t)     ≡ ℬ.a (Σ f t)
         ⟨v⟩ : {v : 𝓘 α Γ}                    → f (𝒜.v v)     ≡ ℬ.v v
         ⟨m⟩ : {𝔞 : 𝔑 α Π}{ε : Π –[ 𝒜 ]→ Γ} → f (𝒜.m 𝔞 ε) ≡ ℬ.m 𝔞 (f ∘ ε)
```

Such syntactic algebras and their homomorphisms form a category **SynAlg**, whose initial object – whenever it exists – will be denoted $\mathbb{T}\,\mathfrak{N}$ (or just $\mathbb{T}$, if the metavariable family is clear from the context) with structural maps $\mathfrak{a}$, $\mathbb{v}$ and $\mathbb{m}$. The universal property of initial objects states that there is a unique syntactic algebra homomorphism $\mathbb{i} : \mathbb{T} \to\!\triangleright \mathcal{A}$ into any syntactic algebra $\mathcal{A}$. In Agda, this translates directly to parametrising the metatheory modules with a variable of type Initial 𝕊ynAlgebras, the initial object of the category of syntactic algebras and homomorphisms. The Initial record of the `agda-categories` library exposes fields for the initial object $\bot$ (renamed to $\mathbb{T}$), the initial morphism ! (renamed to $\mathbb{i}$), and the uniqueness proof !-unique (used to derive the equality operator eq that equates two syntactic algebra homomorphisms from $\mathbb{T}$ into the same object).

Varying $\mathfrak{N}$, the term monad $\mathbb{T}$ maps each family $\mathfrak{N}$ to the initial $\Sigma_{\mathfrak{N}}$-algebra $\mathbb{T}, \mathfrak{N}$. In this section, we sketch the Agda proof of Theorem 11.2.2, establishing that $\mathbb{T}$ is the free Σ-monoid functor on sorted families. Since the initial $\Sigma_{\mathfrak{N}}$-algebra corresponds to the inductively defined syntax of terms, the theorem formally confirms that the syntactic structure alone determines the action and laws of substitution. This suggests that much of syntactic metatheory – renaming, lifting, substitution lemmas, etc. – amounts to uninteresting boilerplate. By using initial algebra semantics, such structure and its associated correctness properties can be derived automatically, allowing us to focus on deeper properties of the language from the outset.

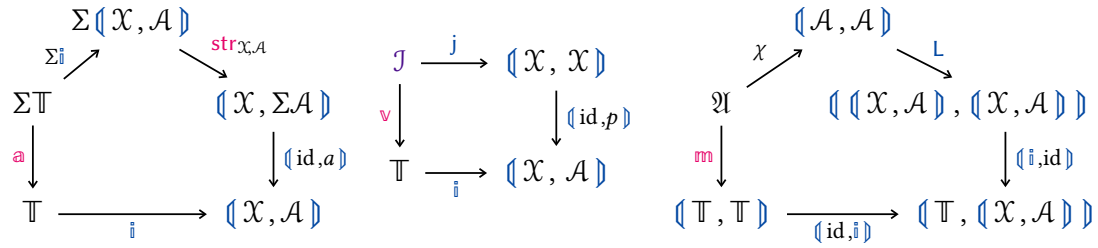
Traversal Our internalisation of syntactic operations at the level of sorted families is justified by the initial algebra approach. Since these operations are defined as maps from the initial syntactic algebra, we can derive renaming $\mathbb{T} \to \Box\mathbb{T}$ and substitution $\mathbb{T} \to (\!|\, \mathbb{T}, \mathbb{T} \,|\!)$ by endowing their codomains – $\Box\mathbb{T}$ and $(\!|\, \mathbb{T}, \mathbb{T} \,|\!)$ – with suitable syntactic algebra structures.

Lemma 12.2.1 *Given a pointed $\Box$-coalgebra $\mathcal{X}$, a Σ-algebra $\mathcal{A}$, and family maps $\varphi : \mathcal{X} \to \mathcal{A}$ and $\chi : \mathfrak{N} \to (\!|\, \mathcal{A}, \mathcal{A} \,|\!)$, the internal hom $(\!|\, \mathcal{X}, \mathcal{A} \,|\!)$ has a $\Sigma_{\mathfrak{N}}$ algebra structure.*

The proof is encapsulated in the Traversal module, instantiations of which will give rise to homomorphic initial algebra interpretations $\mathbb{T} \to (\!|\, \mathcal{X}, \mathcal{A} \,|\!)$. It crucially relies Σ being Strong when defining the structure map $\Sigma\ (\!|\, \mathcal{X}, \mathcal{A} \,|\!) \to (\!|\, \mathcal{X}, \Sigma\ \mathcal{A} \,|\!) \to (\!|\, \mathcal{X}, \mathcal{A} \,|\!)$. A simple corollary (derived by instantiating $\mathcal{X}$ with $\mathcal{I}$) is that if $\mathcal{A}$ is a syntactic algebra, then so is $\Box\,\mathcal{A}$.

```
module Traversal (𝒳* : PCoalg 𝒳) (a : Σ𝒜 → 𝒜) (p : 𝒳 → 𝒜) (χ : 𝔑 → (|𝒜 , 𝒜|)) where
  Travˢ : SynAlg (|𝒳 , 𝒜|)
  Travˢ = record { a = λ h   σ → a (str 𝒳* 𝒜 h σ)
                 ; v = λ v   σ → p (σ v)
                 ; m = λ α ε σ → χ α (λ v → ε v σ) }
```

An initial interpretation into a hom will be called a $\mathcal{Y}$-*parametrised traversal* into $\mathcal{A}$ and written $\mathbb{t} : \mathbb{T} \to (\!|\, \mathcal{X}, \mathcal{A} \,|\!)$. The syntactic homomorphism properties simplify to the following diagrams, which also constitute the proof goal when equating two maps $f, g : \mathbb{T} \to (\!|\, \mathcal{X}, \mathcal{A} \,|\!)$: if they are both $\Sigma_{\mathfrak{N}}$-algebra homomorphisms, they must be equal.



One might be tempted to immediately define substitution $\mathbb{T} \to (\!|\, \mathbb{T}, \mathbb{T} \,|\!)$ as a $\mathbb{T}$-parametrised traversal into $\mathbb{T}$. However, this requires $\mathbb{T}$ to first carry a pointed $\Box$-coalgebra structure, which we have yet to construct. Our formalism makes this dependency explicit: substitution relies on renaming, and the structure of the proof naturally dictates the correct order of construction. By the time we state the theorem, all necessary lemmas are already in place – avoiding the usual search for ad hoc auxiliary results. The sequence of constructions unfolds as follows:

1. Prove that traversals $\mathbb{t} : \mathbb{T} \to (\!|\, \mathcal{X}, \mathcal{A} \,|\!)$ applied to the point $\eta : \mathcal{I} \to \mathcal{A}$ are equal to interpretations $\mathbb{i} : \mathbb{T} \to \mathcal{A}$, assuming p is point-preserving (Lemma 11.2.4).
2. Induce the renaming operator $\mathbb{r} : \mathbb{T} \to \Box\mathbb{T}$ as an $\mathcal{I}^{\Box}_{*}$-parametrised traversal into $\mathbb{T}$ and prove the pointed coalgebra laws to get an instance $\mathbb{T}^{\Box}_{*} : \mathrm{PCoalg}$ (Proposition 11.2.3).

   The r-id law is an instance of traversal-interpretation equivalence above:

$$\mathbb{T} \xrightarrow{\mathbb{r}} \Box\mathbb{T} \xrightarrow{\mathbb{i}} \mathbb{T} \;=\; \mathbb{T}$$

The comultiplication law is the equality of two parametrised maps $\mathbb{T} \rightarrow\!\triangleright \langle\!| \mathcal{I}, \langle\!| \mathcal{I}, \mathcal{A} |\!\rangle |\!\rangle$; showing both to be $\Sigma_{\mathfrak{R}}$-algebra homomorphisms is sufficient to establish their equality by uniqueness. The main complexity in proofs like this is the preservation of Σ-algebra structure, but using strength axioms – namely str-L – the equational reasoning is quite formulaic. For example, the Σ-algebra preservation of the composite (with the right side implementing the comonad comultiplication of $\square$)

$$\mathbb{T} \xrightarrow{\mathbb{r}} \square\mathbb{T} \xrightarrow{\mathsf{L}} \langle\!| \square\mathcal{I}, \square\mathbb{T} |\!\rangle \xrightarrow{\langle\!| \mathsf{j}\,\mathcal{I}, \mathrm{id} |\!\rangle} \square\square\mathbb{T}$$

calculates as follows, using the linearity of $\mathsf{j}\,\mathcal{I} : \mathcal{I} \rightarrow\!\triangleright \langle\!| \mathcal{I}, \mathcal{I} |\!\rangle$:

$$\begin{array}{ll}
\mathbb{r}\ (\mathbb{a}\ t)\ (\varrho \circ \rho) & \equiv\langle\ \langle \mathbb{a} \rangle\ \rangle \\
\mathbb{a}\ (\mathsf{str}\ \mathcal{I}^{\square}_{*}\ \mathbb{T}\ (\Sigma_1\ \mathbb{r}\ t)\ (\varrho \circ \rho)) & \equiv\langle\ \text{str-L}\ \mathbb{T}\ (\mathsf{j}^{\Sigma}\ \mathcal{I}^{\square}_{*})\ \rangle \\
\mathbb{a}\ (\mathsf{str}\ \mathcal{I}^{\square}_{*}\ \mathbb{T}\ (\mathsf{str}\ \mathcal{I}^{\square}_{*}\ (\square\ \mathbb{T})\ (\Sigma_1\ (\lambda\ h\ \rho\ \varrho \rightarrow h\ (\varrho \circ \rho))\ (\Sigma_1\ \mathbb{r}\ t))\ \rho)\ \varrho) & \equiv\langle\ \Sigma.\text{pres-}\circ\ \rangle \\
\mathbb{a}\ (\mathsf{str}\ \mathcal{I}^{\square}_{*}\ \mathbb{T}\ (\mathsf{str}\ \mathcal{I}^{\square}_{*}\ (\square\ \mathbb{T})\ (\Sigma_1\ (\lambda\ t\ \rho\ \varrho \rightarrow \mathbb{r}\ t\ (\varrho \circ \rho))\ t)\ \rho)\ \varrho) & \blacksquare
\end{array}$$

3. Prove that for a $\Sigma_{\mathfrak{R}}$-algebra $\mathcal{A}$ with a $\Sigma_{\mathfrak{R}}$-homomorphic coalgebra structure $r : \mathcal{A} \rightarrow\!\triangleright \square\mathcal{A}$, the interpretation map $\mathbb{i} : \mathbb{T} \rightarrow\!\triangleright \mathcal{A}$ is a $\square$-coalgebra homomorphism (Proposition 11.2.4)
4. Prove that a traversal $\mathbb{t} : \mathbb{T} \rightarrow\!\triangleright \langle\!| \mathcal{X}, \mathcal{A} |\!\rangle$ is a Linear map assuming $\mathcal{A}$ is a pointed $\square$-coalgebra and a and φ are pointed $\square$-coalgebra homomorphisms (Proposition 11.2.5).

   One of the linearity axioms is just a coalgebra homomorphism condition, while the other one has the same structure as the coalgebra comultiplication law, but with $r : \mathcal{A} \rightarrow\!\triangleright \square\mathcal{A}$ as the linear map.
5. Induce the substitution operator $\mathbb{s} : \mathbb{T} \rightarrow\!\triangleright \langle\!| \mathbb{T}, \mathbb{T} |\!\rangle$ as a $\mathbb{T}$-parametrised traversal into $\mathbb{T}$, and prove the monoid laws to get an instance of $\mathbb{T}^{\mathrm{M}} : \mathsf{Mon}$ (Proposition 11.2.6).

   The unit and associativity laws follow a similar pattern to coalgebra and linearity laws. Just as McBride (2005) recognised the similarity of the renaming and substitution operations, we can establish a clear similarity between their laws. Compare one half of the monoid associativity law to the comultiplication law above:

$$\begin{array}{ll}
\mathbb{s}\ (\mathbb{a}\ t)\ (\mathbb{s}[\,\varsigma\,] \circ \sigma) & \equiv\langle\ \langle \mathbb{a} \rangle\ \rangle \\
\mathbb{a}\ (\mathsf{str}\ \mathbb{T}^{\square}_{*}\ \mathbb{T}\ (\Sigma_1\ \mathbb{s}\ t)\ (\mathbb{s}[\,\varsigma\,] \circ \sigma)) & \equiv\langle\ \text{str-L}\ \mathbb{T}\ \mathbb{s}^{\Sigma}\ \rangle \\
\mathbb{a}\ (\mathsf{str}\ \mathbb{T}^{\square}_{*}\ \mathbb{T}\ (\mathsf{str}\ \mathbb{T}^{\square}_{*}\ \langle\!| \mathbb{T}, \mathbb{T} |\!\rangle\ (\Sigma_1\ (\lambda\ h\ \sigma\ \varsigma \rightarrow h\ (\mathbb{s}[\,\varsigma\,] \circ \sigma))\ (\Sigma_1\ \mathbb{s}\ t))\ \sigma)\ \varsigma) & \equiv\langle\ \Sigma.\text{pres-}\circ\ \rangle \\
\mathbb{a}\ (\mathsf{str}\ \mathbb{T}^{\square}_{*}\ \mathbb{T}\ (\mathsf{str}\ \mathbb{T}^{\square}_{*}\ \langle\!| \mathbb{T}, \mathbb{T} |\!\rangle\ (\Sigma_1\ (\lambda\ t\ \sigma\ \varsigma \rightarrow \mathbb{s}\ t\ (\mathbb{s}[\,\varsigma\,] \circ \sigma))\ t)\ \sigma)\ \varsigma) & \blacksquare
\end{array}$$

   The only difference between the proofs is the choice of linear map; the Linear instance $\mathbb{s}^{\Sigma}$ is derived using the lemma in step 4.
6. Prove that $\mathbb{T}$ is an invariant monoid and a Σ-monoid (Proposition 11.2.6).

   The renaming-substitution compatibility condition

$$\mathbb{r}\ t\ \rho \equiv \mathbb{s}\ t\ (\mathbb{v} \circ \rho)$$

   derives from the lemma in step 1 and the linearity of $\mathbb{s}$. This is then needed to show that $\mathbb{T}$ is a Σ-monoid: the homomorphism condition $\langle \mathbb{a} \rangle$ and the substitution-algebra compatibility condition in Σ-Mon differ in the pointed $\square$-coalgebra instance that the strength operates over, but this is precisely addressed by str-eq:

```
𝕤 (𝕒 t) σ                    ≡⟨ ⟨𝕒⟩ ⟩
𝕒 (str 𝕋□* 𝕋 (Σ₁ 𝕤 t) σ)     ≡⟨ InvMon.str-eq ⟩
𝕒 (str 𝕋ᴹ 𝕋 (Σ₁ 𝕤 t) σ)      ■
```

7. Prove that for any other Σ-monoid $\mathcal{M}$ and map $\omega : \mathfrak{X} \to\!\triangleright \mathcal{M}$, there is a unique Σ-monoid homomorphism $\mathbb{T}\,\mathfrak{X} \to\!\triangleright \mathcal{M}$ (Proposition 11.2.6).

   The initial extension and its homomorphism property are derived by equipping $\mathcal{M}$ with a metavariable operator $\mathfrak{X} \xrightarrow{\omega} \mathcal{M} \xrightarrow{\mu} (\!|\mathcal{M}, \mathcal{M}|\!)$. As the extension is not a traversal map, the Σ-algebra homomorphism proofs involved will not use str-L, only its naturality conditions.

An even more efficient proof technique (implemented after the publication of the POPL 2022 paper) is to follow the proof as laid out in Section 11.2.2 verbatim: reasoning by explicitly constructing and composing $\Sigma_{\mathfrak{X}}$-algebra homomorphisms, rather than repeating slight variations of the same calculation. With the homomorphic reasoning framework implemented in the library, the coalgebra and monoid instances for $\mathbb{T}$ are strikingly concise[1]:

```
𝕋□ : Coalg 𝕋
𝕋□ = record
  { r = 𝕣 ; r-id = 𝕥∘η≈𝕤
  ; r-∘ = ≈₂ 𝕋ᵃ (j 𝕁)
  (|– 𝕋ᵃ          –| 𝕣ᵃ⇒          |⟶
      □ᵃ 𝕋ᵃ       –| □¹ᵃ 𝕣ᵃ⇒       |⟶
      □ᵃ (□ᵃ 𝕋ᵃ)                  |■)
  (|– 𝕋ᵃ          –| 𝕣ᵃ⇒          |⟶
      □ᵃ 𝕋ᵃ       –| L[ j^Σ 𝕁*□ ]ᵃ⇒ |⟶
      □ᵃ (□ᵃ 𝕋ᵃ)                  |■) }
```

```
𝕋ᴹ : Mon 𝕋
𝕋ᴹ = record
  { η = 𝕧 ; μ = 𝕤 ; η-μ = ⟨𝕧⟩ ; μ-η = 𝕥∘η≈𝕤 ○ 𝕤≈id
  ; μ-μ = ≈₂ 𝕋ᵃ 𝕤
  (|– 𝕋ᵃ                –| 𝕤ᵃ⇒              |⟶
      (𝕋□*, 𝕋ᵃ) 𝕋ᵃ      –| (𝕋□*, 𝕤ᵃ⇒)ᵃ      |⟶
      (𝕋□*, (𝕋□*, 𝕋ᵃ))                     |■)
  (|– 𝕋ᵃ                –| 𝕤ᵃ⇒              |⟶
      (𝕋□*, 𝕋ᵃ)         –| L[ 𝕤^Σ ]ᵃ⇒       |⟶
      (𝕋□*, (𝕋□*, 𝕋ᵃ))                     |■) }
```

Rather than splitting the homomorphism conditions into three pairs of equality laws, we use combinators to lift the components of the diagrams into $\Sigma_{\mathfrak{X}}$-homomorphisms, which are glued together as composites in $\Sigma_{\mathfrak{X}}$-**Alg**. For example, the $\Box^{a} : \mathrm{SynAlg}\ \mathcal{A} \to \mathrm{SynAlg}\ (\Box\mathcal{A})$ lifts $\Box$ to syntactic algebras, and $\Box^{a}_{1}$ is the corresponding functorial mapping on homomorphisms. The $\mathsf{L}[\_]^{a}\Rightarrow$ operator encapsulates the homomorphism law of the (parametrised) compositor L, proving that the composite below is a $\Sigma_{\mathfrak{X}}$-algebra homomorphism for every pointed linear map $f : \mathcal{W} \to\!\triangleright (\!|\mathcal{X}, \mathcal{Y}|\!)$ (Lemma 11.2.3).

$$(\!|\mathcal{Y}, \mathcal{A}|\!) \xrightarrow{\mathsf{L}} (\!|(\!|\mathcal{X}, \mathcal{Y}|\!), (\!|\mathcal{X}, \mathcal{A}|\!)|\!) \xrightarrow{(\!|f, \mathrm{id}|\!)} (\!|\mathcal{W}, (\!|\mathcal{X}, \mathcal{A}|\!)|\!)$$

Whichever approach we use, the free Σ-monoid structure for the term monad $\mathbb{T}$ is constructed in an efficient and canonical way, with the axiomatised records guiding the development. It is worth reiterating that at no point did we make any assumptions about the syntax, only that its corresponding signature endofunctor is pointed-strong. We can further formalise the second-order features of the metatheory.

[1] There is a slight lie here in that the applications of Grothendieck re-indexing maps as in Proposition 11.2.2 are not shown, but these do not complicate matters all that much.

12.2.2 Second-order features

Unlike any other formalisation framework we are aware of, ours supports parametrised metavariables out of the box. The intricate metasubstitution operation is constructed using the same ideas presented in Section 11.3, using an abstract formalisation of the main results of Chapter 4, namely monad morphisms into the powered clone monad. Upon instantiating the theory with a proof that $\mathbb{T}$ is a powered monad with powering $\mathbb{T}(X \multimap \mathcal{Y}) \to (X \multimap \mathbb{T}\mathcal{Y})$, we immediately obtain the meta-extension and metasubstitution maps

$$\mathtt{me} : \mathbb{T}\mathfrak{A} \to [\, \mathfrak{A} \circ\!\!- \mathcal{M} \,] \multimap \mathcal{M} \qquad \text{and} \qquad \mathtt{ms} : \mathbb{T}\mathfrak{A} \to [\, \mathfrak{A} \circ\!\!- \mathbb{T}\mathfrak{B} \,] \multimap \mathbb{T}\mathfrak{B}$$

Equipping $\mathbb{T}$ with a powering over $\multimap$ and proving the strong monad laws is done by initiality, in departure from proofs by freeness used in Section 11.3.1 whose higher level of abstraction lead to slow typechecking times. The central result is that given a parameter W and a $\Sigma_{\mathfrak{A}}^{\mathcal{X}}$-algebra $(\mathcal{A}, v, a, m)$, $W \multimap \mathcal{A}$ is a $\Sigma_{W \multimap \mathfrak{A}}^{W \multimap \mathcal{X}}$-algebra, with pointed algebra structure given in terms of the powering of Σ, and metavariable operator given by the following composite, where dist is the closed version of the d operation of Proposition 10.3.8:

$$W \multimap \mathfrak{A} \xrightarrow{W \multimap m} W \multimap (\!|\, \mathcal{A}, \mathcal{A} \,|\!) \xrightarrow{\text{dist}} (\!|\, W \multimap \mathcal{A}, W \multimap \mathcal{A} \,|\!)$$

In particular, for $\mathcal{X} \triangleq \mathcal{I}$, and $\mathcal{A} \triangleq \mathbb{T}\mathfrak{A}$, we have that $W \multimap \mathbb{T}\mathfrak{A}$ is a $\Sigma_{W \multimap \mathfrak{A}}$-algebra so, by initiality, we get a $\Sigma_{W \multimap \mathfrak{A}}$-algebra homomorphism

$$\mathtt{p} : \mathbb{T}(W \multimap \mathfrak{A}) \to (W \multimap \mathbb{T}\mathfrak{A})$$

The powered monad proofs proceed by equating maps out of $\mathbb{T}\mathfrak{A}$ for various choices of $\mathfrak{A}$. Some bookkeeping is required to align metavariable families across domain and codomain: a morphism $\mathbb{T}\mathfrak{A} \to \mathbb{T}\mathfrak{B}$ can be shown to be a $\Sigma_{\mathfrak{A}}$-algebra homomorphism provided we have a map $f : \mathfrak{A} \to \mathfrak{B}$ such that $\mathbb{T}\mathfrak{B}$ acquires a compatible $\Sigma_{\mathfrak{A}}$-algebra structure. Nevertheless, the proofs are fairly straightforward and closely resemble those using freeness in Section 11.3.1. For instance, if $\mathcal{A}$ is a $\Sigma_{\mathfrak{A}}^{\mathcal{X}}$-algebra, we can show that the transformation dist : $W \multimap (\!|\, \mathcal{X}, \mathcal{A} \,|\!) \to (\!|\, W \multimap \mathcal{X}, W \multimap \mathcal{A} \,|\!)$ is a $\Sigma_{W \multimap \mathfrak{A}}$-algebra homomorphism. The proof relies on the synthetic associativity of dist (cf. Diagram ($s\alpha\oslash L$)) and the strength–powering compatibility for Σ (cf. Diagram (sdp)), paralleling the argument in Theorem 10.3.5 on lifting $W \multimap$ to a functor on Σ-monoids.

A major benefit of implementing the abstract theory is that all operations – renaming, substitution, and metasubstitution – are fully computable, regardless of whether they are defined directly by recursion, by initiality, or via freeness. The example in Section 2.2 (*Metasubstitution*) illustrates how metasubstitution both computes and interacts with other syntactic operations. Agda adds further value by enabling a more convenient interface and syntactic sugar, particularly for equational reasoning. The equational logic developed in Section 11.3.2 can be internalised as an inductive data type: a relation Axiom : $\forall(\mathfrak{A}\ \Gamma\ \alpha) \to \mathbb{T}, \mathfrak{A}\ \alpha\ \Gamma \to \mathbb{T}, \mathfrak{A}\ \alpha\ \Gamma \to$ Set, which can be extended to a sound equational theory by defining the corresponding equivalence relation:

```
data _▷_⊢_≈_ : (𝔄 : Fam_S)(Γ : Ctx){α : S} → 𝕋 𝔄 α Γ → 𝕋 𝔄 α Γ → Set₁ where
  eq  : t ≡ s → 𝔄 ▷ Γ ⊢ t ≈ s
  sy  : 𝔄 ▷ Γ ⊢ t ≈ s → 𝔄 ▷ Γ ⊢ s ≈ t
  tr  : 𝔄 ▷ Γ ⊢ t ≈ s → 𝔄 ▷ Γ ⊢ s ≈ u → 𝔄 ▷ Γ ⊢ t ≈ u
  ax  : Axiom 𝔄 Γ t s → 𝔄 ▷ Γ ⊢ t ≈ s
  rn  : (ρ : Γ ⇝ Δ) → 𝔄 ▷ Γ ⊢ t ≈ s → 𝔄 ▷ Δ ⊢ r[ ρ ] t ≈ r[ ρ ] s
  ms  : (ζ ξ : [ 𝔄 ⟜ 𝕋 𝔅 ] Δ) → (∀{τ Π}(a : 𝔄 τ Π) → 𝔅 ▷ Π + Γ ⊢ ζ a ≈ ξ a) →
        𝔄 ▷ Γ ⊢ t ≈ s → 𝔅 ▷ Δ + Γ ⊢ ms[ ζ ] t ≈ ms[ ξ ] s
```

The ms constructor expresses that whenever two terms t and s are equivalent, and two instantiations for their metavariable context ζ and ξ are equivalent (for every metavariable), then performing the metasubstitution also gives equivalent terms. Using the equivalence constructors one can derive useful proof combinators and a library for equational reasoning; for example, ax≈ equates two terms via the instantiation of an axiom:

```
ax≈ : Axiom 𝔄 Γ t s → (ζ : [ 𝔄 ⟜ 𝕋 𝔅 ] Γ) → 𝔅 ▷ Γ ⊢ ms t ζ ≈ ms s ζ
ax≈ a ζ = ms (ax a) ζ ζ (λ _ → eq refl)
```

The biggest gains, however, come from not having to tediously encode the congruence rules for every subterm of every term of the syntax. To rewrite a deeply nested subexpression, we simply mark its location in the term with a "typed hole" implemented as a distinguished metavariable ○, and use ms to instantiate it with the two sides of an equality rule: for example, $f \approx g$ and $(\bigcirc\, a) \approx (\bigcirc\, a)$ imply that

$$f\,a = \mathtt{ms}\ (\bigcirc\, a)\ (\lambda\{\bigcirc \to f\}) \overset{\mathtt{ms}}{\approx} \mathtt{ms}\ (\bigcirc\, a)\ (\lambda\{\bigcirc \to g\}) = g\,a$$

Further examples of this and other combinators are given in the next section.

The resulting equational reasoning system is very useful in its own right, as instantiating it with the minimum necessary requirements – the second-order axioms schemes, generalising an infinite number of equations between terms – generates all the required infrastructure to write detailed, explicit equational proofs. We also know that this equational system is sound: given any denotational model of the axioms, every derivable equality in the syntax interprets to equal elements of the model. With the full formalisation of Section 11.3.1 – including what amounts to a proof of the enriched monad structure for $\mathbb{T}$ – the soundness proof can too be translated into Agda with very little change. We define an equational model as a Σ-monoid that equates every term related by Axiom:

```
record EqModel (𝔐^Σ : Σ-Mon 𝔐) : Set₁ where
  field ⊨ax : (κ : [ 𝔄 ⟜ 𝔐 ] Γ)(t s : 𝕋𝔄 α Γ) → Axiom 𝔄 Γ t s → me[ ω ] t ≡ me[ ω ] s
```

Then, given an equational model $\mathfrak{M}$, the proof of the soundness theorem proceeds by induction on the equivalence judgment, just as in Theorem 11.3.2:

```
sound : (κ : [ 𝔄 ⟜ 𝔐 ] Γ)(t s : 𝕋𝔄 α Γ) → 𝔄 ▷ Γ ⊢ t ≈ s → me[ κ ] t ≡ me[ κ ] s
sound κ t s (ax x)      = ⊨ax κ t s x
sound κ t s (eq refl)   = refl
sound κ t s (sy t≈s)    = sym (sound κ s t t≈s)
```

```
sound κ t u (tr t≈s s≈u) = trans (sound κ t s t≈s) (sound κ s u s≈u)
sound κ .(𝕣[ ρ ] t) .(𝕣[ ρ ] s) (rn ρ t≈s) = begin
  𝕞𝕖[ κ ] (𝕣[ ρ ] t)          ≡⟨ 𝕞𝕖□⇒.⟨r⟩ ⟩
  ℳ.r[ ρ + Θ ] (𝕞𝕖[ κ ] t)    ≡⟨ ℳ.r≈ (sound κ t s t≈s) ⟩
  ℳ.r[ ρ + Θ ] (𝕞𝕖[ κ ] s)    ≡⟨ 𝕞𝕖□⇒.⟨r⟩ ⟩
  𝕞𝕖[ κ ] (𝕣[ ρ ] s)          ■
sound κ .(𝕞𝕤[ ζ ] t) .(𝕞𝕤[ ξ ] s) (ms ζ ξ ζ≈ξ t≈s) = begin
  𝕞𝕖[ κ ] (𝕞𝕤[ ζ ] t)          ≡⟨ 𝕞𝕖-assoc ⟩
  𝕞𝕤[ 𝕞𝕖[ κ ] ∘ ζ ] t          ≡⟨ sound (𝕞𝕖[ κ ] ∘ ζ) t s t≈s ⟩
  𝕞𝕤[ 𝕞𝕖[ κ ] ∘ ζ ] s          ≡⟨ 𝕞𝕤≈ (λ a → sound κ (ζ a) (ξ a) (ζ≈ξ a)) ⟩
  𝕞𝕤[ 𝕞𝕖[ κ ] ∘ ξ ] s          ≡⟨ 𝕞𝕖-assoc ⟩
  𝕞𝕖[ κ ] (𝕞𝕤[ ξ ] s)          ■
```

Soundness of the renaming rule follows from 𝕞𝕖 being a □-coalgebra homomorphism, while metasubstitution soundness uses the 𝕞𝕖-assoc corollary from monad multiplication preservation (??). However, the practical value of the soundness theorem is limited: although the generic proof applies to all syntaxes, equational theories, and models, instantiating it for a concrete syntax and axiom set often results in near-intractable typechecking times. If a formalisation relies on the computational content of the soundness theorem, this generic equational framework is unlikely to be applicable. Still, it serves as a valuable formalised proof of a foundational result in second-order equational logic.

In the next section we cover more implementation-specific details of the formalisation, namely the inductive construction of the datatype of terms from a second-order signature.

12.3 Generic signatures

The abstract development so far has been entirely generic over second-order signatures and term syntax. In this section, we explain how to construct endofunctors Σ from syntactic descriptions, compare different term representations and their trade-offs, and show how code generation turns our library into a practical tool for language formalisation. Thanks to our modular design, we retain flexibility in each of the following areas:

- how to encode the signature of a second-order syntax (e.g. binding algebras (Fiore et al., 1999), indexed containers (Altenkirch et al., 2015), Allais et al. (2021)-style Descriptions);
- how to convert the signature into a $\mathbf{Fam}_S$ endofunctor Σ (e.g. polynomial functors (Fiore, 2012; Arkor and Fiore, 2020), higher- or first-order argument collections, Desc interpretations);
- how to define the data type for the initial $\Sigma_{\mathfrak{A}}$-algebra (implicit or explicit encodings).

Each option comes with its own trade-offs, but we identify three choices that balance convenience, flexibility, and good computational behaviour particularly well. Section 12.3.1 defines second-order signatures and their corresponding endofunctors; Section 12.3.2 covers the construction of their initial algebras; and Section 12.3.3 showcases our framework's code-generation capabilities.

12.3.1 Signature endofunctor

Binding signatures introduced by Aczel (1978) generalise standard algebraic signatures to languages with variable binding. Their definition given by Fiore and Hur (2010) and Definition 11.1.1 is straightforward to translate to an Agda record with helper functions:

```
record Signature (O : Set) : Set₁ where
  constructor sig
  field ⌊_⌋ : O → List (Ctx × S) × S
```

```
Arity : O → List (Ctx × S)
Arity o = proj₁ ⌊ o ⌋

Sort : O → S
Sort o = proj₂ ⌊ o ⌋
```

The set S of sorts is normally given as an inductive data type, and O as an enumeration of operators. For example, the STLC has the following sorts and operator symbols:

```
data ΛT : Set where
  N   : ΛT
  _→_ : ΛT → ΛT → ΛT
```

```
data Λₒ : Set where
  appₒ lamₒ : {α β : ΛT} → Λₒ
```

The Signature instances are translations of the signature of the λ-calculus from Example 11.1.2. Shorthands for specifying argument lists and bound variables make the declaration concise.

```
Λ:Sig : Signature Λₒ
Λ:Sig = sig λ where (appₒ {α}{β}) → (⊢₀ α → β) , (⊢₀ α)  ⟼₂  β
                    (lamₒ {α}{β}) →          (α ⊢₁ β)  ⟼₁  α → β
```

The signature contains all the information needed to determine the syntactic structure of a language. To make use of the abstract development in Section 12.2, we convert a Signature into a sorted-family endofunctor Σ, which associates constructors with argument terms. For example, given the signature of the STLC, an element of $\Sigma\,\mathcal{X}\,\beta\,\Gamma$ may be the operator app associated with two $\mathcal{X}$-terms $f : \mathcal{X}\,(\alpha \to \beta)\,\Gamma$ and $a : \mathcal{X}\,\alpha\,\Gamma$, while an element of $\Sigma\,\mathcal{X}\,(\alpha \to \beta)\,\Gamma$ may be the operator lam with a term $b : \mathcal{X}\,\beta\,(\alpha \cdot \Gamma)$.

For technical reasons explained later, we choose to represent the "collection" of arguments of an operator as a tuple of terms. An alternative would be a higher-order encoding as a mapping from an argument index to a term (similar to the implementation of substitution rules); however, even though constructing the Strong instance for such a representation would be easier, it complicates the initiality proof which we wish to keep as simple as possible.

```
Arg : List (Ctx × T) → Famₛ → Fam
Arg []              𝒳 Γ = ⊤
Arg ((Θ , τ) :: as) 𝒳 Γ = δ Θ 𝒳 τ Γ × Arg as 𝒳 Γ
```

Note the use of the context extension endofunctor δ, used to add the new variables bound by an argument to the global context, making all variables available in the body of the binder. The definition of the signature endofunctor for a signature $(T, O, \mathrm{Arity}, \mathrm{Sort})$ is then as follows:

```
Σ : Famₛ → Famₛ
Σ 𝒳 α Γ = Σ[ o ∈ O ] (α ≡ Sort o × Arg (Arity o) 𝒳 Γ)
```

An element of the set $\Sigma\ \mathcal{X}\ \alpha\ \Gamma$ is a dependent tuple consisting of an operator symbol $o \in O$, a proof that the output sort of the operator is α, and a tuple of $\mathcal{X}$-terms for each operand of the operator of the type and extension context given by the operator arity. For example, an element of $\Sigma\,\mathcal{X}\,\beta\,\Gamma$ is $(\mathsf{app}, \mathsf{refl}, (f, t, \mathsf{tt}))$, for terms $f : \mathcal{X}\,(\alpha \to \beta)\,\Gamma$ and $t : \mathcal{X}\,\alpha\,\Gamma$. One can suppress the tt for operators of positive arity by adding a case for a singleton argument list in the definition of Arg, and use a pattern synonym (Pickering et al., 2016) to hide the refl element, writing $\mathsf{app} \wr (f, t)$ for the above.

The Strong instance for Σ may be derived from a similar lawful strength transformation $\mathsf{Arg}\ as\ (\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!) \to\!\!\!\cdot\ (\!|\,\mathcal{X}\,,\,\mathsf{Arg}\ as\ \mathcal{Y}\,|\!)$ for Arg, which applies the δ strength to every component of type $\delta\ \Theta\ (\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!)\ \tau\ \Gamma$ in the argument list:

$$\begin{aligned}
&\mathsf{str}^A : (\mathcal{X}^{\square}_{*} : \mathsf{PCoalg}\ \mathcal{X})(\mathcal{Y} : \mathsf{Fam}_{\mathbb{S}})(as : \mathsf{List}\ (\mathsf{Ctx} \times T)) \to \mathsf{Arg}\ as\ (\!|\,\mathcal{X}\,,\,\mathcal{Y}\,|\!) \to\!\!\!\cdot\ (\!|\,\mathcal{X}\,,\,\mathsf{Arg}\ as\ \mathcal{Y}\,|\!)\\
&\mathsf{str}^A\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ []\qquad\quad \mathsf{tt}\qquad \sigma = \mathsf{tt}\\
&\mathsf{str}^A\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ ((\Theta\ ,\ \tau) :: as)\ (h\ ,\ hs)\ \sigma = (\delta{:}\mathsf{Str}.\mathsf{str}\ \Theta\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ h\ \sigma)\ ,\ (\mathsf{str}^A\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ as\ hs\ \sigma)
\end{aligned}$$

The strength laws are similarly established by pointwise application of the appropriate δ:Str fields to the elements of the argument tuple. Extending Arg-strength to Σ is easy, since the operation does not modify the operator or sort equality proof. In addition to Σ:Str below, we also have an instance of convolutional strength for Σ derived via weakening.

$$\begin{aligned}
&\Sigma{:}\mathsf{Str} : \mathsf{Strong}\ \Sigma\mathsf{F}\\
&\Sigma{:}\mathsf{Str} = \mathsf{record}\ \{\ \mathsf{str} \quad\ = \lambda\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ (o\ ,\ e\ ,\ a)\ \sigma \to (o\ ,\ e\ ,\ \mathsf{str}^A\ \mathcal{X}^{\square}_{*}\ \mathcal{Y}\ (\mathsf{Arity}\ o)\ a\ \sigma)\\
&\qquad\qquad ;\ \mathsf{str\text{-}nat}_1 = \lambda\ f^{\square}_{*}\Rightarrow\ (o\ ,\ e\ ,\ a)\ \sigma \to \mathsf{cong}\ (o\ ,\ e\ ,_)\ (\mathsf{str}^A\text{-}\mathsf{nat}_1\ f^{\square}_{*}\Rightarrow (\mathsf{Arity}\ o)\ a\ \sigma)\ ;\ \ldots\ \}
\end{aligned}$$

12.3.2 Term syntax

The final piece of the puzzle is constructing the initial $\Sigma_{\mathfrak{Y}}$-algebra $\mathbb{T}$ from a second-order signature endofuctor. Such initial algebras correspond to inductive data types whose constructors combine $\mathbb{T}$-terms into other $\mathbb{T}$-terms, allowing for arbitrarily nested syntactic structure.

One way to encode syntax is to treat the tuples $(\mathsf{op} \wr (a_1, \ldots, a_n))$ as the terms, directly encoding the Σ-algebra structure as a unified term constructor $\mathsf{con} : \Sigma\,\mathbb{T} \to\!\!\!\cdot\ \mathbb{T}$; another is the explicit encoding with one constructor per operator.

Implicit encoding Alongside the Σ-algebraic structure, terms of a second-order syntax also have to include constructors for variables and metavariables. This suggests the following generic data type of terms for an arbitrary signature:

$$\begin{aligned}
&\mathsf{data}\ \mathsf{Tm} : \mathsf{Fam}_{\mathbb{S}}\ \mathsf{where}\\
&\quad \mathsf{con}\ \ : \Sigma\ \mathsf{Tm}\ \tau\ \Gamma \qquad\qquad\quad \to \mathsf{Tm}\ \tau\ \Gamma\\
&\quad \mathsf{var}\ \ : \mathcal{I}\ \tau\ \Gamma \qquad\qquad\quad \to \mathsf{Tm}\ \tau\ \Gamma\\
&\quad \mathsf{mvar} : \mathfrak{A}\ \tau\ \Pi \to \mathsf{Sub}\ \mathsf{Tm}\ \Pi\ \Gamma \to \mathsf{Tm}\ \tau\ \Gamma
\end{aligned}$$

In place of higher-order substitution rules $\Pi\ {-}[\ \mathsf{Tm}\ ]{\to}\ \Gamma$, we use the inductive family Sub to implement the metavariable environment:

```
data Sub (𝒳 : Fam_S) : Ctx → Ctx → Set where
  •   : Sub 𝒳 ∅ Γ
  _►_ : 𝒳 α Γ → Sub 𝒳 Π Γ → Sub 𝒳 (α · Π) Γ
```

Sub a first-order, inductive encoding of simultaneous substitution rules, assigning a term in context Γ to every variable in context Π. Though isomorphic to the higher-order encoding (with conversion functions lookup and tabulate), this representation is better suited for syntax as it evaluates structural recursion fully. Since interpretations 𝕚 recurse into the metavariable environment, a higher-order encoding would suspend recursion under binders: the application 𝕚 (mvar 𝔞 (λ new → con (op ≀ t))) only reduces to mvar 𝔞 (λ new → 𝕚 (con (op ≀ t))), and 𝕚 doesn't reach inside the constructor unless the function is applied to new. In contrast, our first-order encoding Sub fully normalises each substitution component, so 𝕚 (mvar 𝔞 (con (op ≀ t) ► •)) directly reduces to mvar 𝔞 (con (op ≀ 𝕚 t) ► •). This same consideration motivates our choice to represent operator arguments as tuples, not higher-order assignments: though the strength instance would be simpler with the latter, tuple-based representations simplify recursion and equational reasoning, since syntactic operations only descend one level into subterms.

Proving initiality for Tm requires defining a recursive function 𝕚 : Tm –▹ 𝒜 for any syntactic algebra 𝒜, interpreting constructors and variable structures. The naïve definition runs into trouble: applying 𝕚 over a tuple of subterms prevents Agda from seeing the call as structurally recursive. Allais et al. (2021, Section 4) faced the same issue, resolving it with sized types (Abel, 2010), which mark terms as larger than their subterms. While this enables defining 𝕚, it introduces a pervasive size index and doesn't extend to proving uniqueness – a crucial property in our framework (see Pitts (2019) for details in the context of general F-algebras).

Our solution is simple and effective: instead of interpreting subterms via functorial lifting (𝒳 –▹ 𝒴) → (Arg *as* 𝒳 Γ → Arg *as* 𝒴 Γ), we define mutually recursive functions 𝔸 and 𝕊 that apply 𝕚 directly to each subterm. This inlining satisfies Agda's termination checker without needing sized types.

```
𝔸 : ∀ as → Arg as Tm Γ → Arg as 𝒜 Γ
𝔸 []       tt       = tt
𝔸 (a :: as) (t , ts) = (𝕚 t , 𝔸 as ts)
```

```
𝕊 : Sub Tm Π Γ → Π –[ 𝒜 ]→ Γ
𝕊 (t ► σ) new    = 𝕚 t
𝕊 (t ► σ) (old v) = 𝕊 σ v
```

```
𝕚 (con (o , e , a)) = a (o , e , 𝔸 (Arity o) a)
𝕚 (var v)           = v v
𝕚 (mvar 𝔞 ε)        = m 𝔞 (𝕊 ε)
```

The proof that 𝕚 is the unique $\Sigma_{\mathfrak{N}}$-algebra homomorphism is also fairly straightforward, requiring only a few mutually inductive lemmas about 𝔸 and 𝕊. This establishes an instance of Initial SynAlgs for the term datatype Tm, enabling us to instantiate the entire generic metatheory – gaining substitution, its correctness properties, sound compositional interpretations in models, and more, essentially for free.

Explicit encoding The implicit encoding achieves our conceptual goal: it provides a first-order initial syntactic algebra for any second-order signature. Its main practical drawback is

that the resulting term syntax is tightly coupled to the algebraic framework, forcing users into a cumbersome representation that deviates from the more natural and elegant "one constructor per typing rule" style of intrinsic typing. Pattern synonyms can help improve surface syntax, but we found them fragile – especially in parametrised modules – and their untyped nature feels ill-suited for something as central as the syntax of a formalised language. Ideally, our framework should integrate seamlessly into existing developments, allowing users to retain their original syntax definitions without needing to overhaul their core term data types.

This is entirely feasible thanks to our clear separation between signatures, endofunctors, and initial syntactic algebras. Defining an initial algebra instance for an existing data type is not particularly burdensome: it requires a recursive interpretation function, a homomorphism proof, and an inductive uniqueness proof. Metavariables still require the mutually recursive transformation $\mathbb{S}$ (defined as before and omitted here), but since subterm recursion is now handled manually, the auxiliary transformation $\mathbb{A}$ is no longer necessary.

```
data Λ : Fam_S where
  var  : ℐ τ Γ → Λ τ Γ
  mvar : 𝔄 τ Π → Sub Λ Π Γ → Λ τ Γ
  app  : Λ (α → β) Γ → Λ α Γ → Λ β Γ
  lam  : Λ β (α · Γ) → Λ (α → β) Γ
```

```
𝕚 : Λ ⇾ 𝒜
𝕚 (var v)     = v v
𝕚 (mvar a ε) = m a (𝕊 ε)
𝕚 (app g a)   = a (app₀ ≀ 𝕚 g , 𝕚 a)
𝕚 (lam b)     = a (lam₀ ≀ 𝕚 b)
```

The homomorphism instance uses a simple lemma $\mathbb{S}$-tab : $\forall \varepsilon \to \mathbb{S}$ (tabulate ε) = $\mathbb{i} \circ \varepsilon$, and the Σ-algebra homomorphism proof, which is satisfied merely by pattern-matching on the operand and sort equality proof.

```
𝕚^Σ⇒ : MetaAlg⇒ Λ^Σ 𝒜^Σ 𝕚
𝕚^Σ⇒ = record { ⟨a⟩ = λ {t = t} → ⟨a⟩ t ; ⟨v⟩ = refl
              ; ⟨m⟩ = λ {a = a}{ε} → cong (m a) (𝕊-tab ε) }
  where ⟨a⟩ : (t : Σ Λ α Γ) → 𝕚 (Λ^Σ.a t) ≡ 𝒜^Σ.a (Σ₁ 𝕚 t)
        ⟨a⟩ (app₀ ≀ _) = refl
        ⟨a⟩ (lam₀ ≀ _) = refl
```

The uniqueness proof – that $\mathbb{i}$ is equal to any syntactic algebra homomorphism $g : \Lambda \rightarrowtriangle \mathcal{A}$ – involves the mutually inductive lemma $\mathbb{S}$-lu and the inverse property of tabulate and lookup in the metavariable case; everything else follows from the homomorphism properties of g.

```
𝕊-lu : (ε : Sub Λ Π Γ)(v : ℐ α Π) → 𝕊 ε v ≡ g (lookup ε v)
𝕊-lu (t ► ε) new     = 𝕚! t
𝕊-lu (t ► ε) (old v) = 𝕊-lu ε v

𝕚! : (t : Λ α Γ) → 𝕚 t ≡ g t
𝕚! (var v) = sym ⟨v⟩
𝕚! (mvar a ε) rewrite 𝕊-lu ε = trans (sym ⟨m⟩) (cong (g ∘ mvar a) (tab-lu-id ε))
𝕚! (app f a)  rewrite 𝕚! f | 𝕚! a = sym ⟨a⟩
𝕚! (lam b)    rewrite 𝕚! b       = sym ⟨a⟩
```

With minimal additional boilerplate, we can prove that the inductively defined family Λ is an initial syntactic algebra and apply this result to instantiate our metatheory. The Theory module associated with the initial syntactic algebra provides direct access to all definitions and lemmas in the framework: for example, the coveted single-variable substitution operation is immediately available as Theory.[_/] : $\Lambda\ \alpha\ \Gamma \to \Lambda\ \beta\ (\alpha \cdot \Gamma) \to \Lambda\ \beta\ \Gamma$, with no extra effort required. We can reduce the boilerplate even further with code generation.

12.3.3 Code generation

The next natural step is to eliminate boilerplate entirely by allowing users to move directly from a second-order signature specification to a formalised metatheory of an explicitly encoded term datatype. Since the initiality proof for the explicit encoding is highly schematic and largely signature-independent, we generate the corresponding Agda code from a syntax description via a simple Python script. This signature-to-Agda "compiler" takes a concise textual specification of type and term signatures and produces the Signature declaration and initiality proof code, as described in Sections 12.3.1 and 12.3.2. For example, the signature of the STLC can be specified as follows, with optional infix symbols and fixity annotations:

```
type                        term
  N    : 0-ary                  app : (α → β)   α   → β          | _$_ l20
  _→_ : 2-ary | r30             lam : α.β           → (α → β)   | λ_   r10
```

The input file consists of a type signature and a term syntax block. Type operators are annotated with their arity, and term operators are given with their type signatures. Binding is expressed by prefixing the type of bound variables to the type of the body – for example, the lam constructor takes an expression of type β that binds an additional variable of type α. For convenience, term operators may be annotated with mixfix symbolic representations and fixity (in Agda style): for instance, application will be compiled to the _$_ constructor, associating to the left with precedence 20.

Saving this file as `stlc` and running `python soas.py stlc` generates `Signature.agda` and `Syntax.agda`, containing all necessary imports, the signature declaration Λ:Sig, the inductive term family Λ, and the initiality proof. This enables the user to jump straight to the substantive aspects of language formalisation – like operational or denotational semantics – without having to worry about low-level syntax machinery. The generated encoding is idiomatic and intrinsically typed, leaving users free to build on it however they choose.

The compiler is a simple Python program that parses the signature description and outputs formatted Agda using string templates. Its simplicity is an advantage: the generated code is minimal and systematic (about 1–2 lines per type constructor and 6–7 per term constructor), and has been manually tested across a range of examples for robustness. Unlike other code-generation approaches that output the full syntax-specific formalisation – including fragile and impenetrable de Bruijn proofs – we leverage our generic metatheory and generate just enough code to instantiate it: namely, the initiality proof, which is both elegant and reusable. As a result, the output is concise, readable, and easy to maintain.

The compiler also supports useful syntactic sugar in the specification language, including an Agda-style module system for composing signatures. These are desugared into flat Agda

datatypes, avoiding the complexity of defining categorical sums, products, etc., in the underlying formalisation. More examples follow in the next section.

To conclude, we compare our approach to existing proposals for generic encoding of signatures and corresponding generic metatheory.

Signatures For signatures involving variable binding, notable approaches to generic syntax encoding include *indexed containers* (Altenkirch et al., 2015), initial algebraic encodings of *induction-recursion* (Dybjer and Setzer, 1999; Benke et al., 2003), and the *universe of descriptions* framework by Chapman et al. (2010). The latter underpins the generic syntax descriptions of Allais et al. (2021), which allow one to specify constructors by listing their arities and binding behaviour, and interpret them as endofunctors on families. The encoding is similar to ours in that it combines a set of symbols with an arity function, mapping every symbol to its intended type signature:

```
data 'STLC : Set where
  App Lam : Type → Type → 'STLC
```

```
STLC : Desc Type
STLC = 'σ 'STLC $ λ where
  (App α β) → 'X [] (α → β) ('X [] α ('■ β))
  (Lam α β) → 'X α :: [] β ('■ (α → β))
```

The Desc datatype is interpreted as an endofunctor on families in a way similar to our construction of the Σ endofunctor: operators are dependently combined with a list of interpretations of constructor arguments:

```
⟦_⟧: Desc S → S-Scoped → S-Scoped
⟦ 'σ A d   ⟧ 𝒳 α Γ = Σ[ o ∈ A ] (⟦ d a ⟧ 𝒳 α Γ)
⟦ 'X Δ β d ⟧ 𝒳 α Γ = 𝒳 β (Δ + Γ) × ⟦ d ⟧ 𝒳 α Γ
⟦ '■ β     ⟧ 𝒳 α Γ = α ≡ β
```

The interpretations correspond to the operator index, Arg list, and return sort equality proof, respectively. The benefit of our modular approach is that as long as this generated endofunctor is pointed and convolutional strong (which it is), our Signature encoding of signatures can be replaced with the Desc encoding, without altering any of the metatheory.

Term encoding Allais et al. (2021) use the Desc interpretation function in their implicit term encoding Tm, representing terms with the constructor 'con : S⟦ d ⟧ (Tm s) –▹ Tm (↑ s), where s is the size index needed to ensure that the interpretation map terminates. This ties the term representation closely to the Desc datatype, which makes much of the generic metatheory coupled to Desc, rather than an arbitrary endofunctor. In contrast, we have clear abstraction barriers between the signature encoding, term encoding, and generic metatheory, exemplified by the implicit and explicit encoding of terms that nevertheless satisfy the same initiality theorem.

Initiality Although Allais et al. (2021) circle around the idea, they never actually prove the initiality of the syntax datatype. Their central proof – the *Fundamental Lemma of Semantics* – is in fact the construction of a *parametrised traversal*. The Semantics module for a given description d : Desc axiomatises the structure needed on families $\mathcal{X}, \mathcal{A}$ to induce a function

semantics : Γ −[𝒳]→ Δ → (Tm s α Γ → 𝒜 α Γ), which, with a bit of rearranging, is the type of an initial traversal map Tm s →̣ ⟪ 𝒳 , 𝒜 ⟫. The requirements are that 𝒳 is Thinnable (i.e. a □-coalgebra), there is an embedding 𝒳 →̣ 𝒜 (analogous to our map v for a $\Sigma^{\mathcal{X}}_{\mathfrak{A}}$-algebra 𝒜), and a map S⟦ d ⟧ →̣ 𝒜 for the interpretation function S⟦ _ ⟧ given below:

```
S⟦ _ ⟧: Desc S → S-Scoped → S-Scoped
S⟦ 'σ A d    ⟧ 𝒳 α Γ = Σ[ o ∈ A ] (⟦ d a ⟧ 𝒳 α Γ)
S⟦ 'X [] β d ⟧ 𝒳 α Γ = 𝒜 β Γ × S⟦ d ⟧ 𝒳 α Γ
S⟦ 'X Δ β d ⟧ 𝒳 α Γ = □((Sub 𝒳 Δ) →̣ 𝒜 β) × S⟦ d ⟧ 𝒳 α Γ
S⟦ '■ β      ⟧ 𝒳 α Γ = α ≡ β
```

The operation is admittedly difficult to parse, even with some of the constructs inlined. At its core, it translates terms with bound variables into terms parametrised by a renaming function and an instantiation of those variables as 𝒳-terms in the renamed context. While the analogy with Kripke function spaces is suggested, it remains underdeveloped, and the definition of semantics – mutually recursive with body – is less intuitive than simply requiring syntactic structure on 𝒜. The somewhat ad hoc constructions – Kripke, Scope, body, and others – also affect downstream results, such as the generic simulation framework. Although Allais et al. (2021, Section 10.3) express scepticism toward the abstract categorical approach of Fiore et al. (1999), their library ends up sidestepping more canonical and well-understood categorical tools in favour of bespoke abstractions that may feel unfamiliar or opaque, regardless of whether the reader approaches from formalisation or category theory.

In this section, we outlined the Agda formalisation of our categorical framework, highlighting features that retroactively justify our key design choices: the use of sorted families (which are both natural to represent and easy to manipulate), closed structure and linearity (ensuring dependent function spaces are readily available), and linear maps (which translate quotienting conditions into function axioms). Throughout, we upheld a modular development style, limiting interactions between components to minimal, well-defined interfaces such as strong functors and initial algebras. We also drew comparisons with the state-of-the-art in type- and scope-safe formalisation, arguing that the `agda-soas` library successfully addresses limitations in prior work. Next, we showcase a variety of examples of our framework in use.

CHAPTER 13

Examples

Our framework is flexible and unopinionated: it can be integrated into any intrinsically-typed formalisation of a second-order calculus to equip the syntax with essential operations like weakening, substitution, and their associated laws. It also aids in defining evaluation functions, interpreters, and syntactic translations in a concise and provably correct way, while supporting the construction of signature-generic syntactic operations. In this chapter, we illustrate several ways the library can be applied: Section 13.1 explores its metatheoretic and equational capabilities through the analysis of various syntaxes, while Section 13.2 focuses on generic, structurally recursive, and context-aware operations.

13.1 Formal systems and second-order calculi

We present some of the main features of our Agda formalisation framework and syntax description language using the examples of the simply-typed λ-calculus, algebraic structures, and partial differentiation.

13.1.1 Semantics of the simply-typed λ-calculus

The simply-typed lambda calculus (STLC) has served as our running example, and our framework offers an ideal playground for experimenting with its various extensions – whether by adding new types, term constructs, or equations. Below is a list of features that can be easily specified and compiled into Agda. While the semantics of such an expressive language would be complicated, its syntax is still captured as a second-order signature.

```
type
  N    : 0-ary    -- Natural numbers
  _→_  : 2-ary    -- Functions
  _⊗_  : 2-ary    -- Products
  _⊕_  : 2-ary    -- Sums
  ¬_   : 1-ary    -- Continuations
  T    : 1-ary    -- Monadic computations
```

```
term
  app    : (α → β)   α     → β
  lam    : α.β             → (α → β)

  let    : α      α.β      → β
  fix    : (α → α)         → α

  throw  : α      ¬ α      → β
  callcc : (¬ α).α         → α

  return : α               → T α
  bind   : T α    α.(T β)  → T β

  pair : α   β             → α ⊗ β
  fst  : α ⊗ β             → α
  snd  : α ⊗ β             → β

  inl  : α                 → α ⊕ β
  inr  : β                 → α ⊕ β
  case : α ⊕ β  α.γ  β.γ   → γ

  ze   :                   → N
  su   : N                 → N
  nrec : N  α  (α,N).α     → α
```

We will use the minimal fragment of STLC (with app and lam) to showcase the construction of models and interpretations. The CCC model of the STLC (Lambek, 1986) in the category **Set** interprets types as sets, and terms $\Gamma \vdash t : \alpha$ as functions from the interpretation of Γ to the interpretation of α. Interpretation of contexts can be higher-order or first-order (as a cartesian product of type interpretations) – with the higher-order encoding the model definition is remarkably concise.

```
⟦_⟧ : ΛT → Set
⟦ N ⟧       = ℕ
⟦ α → β ⟧ = ⟦ α ⟧ → ⟦ β ⟧

⟦_⟧ᶜ : Ctx → Set
⟦ Γ ⟧ᶜ = ∀{α} → ℐ α Γ → ⟦ α ⟧

_⁺_ : ⟦ α ⟧ → ⟦ Γ ⟧ᶜ → ⟦ α · Γ ⟧ᶜ
(a ⁺ γ) new    = a
(a ⁺ γ) (old v) = γ v

Env : Family_S
Env α Γ = ⟦ Γ ⟧ᶜ → ⟦ α ⟧
```

```
Env^ΣM : ΣMon Env
Env^ΣM = record
  { ᴹ = record { η = λ v γ → γ v ; μ = λ t σ γ → t (λ v → σ v γ)
               ; lunit = refl ; runit = refl ; assoc = refl }
  ; alg     = λ { (app₀ ⁝ f , a) γ → f γ (a γ)
                ; (lam₀ ⁝ b)     γ → λ a → b (a ⁺ γ) }
  ; μ⟨alg⟩ = λ { (app₀ ⁝ _) → refl
                ; (lam₀ ⁝ b) → ext² λ δ a → cong b (dext
                    λ { new → refl ; (old v) → refl }) } }

module Env = FreeMonoid ∅ Env^ΣM (λ ())

eval : Λ₀ ↛ Env
eval = Env.𝕚
```

Here we restrict to the sorted family $\Lambda_0 = \Lambda\ \emptyset$ of λ-terms without metavariables for simplicity. The eval function interprets λ-terms as Agda programs; for example, eval (lam (lam x_1)) (λ ()) (where the 1st de Bruijn index var (old new) is abbreviated x_1) normalises to the Agda function $\lambda\, x\, y \to x$. Since it is derived by initiality, the interpretation is compositional and satisfies the semantic substitution lemma *by construction*. This is an enormous time-saver, since proving the soundness of substitution is often one of the most tedious steps required for the development of denotational semantics – the binding terms are the usual suspects, forcing one to reason about semantics of lifting, weakening, renaming, etc. Our framework does all the heavy lifting, allowing users to move on to less bureaucratic proofs. For example, after defining the predicate Val satisfied by value terms of the form $\lambda\ b$ for $b : \Lambda_0\ \beta\ (\alpha \cdot \Gamma)$, it takes minimal effort to equip the language with an intrinsically-typed call-by-value reduction relation and a proof that it preserves the interpretation of terms:

```
data _⇝_ : Λ₀ α Γ → Λ₀ α Γ → Set where
  ζ-$₁ : {f g : Λ₀ (α → β) Γ} {a   : Λ₀ α Γ}                → f ⇝ g → f $ a   ⇝ g $ a
  ζ-$₂ : {f   : Λ₀ (α → β) Γ} {a b : Λ₀ α Γ}      → Val f → a ⇝ b → f $ a   ⇝ f $ b
  β-λ  : {t   : Λ₀ α Γ}       {b   : Λ₀ β (α · Γ)} → Val t          → (λ b) $ t ⇝ [ t /] b
```

```
sound : {t s : Λ₀ α Γ} → t ⇝ s → (γ : ⟦ Γ ⟧ᶜ) → eval t γ ≡ eval s γ
sound (ζ-$₁ r)       γ rewrite sound r γ = refl
sound (ζ-$₂ _ r)     γ rewrite sound r γ = refl
sound (β-λ {t}{b} _) γ rewrite Env.sub-lemma t b
  = cong (eval b) (ext λ { new → refl ; (old v) → refl })
```

Note the use of the freely-obtained single-variable substitution `[ t /] b` in the *β-λ* axiom, and the invocation of sub-lemma which translates `eval ([ t /] b) γ` to `Env.[ eval t /] (eval s) γ`. The only technical "intervention" needed is forcing the evaluation of environment entries using function extensionality, as otherwise Agda wouldn't reduce the equalities in the higher-order *γ* argument (cf. discussion of inductive and higher-order substitution rules in Section 12.3.2).

Despite its simplicity, implementing the STLC's syntax, type system, operational and denotational semantics from scratch still involves a considerable amount of effort, mostly spent defining and reasoning about substitution. Leveraging our framework, this standard but relatively robust formalisation is possible in fewer than 100 lines of Agda code. Further proofs like progress, determinacy, normalisation, etc. will rarely be held up by having to establish a metasyntactic property – it is likely to be present in the Theory module already.

13.1.2 Modular theories

While experimenting with the syntax description language, the benefit of modularity and reuse quickly became apparent: algebraic structures and calculi involve some hierarchical inheritance that can be elegantly exposed using a simple, Agda-like module system. Combined with the equational reasoning framework, the system becomes an effective tool for formalising algebraic structures, logics, and second-order extensions thereof.
For example, consider the following signature of a monoid, where the lack of a type declaration marks the signature as one-sorted with sort `*`:

```
syntax Monoid | M

term
  unit :            * | ε
  mult : * * →      * | _⊕_ l20
```

Feeding this into the generator produces an axiomatisation of monoids and the Agda data type `M` with four constructors: variables, metavariables, the unit *ε* and monoid operation ⊕. To make use of the equational reasoning framework of Section 12.2.2, we extend the syntax description with the equational axioms of monoids, which will compile to an Axiom instance in Agda. The system supports formulaic descriptions of several common algebraic properties, making the axiomatisation very familiar:

```
theory
  'unit' unit of 'add'
  'add' associative
```

The generated `Equality.agda` file contains the following relation, expressing the properties as constructors of a (metavariable) context-indexed relation between terms:

```
data _⊳_⊢_≈A_ : (𝔄 : MCtx)(Γ : Ctx){α : ∗T} → M 𝔄 α Γ → M 𝔄 α Γ → Set where
  εU⊕L : ⌈ ∗ ⌉              ⊳ ∅ ⊢ ε ⊕ 𝔞      ≈A 𝔞
  εU⊕R : ⌈ ∗ ⌉              ⊳ ∅ ⊢ 𝔞 ⊕ ε      ≈A 𝔞
  ⊕A   : ⌈ ∗ ⌉ ⌈ ∗ ⌉ ⌈ ∗ ⌉ ⊳ ∅ ⊢ (𝔞 ⊕ 𝔟) ⊕ 𝔠 ≈A 𝔞 ⊕ (𝔟 ⊕ 𝔠)
```

Every axiom quantifies over some number of metavariables listed in the MCtx declaration before the ⊳ that may be mentioned in the equation (as "alphabetic" de Bruijn indices 𝔞, 𝔟, 𝔠, etc.) and instantiated independently; in this case, the metavariables have no type or parameters, but we will see more interesting examples later. Each $\mathfrak{A} : \mathrm{MCtx}$ can be turned into the corresponding sorted family of metavariables $\underline{\mathfrak{A}} : \mathrm{Fam}_s$, with the number of constructors determined by the number of $\lceil\, \Pi \Vdash \tau \,\rceil$ blocks (or $\lceil\, \tau \,\rceil$, if Π is empty), and the parameter context and sort of the metavariables given by Π and τ respectively.

Instantiating the generic equational logic with this set of axioms yields a formally verified equational proof system for monoids. While the resulting derivations can appear pedantic, this level of rigour is not necessarily unwelcome: carefully stepping through algebraic proofs – explicitly invoking properties such as associativity, identity laws, and congruence – can be not only satisfying but also deeply educational. For instance, consider the following reduction of a monoid expression $(x \oplus \varepsilon) \oplus (y \oplus x) = (x \oplus y) \oplus x$:

```
calc : ■ ⊳ ⌊ ∗ · ∗ ⌋ ⊢ (x₀ ⊕ ε) ⊕ (x₁ ⊕ x₀) ≈ (x₀ ⊕ x₁) ⊕ x₀
calc = begin  (x₀ ⊕ ε) ⊕ (x₁ ⊕ x₀)   ≈⟨ cong⌊ ax εU⊕R with⟪ x₀ ⟫ ⌋inside ○ ⊕ (x₁ ⊕ x₀) ⟩
              x₀ ⊕ (x₁ ⊕ x₀)          ≈⟨ ax ⊕A with⟪ x₀ ◃ x₁ ◃ x₀ ⟫ ⟩s
              (x₀ ⊕ x₁) ⊕ x₀          ■
```

The reasoning steps are instances of the ax and ms constructors of the _⊳_⊢_≈_ equality type. Common metasubstitution patterns are extracted into helper operators, making the proofs significantly more readable: for example, the ax *a* with ⟪ $t_1 \triangleleft \ldots \triangleleft t_n$ ⟫ notation associates an axiom *a* with an instantiation of metavariables in the axiom's metavariable context, given as a list of terms – in this case, the right unit is applied to the object variable x_0, and associativity to x_0, x_1 and x_0. Applications of an equation in a subexpression of *t* is done with the congruence combinator cong⌊ *e* ⌋inside *t*[○], where the hole ○ denotes the place in the expression in which *e* is applied. In the calc example, we first apply the right unit axiom to the left subterm of ⊕ by guiding it to the right place with the ○ ⊕ ($x_1 \oplus x_0$) congruence context.

Returning to modularity, a whole zoo of algebraic structures can be specified by extending monoids with new axioms and operators, or combining existing structures. For example, a commutative monoid is a monoid where the binary operator is commutative, while a group is a monoid with an inverse operator:

```
syntax CMonoid | CM extends monoid

theory
  'add' commutative
```

```
syntax Group | G extends monoid

term
  neg : ∗ ⟶ ∗ | ⊖_ r40

theory
  ‘neg’ inverse of ‘add’ with ‘unit’
```

The `extends` keyword imports the signature and equations of the named module. It supports multiple “inheritance” and renaming of operators and changing of fixities, as shown in the following declaration of a semiring: an additive commutative monoid with a multiplicative monoid, connected by distributivity and annihilation axioms.

```
syntax Semiring | SR extends
 - cmonoid (renaming unit:𝜀 to zero:𝟘)
 - monoid  (renaming unit:𝜀 to one:𝟙, add:⊕ to mult:⊗:l30)

theory
  ‘mult’ distributes over ‘add’
  ‘zero’ annihilates ‘mult’
```

A ring can be presented either by combining an additive commutative group with a multiplicative monoid, or by equipping a semigroup with a negation operator – both constructions ultimately yield the same syntax. However, this modularity exists only at the level of syntax descriptions: in the generated Agda code, all such extensions are flattened into a single, unified axiomatisation. While supporting modularity within the formalisation framework itself is possible, it would require managing nested sums of signature endofunctors and explicit injections for operators – choices that introduce considerable syntactic and notational overhead.

As an example of a heterogeneous signature with nonstandard equations, we axiomatise group actions as a group extended with a new sort `A` and left action operator of a group element to an element of `A`. The two axioms are written out as explicit second-order equations of the form `(name) metavars ⊳ expr₁ = expr₂`, quantifying over metavariables of different sorts: `g` and `h` are group elements (of the implicit sort `∗`), while `a` is an element of the set `A`.

```
syntax GroupAction | GA extends group

type
  A : 0-ary

term
  act : ∗ A ⟶ A | _⊕_ r30

theory
  (𝜀A⊙)        a : A  ⊳ act(unit, a)      = a
  (⊕A⊙)  g  h  a : A  ⊳ act(add(g, h), a) = act(g, act(h, a))
```

Writing out the axioms using the textual operators and applicative style eliminates any ambiguity, but the generated Agda axioms use the more natural symbolic operators. In fact, the descriptive algebraic properties we used above desugar into the explicit specifications: the `'neg' inverse of 'add' with 'unit'` directive expands into the axiom

$$\texttt{(}\ominus\mathsf{N}\oplus^{L}\texttt{) a} \rhd \texttt{add(neg(a), a) = unit},$$

with the axiom name splicing the appropriate symbols for the operators. A simple property of group actions is that action of a group element can be cancelled by action of its inverse, and the proof in Agda is as simple and clear as it would be on paper:

```
act-inv : ⌈ ∗ ⌉ ⌈ A ⌉ ▷ ∅ ⊢ (⊖ 𝔞) ⊙ (𝔞 ⊙ 𝔟) ≈ 𝔟
act-inv = begin  (⊖ 𝔞) ⊙ (𝔞 ⊙ 𝔟)     ≈⟨ ax ⊕A⊙ with⟪ ⊖ 𝔞 ◃ 𝔞 ◃ 𝔟 ⟫ ⟩ₛ
                 ((⊖ 𝔞) ⊕ 𝔞) ⊙ 𝔟     ≈⟨ cong⌊ ax ⊖N⊕ᴸ with⟪ 𝔞 ⟫ ⌋inside ○ᶜ ⊙ 𝔟 ⟩
                 ε ⊙ 𝔟               ≈⟨ ax εA⊙ with⟪ 𝔟 ⟫ ⟩ₛ
                 𝔟                   ■
```

13.1.3 Partial differentiation

Another example of a second-order calculus is the axiomatisation of partial differentiation laid out by Plotkin (2020). The syntax consists of the first-order theory of commutative rings with a second-order partial-differentiation operator $\mathrm{PDiff}(x.e[x], d)$, interpreted as the partial derivative of the expression $e[x$ with respect to x, evaluated at d (which has no free occurrences of x). This is usually denoted $\frac{\partial e\langle x\rangle}{\partial x}\big|_{x=d}$. To differentiate $e\langle x\rangle$ without evaluation, one renames x to a dummy variable w, differentiates $e\langle w\rangle$ with respect to w, then evaluates the result at $w = x$ – thus, the notation $\frac{\partial}{\partial x}e\langle x\rangle$ is taken as abbreviating $\frac{\partial e\langle w\rangle}{\partial w}\big|_{w=x}$.

The signature of commutative rings augmented with the partial-differentiation operator can be readily expressed as an unsorted syntax description, extending the syntax `cring`. Derived operations in braces must be manually implemented in Agda and allow for arbitrary output context: here, ∂_0 and ∂_1 implement the partial derivatives with respect to the first and second variable. Note the use of weakening $\mathbb{wk}$ to extend the context of e with an extra dummy variable that is not affected by the differentiation.

```
syntax PDiff | PD extends cring

term
  pdiff : ∗.∗   ∗ ⟶ ∗     | ∂_|_
  {d0   : ∗.∗     ⟶ ∗.∗   | ∂₀_}
  {d1   : (∗,∗).∗ ⟶ (∗,∗).∗ | ∂₁_}
```

```
∂₀_ : PD 𝔄 ∗ (∗ · Γ) ⟶ PD 𝔄 ∗ (∗ · Γ)
∂₀ e = ∂ (𝕨𝕜 e) ∣ x₀
∂₁_ : PD 𝔄 ∗ (∗ · ∗ · Γ) ⟶ PD 𝔄 ∗ (∗ · ∗ · Γ)
∂₁ e = ∂ (𝕨𝕜 e) ∣ x₁
```

The equational theory of partial differentiation may be directly translated from the paper. Plotkin's use of function variables as abstract terms parametrised by expressions matches our notion of a parametrised metavariable, so the laws may be stated as second-order axioms universally quantifying over both object-level variables, and parametrised metavariables, using the extended syntax `(name) metavars ▷ vars ⊢ expr₁ = expr₂` that also includes a typing context given by `vars`.

```
theory
 'zero' unit of 'add'
 'mult' distributes over 'add'

(∂⊕) a : *      ▷ x : *          ⊢ d0 (add (x,a))   = one
(∂⊗) a : *      ▷ x : *          ⊢ d0 (mult(a, x))  = a
(∂C) f : (*,*).*  ▷ x : *  y : *  ⊢ d1 (d0 (f[x,y])) = d0 (d1 (f[x,y]))
```

Plotkin's binary chain rule generalises the usual unary chain rule, and may be inductively extended to a chain rule of arbitrary arity. The corresponding axiom in the theory is as follows:

```
(∂Ch₂) f : (*,*).* g h : *.*  ▷ x : *
       ⊢ d0 (f[g[x], h[x]])
       = add ( mult(pd(z. f[z, h[x]], g[x]), d0(g[x])),
               mult(pd(z. f[g[x], z], h[x]), d0(h[x])))
```

The familiar unary chain rule is then marked as a derived law, to be implemented in Agda manually after code generation:

```
{(∂Ch₁) f g : *.*  ▷ x : *
        ⊢ d0 (f[g[x]]) = mult (pd (z. f[z], g[x]), d0(g[x]))}
```

The equational logic generated from the axiomatisation is the following Agda datatype:

```
data _▷_⊢_≈A_ : (𝔄 : MCtx)(Γ : Ctx){α : ∗T} → PD 𝔄 α Γ → PD 𝔄 α Γ → Set where
  ∂⊕  : ⌈ ∗ ⌉ ▷ ⌊ ∗ ⌋ ⊢ ∂₀ (x₀ ⊕ a)  ≈A 𝟙
  ∂⊗  : ⌈ ∗ ⌉ ▷ ⌊ ∗ ⌋ ⊢ ∂₀ (a ⊗ x₀)  ≈A a
  ∂C  : ⌈ ∗ · ∗ ⊩ ∗ ⌉ ▷ ⌊ ∗ · ∗ ⌋ ⊢ ∂₁ (∂₀ a⟨ x₀ ► x₁ ⟩) ≈A ∂₀ (∂₁ a⟨ x₀ ► x₁ ⟩)
  ∂Ch₂ : ⌈ ∗ · ∗ ⊩ ∗ ⌉ ⌈ ∗ ⊩ ∗ ⌉ ⌈ ∗ ⊩ ∗ ⌉ ▷ ⌊ ∗ ⌋ ⊢
         ∂₀ a⟨ b⟨ x₀ ⟩ ► c⟨ x₀ ⟩ ⟩ ≈A (∂ a⟨ x₀ ► c⟨ x₁ ⟩ ⟩ ∣ b⟨ x₀ ⟩) ⊗ (∂₀ b⟨ x₀ ⟩) ⊕
                                      (∂ a⟨ b⟨ x₁ ⟩ ► x₀ ⟩ ∣ c⟨ x₀ ⟩) ⊗ (∂₀ c⟨ x₀ ⟩)
```

Some of Plotkin's Lemma 2.1 may now be formalised: for example, the simple corollary $\frac{\partial 0}{\partial x} = 0$ is derivable from the left annihilation law of $\mathbb{0}$ and $\otimes$, and the product differentiation axiom.

```
∂𝟘 : ⌈⌉ ▷ ⌊ ∗ ⌋ ⊢ ∂₀ 𝟘 ≈ 𝟘
∂𝟘 = begin  ∂₀ 𝟘          ≈⟨ cong[ ax 𝟘X⊗ᴸ with⟪ x₀ ⟫ ]in ∂₀ ○ ⟩s
           ∂₀ (𝟘 ⊗ x₀)    ≈⟨ ax ∂⊗ with⟪ 𝟘 ⟫ ⟩
           𝟘              ■
```

The implementation of the derived unary chain rule is more involved, but every step is necessary at the level of rigour we operate in this setting. It is the result of instantiating the binary axiom with $f(x, y) \triangleq f(x)$ and $h(x) = 0$, and applying the $\partial\mathbb{0}$ corollary above, the unit and annihilation laws for $\mathbb{0}$:

```
∂Ch₁ : ⌈ ∗ ⊩ ∗ ⌉ ⌈ ∗ ⊩ ∗ ⌉ ⊢ ∂₀ 𝔞⟨ 𝔟⟨ x₀ ⟩ ⟩ ≈ (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩)
∂Ch₁ = begin  ∂₀ 𝔞⟨ 𝔟⟨ x₀ ⟩ ⟩
        ≈⟨ ax ∂Ch₂ with⟪ 𝔞⟨ x₀ ⟩ ◃ 𝔟⟨ x₀ ⟩ ◃ 𝟘 ⟫ ⟩
            (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩) ⊕ (∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘) ⊗ ∂₀ 𝟘
        ≈⟨ cong⌊ thm ∂𝟘 ⌋in [...] ⊕ (∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘) ⊗ ○ᶜ ⟩
            (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩) ⊕ (∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘) ⊗ 𝟘
        ≈⟨ cong⌊ thm 𝟘X⊗ᴿ with⟪ ∂ 𝔞⟨ 𝔟⟨ x₁ ⟩ ⟩ ∣ 𝟘 ⟫ ⌋in [...] ⊕ ○ᶜ ⟩
            (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩) ⊕ 𝟘
        ≈⟨ ax 𝟘U⊕ᴿ with⟪ (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ ∂₀ 𝔟⟨ x₀ ⟩ ⟫ ⟩
            (∂ 𝔞⟨ x₀ ⟩ ∣ 𝔟⟨ x₀ ⟩) ⊗ (∂₀ 𝔟⟨ x₀ ⟩)      ■
```

The de Bruijn index annotations on the holes are needed when the proof operates in a nonempty metavariable context. We wrote [...] above for the sake of brevity – the context of the congruence needs to be written out explicitly for Agda to be able to evaluate the metasubstitution that instantiates the distinguished hole metavariable. Note also the use of thm, which uses an established (non-axiomatic) equality as a proof step. Thanks to the precise and sufficiently general definition of metasubstitution in our framework, metavariables and object variables are always accessible where needed, making the construction of equational proofs verbose but quite intuitive.

13.2 Generic operations

The core advantage of generic programming lies in its ability to define operations over data without requiring knowledge of the data's specific structure. In the setting of formalised calculi, this allows us to define auxiliary operations like the free variable function in a syntax-independent way: it performs the "obvious" recursive traversal, unions the free variables of subterms, and removes any that are bound. Similarly, pretty-printing is another operation that can be derived generically – provided we handle free and bound variables correctly – by specifying how each constructor should be printed. In this section, we present signature-generic definitions of both free-variable lookup and pretty-printing, all derived via initiality.

13.2.1 Free variables

Thanks to initial algebra semantics, defining a recursive function on syntactic terms amounts to equipping the codomain with a syntactic algebra structure. To derive an operation computing the set of free variables, $fv(t)$, we must express the output in a form compatible with the familial model – specifically, as a sorted family. The intrinsic representation of the set of free variables is given by a context-indexed family:

```
FV : Fam
FV Γ = List (Σ[ τ ∈ T ] ℑ τ Γ)
```

The type FV Γ represents a list of tuples consisting of sorts paired with proofs that they appear in the context Γ. While defining a free variable function might seem unnecessary for intrinsically-typed terms – since the typing context already indicates the variables in scope – the key point is that the context Γ is merely an upper bound on the free variables actually used in a term: $fv(t) \subseteq \Gamma$. In the FV family, this means that not every variable in Γ must be listed. The role of the rmVars function below is to eliminate all variables from a term that appear in a given context, relying on a list lookup function that returns a value of type Maybe.

```
rmVars : (Θ : Ctx) → FV (Θ + Γ) → FV Γ
rmVars Θ [] = []
rmVars Θ ((τ , v) :: fv) with Θ ∋? v
... | just x  = (τ , x) :: rmVars Θ fv
... | nothing = rmVars Θ fv
```

The syntactic algebra structure for FV (more precisely, the trivially sorted form $\mathsf{FV}_s\ \alpha = \mathsf{FV}$) operates as expected: a variable constructor is free, metavariables traverse the environment and concatenate the free variables, and for the algebra structure we use the following restriction to Arity, which concatenates the free variables in subterms, but removes every bound variable from the recursive list.

```
alg : (as : List (Ctx × T))(fv : Arg as FV Γ) → FV Γ
alg []          tt       = []
alg ((Θ , τ) :: as) (fv , fvs) = rmVars Θ fv ++ alg (a :: as) fvs
```

With one additional round of deduplicate, which removes duplicates from the list in case of multiple occurrences of the same variable, we obtain the free variable function $\mathsf{fv} : \mathbb{T} \to\!\!\!\!\bullet\ \mathsf{FV}_s$ as an initial interpretation into the syntactic algebra built on FV_s. For example, given the terms

```
tm₁ : Λ (N → α → α)(B · (N → α) · ∅)
tm₁ = λ (λ (nrec x₁ x₀ (x₅ $ (plus $ x₁ $ x₃))))

tm₂ : Λ (N → α → α)(B · (N → α) · ∅)
tm₂ = λ (λ (nrec x₁ x₀ (x₅ $ if x₄ ze (plus $ x₁ $ x₃))))
```

differing only in whether the last variable is used in the term or not, we get the equalities

```
fv tm₁ ≡ ((N → α , old new) :: [])      fv tm₂ ≡ ((N → α , old new) :: (B , new) :: [])
```

that differentiate between variables being listed in the context and actually used in the term.

13.2.2 Pretty-printing

Another standard operation on syntax is printing out a textual representation of the terms. For a language without binding, a straightforward fold is sufficient: if we know how to print out the leaf nodes (constants or variables) and branch nodes (operators), an in- or pre-order traversal produces the desired output. However, variable binding introduces challenges – chief

among them being the generation of fresh variable names and the maintenance of correct binding relationships – unless one is content with printing raw de Bruijn indices.

Our solution builds on the approach of Allais et al. (2021, Section 7.1), but adapts it to our framework rooted in initial-algebra semantics. Fresh variable names are generated using a stream of identifiers managed via a State monad. The pretty-printing function is then parametrised by the names assigned to the free variables in the term, and by a user-defined template function for rendering each term constructor. This template function, of type DisplayOp, is defined by pattern-matching on the operator symbol, its operands, and any bound variables associated with those operands. It produces the corresponding output string, with full access to the variable names and type annotations involved. We show the definition of the function first, then explain its components:

```
dispOp : DisplayOp
dispOp appₒ (f ⋮ (α → β) , a ⋮ α) = f ++ " (" ++ a ++ ")"
dispOp lamₒ ((x ⋮ α ⊨ b) ⋮ β)     = "(λ " ++ x ++ ":" ++ dispTy α ++ ". " ++ b ++ ")"
dispOp pairₒ (a ⋮ α , b ⋮ β)      = "⟨" ++ a ++ ", " ++ b ++ "⟩"
dispOp fstₒ  (p ⋮ α ⊗ β)          = "fst " ++ p
dispOp sndₒ  (p ⋮ α ⊗ β)          = "snd " ++ p
```

The type of pattern functions for metavariables, types, and operators is as follows:

```
DisplayMVar : Familyₛ → Set          DisplayTy : Set
DisplayMVar 𝒳 = 𝒳 ⇾ K String        DisplayTy = S → String

DisplayOp : Set
DisplayOp = (o : O) → ArgStrings (Arity o) → String
```

DisplayMVar is a family of functions from a family of metavariables to the constant family of strings. DisplayTy pattern-matches on the datatype of sorts and recursively constructs the string representation. DisplayOp works similarly, but pattern-matches on the operator symbol o, and also accepts the printed payload of the operator as a tuple of type-annotated strings with optional bound variables:

```
ArgStrings : List (Ctx × S) → Set
ArgStrings []                = ⊤
ArgStrings (([] , τ) :: as) = Ann String      τ × ArgStrings as
ArgStrings ((Θ , τ) :: as)  = Ann (Bound Θ) τ × ArgStrings as
```

If an operator is nullary, there are no printed arguments. For an n-ary operator, we return a type-annotated string, and (type-annotated) names for all variables that appear in the body.

```
data Ann (A : Set) : S → Set where
  _⋮_ : A → (τ : S) → Ann A τ
```

```
Vars : Ctx → Set
Vars ∅ = ⊤
Vars (α · Γ) = Ann String α × Vars Γ
```

```
data Bound (Γ : Ctx) : Set where
  _⊨_ : Vars Γ → String → Bound Γ
```

For example, in the case of lam_o above, the payload $((x \vdots \alpha \Vdash b) \vdots \beta)$ is a string for the body b of type β, with the bound variable in b named x, of type α. The output puts the variable name x : String and the pretty-printed type dispTy α between a "λ" and a ".", followed by the body b. The printing algorithm will ensure that the bound variable in b indeed has the name x, retaining the correct binding relationships. The rest of the constructors is similarly simple, intercalating printed subexpressions with commas, parentheses, and operator names.

Allais et al.'s (2021) trick for generating fresh variables via a stream of unique characters stored in a state monad works in our approach too: the WithStringSupply = State (Stream String) monad equips data with an infinite supply of strings, fresh generates such a string and "increments" the supply (e.g. outputs "a" and shifts the stream to continue from "b"), and freshVars turns a context into a type-annotated tuple of variable names as strings.

```
fresh : WithStringSupply String
fresh = do vs ← get
           put (tail vs)
           return (head vs)
```

```
freshVars : (Γ : Ctx) → WithStringSupply (Vars Γ)
freshVars ∅ = return tt
freshVars (α · Γ) = do v  ← fresh
                       vs ← freshVars Γ
                       return ((v ⋮ α) , vs)
```

The target of the semantics is the constant family that takes a tuple of printed free variables (in Γ) to a string with access to a supply of strings:

```
Printer : Family_S
Printer _ Γ = Vars Γ → WithStringSupply String
```

This is a pointed family, exhibited by the variable-printing function printVar:

```
printVar : ℐ ⇾ Printer
printVar new (x ⋮ α)    = return x
printVar (old v) (_ , s) = printVar v s
```

The printArgs function uses monadic do notation to process a tuple of such printer functions (one for each argument of an operator) into a printer function for the whole collection of arguments. For an argument with no binding, it prints the term in the Var context $v\Gamma$, followed by recursively processing the other arguments; for an argument that binds Θ fresh variables, it prints them using freshVars, prints the term in the context $v\Gamma$ extended with the bound variables, followed by the rest of the arguments recursively.

```
printArgs : (as : List (Ctx × S))(at : Arg as Printer Γ) →
            Vars Γ → WithStringSupply (ArgStrings as)
printArgs [] tt vΓ = return tt
printArgs ((∅ , τ) :: as) (tp , tps) vΓ = do ts ← tp vΓ
```

```
                                                  tss ← printArgs as tps vΓ
                                                  return ((ts ⁝ τ) , tss)
printArgs ((Θ , τ) :: as) (tp , tps) vΓ = do bound ← freshVars Θ
                                             body ← tp (bound ++ᵛ vΓ)
                                             tss ← printArgs as tps vΓ
                                             return ((bound ⊨ body) ⁝ τ , tss)
```

Finally, printing of metavariables uses the following function that prints their environment. Note that the variable context of the output (containing the names of the variables in scope) is fed into the recursive printing of the environment terms, ensuring that all the de Bruijn indices get translated into their appropriate string representation.

```
printEnv : (Π : Ctx) → (Π –[ Printer ]→ Γ) → Printer Γ
printEnv ∅       σ vΓ = return ""
printEnv (α · Π) σ vΓ = do ts ← σ new vΓ
                           tss ← printEnv Π (σ ∘ old) vΓ
                           return (ts ++ " ; " ++ tss)
```

With these, a SynAlg instance for Printer can be defined as follows, parametrised by pattern functions for operators and metavariables:

```
Printerᵃ : (dop : DisplayOp) (dmv : DisplayMVar 𝒳) → SynAlg 𝒳 Printer
Printerᵃ 𝒳 dop dmv = record
  { v = printVar
  ; a = λ{ (o ≀ as) vΓ → do pargs ← printArgs (Arity o) as vΓ
                            return (dop o pargs) }
  ; m = λ {Π} 𝔞 ε  vΓ → do envs ← printEnv Π ε vΓ
                           return (dmv 𝔞 ++ "[" ++ envs ++ "]") }
```

The semantic interpretation of terms into Printers is parametrised by the variable names in the typing context, and returns a State monad whose first component also expects an infinite supply of fresh variable names, which we provide as a stream of characters with an optional numeric suffix to guarantee uniqueness.

```
genPrint : (dop : DisplayOp) (dmv : DisplayMVar 𝒳) → 𝕋 𝒳 α Γ → Vars Γ → String
genPrint 𝒳 dop dmv t vΓ = proj₁ (𝕚 (Printerᵃ dop dmv) t vΓ names)
```

This generic printing operation can be specialised to a particular syntax by providing an operator pattern function. The DisplayMVar argument can be can be instantiated for inductive metavariable contexts $\mathfrak{M} : \mathsf{MCtx}$ by translating de Bruijn indices into alphabetic characters "M", "N", etc. For example, the term $\mathsf{tm}_1 = \lambda\,(\lambda\,(\mathsf{nrec}\ \mathsf{x}_1\ \mathsf{x}_0\ (\mathsf{x}_5\ \$\ (\mathsf{plus}\ \$\ \mathsf{x}_1\ \$\ \mathsf{x}_3))))$ pretty-prints to the following (line breaks added for clarity):

```
_ : print tm₁ (("x" ⁝ B) , ("y" ⁝ N → 𝟙))
  ≡ "▷ x : 𝔹, y : (ℕ → 𝟙)
     ⊢ (λ a:ℕ. (λ b:𝟙. nrec[𝟙] (a; ⊙ ⟶ b | (d + 1) c ⟶
       y ((λ e:ℕ. (λ f:ℕ. nrec[ℕ] (e; ⊙ ⟶ f | (h + 1) g ⟶ g + 1)))
         (d) (a))))) : (ℕ → (𝟙 → 𝟙))"
_ = refl
```

The two free variables in the context $(\mathsf{B} \cdot (\mathsf{N} \to \alpha) \cdot \emptyset)$ are given names "x" and "y", and all other new variables are given fresh names taken from the stream names. The term plus is merely an Agda definition, so the primitive recursive printing function expands it fully; on the other hand, DispTy recurses explicitly, and we are able to "intercept" the type $\mathbb{1} \oplus \mathbb{1}$ as a special case of the printing for $\oplus$, and return the string "$\mathbb{B}$" instead.

For a term in a nonempty metavariable context, consider:

$$\begin{aligned}\mathsf{tm}_3 &: (\llbracket\, \mathsf{N} \Vdash \beta \,\rrbracket\ \llbracket\, \alpha \,\rrbracket \rhd \Lambda)\ \beta\ ((\alpha \to \mathsf{N}) \cdot \emptyset)\\ \mathsf{tm}_3 &= (\lambda\ \mathfrak{a}\langle\ (\mathsf{x}_1\ \$\ \mathsf{x}_0)\ \rangle)\ \$\ \mathfrak{b}\end{aligned}$$

```
_ : print tm₃ ("f" : B → N)
  ≡ "M : ℕ ⊩ 𝔹 ; N : ⊩ 𝔹 ▷ f : (𝔹 → ℕ) ⊢ (λ a:𝔹. M[f (a)]) (N[]) : 𝔹"
_ = refl
```

As the examples in this section show, the computational behaviour of semantic interpretations aligns with expectations: terms are represented as syntax trees made out of constructors, and they are consumed by structurally recursive functions. The ability to interpret terms in families facilitates context-sensitive interpretations that take variables, binding, etc. into account.

Summary of Part IV

In this part we introduced the our Agda language formalisation library and showcased some applications of the familial model through the generic mechanisation of second-order abstract syntax.

CHAPTER 14

Conclusions

This thesis introduced the familial model of second-order abstract syntax – an implementation-friendly metatheoretic framework derived from the presheaf approach. We formally motivated our solutions to reconcile discrepancies between the two models, grounding widely used but often opaque techniques in intrinsically-typed formalisation on solid mathematical foundations. Below, we summarise the main contributions and outline directions for future research.

14.1 Summary of contributions

We highlight the questions that the familial model answers both from a language formalisation and model-theoretic perspective.

Why bother with intrinsic typing? Where available, intrinsic typing offers the rigorous discipline essential for formal verification. Instead of promising uninterrupted progress in formalising a syntax and semantics only to demand laborious sort-preservation and soundness proofs later, intrinsic typing is clear and upfront: every term must be well-sorted and well-scoped, and every operation must preserve these properties. As a result, well-sortedness is built into definitions, streamlining formalisation and reducing metatheoretic overhead. The familial model naturally supports this approach: elements of a sorted family set encode their sort and scope at the type level, ensuring that ill-sorted terms and operations are caught as type errors in the formalisation language.

Why bother with a formalisation framework? Intrinsically-typed encodings of syntax offer striking directness and elegance: term constructors are typed by their sorting rules, making ill-sorted terms unrepresentable. This extends to syntactic operations, which are typed by their sort-preservation laws. Defining such operations typically requires additional lemmas – structural properties, compatibility conditions, fusion laws, etc. While these proofs often proceed by induction on the term grammar, the structure is formulaic: structural recursion with careful handling of bound variables. A capable formalisation framework can automate much of this routine, freeing attention for the truly challenging aspects of verification.

The familial model's abstract theory encapsulates the metatheoretic boilerplate – renaming, substitution, denotational interpretation, and their correctness proofs – within a mathematically grounded framework of modules, strong signature endofunctors, and algebraic monoids. Rather than relying on an ad hoc collection of definitions and lemmas, the model's components build coherently on one another, enabling modular interaction with broader categorical tools. For instance, the complex compatibility conditions between renaming, strength, substitution, and metasubstitution arise as instances of a general lifting framework in synthetic monoidal categories (see Theorem 10.3.3).

Why not just formalise the presheaf model? The presheaf model of Fiore et al. (1999) and Fiore (2008) offers a comprehensive algebraic account of second-order abstract syntax. However, its reliance on categorical tools such as colimits, coends, and quotienting makes it impractical for formalisation in dependently typed proof assistants, which lack native support for these constructs without resorting to cumbersome workarounds like setoids, or extensions such as cubical type theory. The familial model reformulates the presheaf approach to address these limitations, making it suitable for implementation and practical metatheory development with predictable computational behaviour. For instance, instead of relying on Adámek-style constructions of initial algebras, the syntax of a signature is represented directly as an algebraic data type, allowing terms to be observable and amenable to syntactic transformation.

What are the discrepancies between the models? The main limitation of the familial approach lies in its weaker substitution structure compared to presheaves, which disrupts many constructions and results from the original model. This thesis reassembles these elements into a coherent and practical framework, addressing the following challenges and questions:

- No inherent renaming structure in families
  → Families can be equipped with a module structure equivalent to presheaves (see Section 10.1.3)
- Substitution tensor in families is skew-monoidal and does not lift to pointed modules
  → The correct structure is that of a synthetic monoidal category (see Section 6.1)
- Laws involving the substitution tensor fail without quotienting
  → These laws can be reformulated using multilinear maps that extensionalise renaming-invariance (see Section 10.2.1)
- Pointed strength of signature endofunctors cannot be axiomatised directly as a functor in skew modular categories
  → It is a homomorphism between synthetic modular categories, with multiplication morphisms defined via multilinear maps (see Section 10.2.2)
- Relationship between signature endofunctors in presheaves and families
  → $\Sigma\colon \mathbf{Fam}_S \to \mathbf{Fam}_S$ lifts to $\Theta\colon \mathbf{PSh}_S \to \mathbf{PSh}_S$ in the category of synthetic monoidal categories (see Section 6.3)
- Relationship between categories of semantic models for related signature endofunctors in presheaves and families

→ A synthetic strong family endofunctor lifts to presheaves, and the categories of algebraic monoids are equivalent (see Theorem 10.2.2)

- Relationship between initial syntactic algebras for related signature endofunctors
  → The initial algebra in families induces the corresponding initial algebra in presheaves (see Section 3.3)
- Parametrised initiality does not directly extend to skew-monoidal categories
  → Initial-algebra semantics into the closed substitution structure avoids issues due to skew associativity (see Section 11.2)
- The cartesian closed structure of families cannot express enriched Kleisli extension for capture-permitting metasubstitution
  → The term monad in families is enriched over the Day internal hom derived from context extension (see Section 11.3.1)

14.2 Future directions

The work presented in this thesis forms the foundation of a multitude of avenues for further research, being the first step towards bringing the theory of presheaf models into a streamlined categorical setting. The abstract perspective also suggested some categorical generalisations that embed the current second-order (skew-)monoidal models into a wider setting.

14.2.1 Advanced type theories

Some extensions involve incorporating advanced type-theoretic features into the presheaf or familial models, making the approach general enough to capture a variety of syntaxes that appear in literature.

Substructural syntax Both the presheaf and familial models are cartesian in that their base category of contexts and renamings admits cocartesian structure, enabling structural lemmas like weakening, exchange, and contraction. However, *substructural* type theories impose stricter constraints– linear type theory, for instance, permits only exchange, forbidding duplication or discarding of variables. This restriction must be reflected in the base category itself. In linear syntax, each bound variable must be used exactly once, deeply affecting the behaviour of binding and the structure of metatheoretic proofs.
Tanaka (2000) developed a presheaf model of abstract syntax with linear binders by working over the free *strict symmetric monoidal* category on sorts. This base supports reassociation and permutation, but not changes in context length. In this setting, a presheaf over linear contexts encodes terms $\Gamma \vdash t : \tau$ where the set of free variables in t is *exactly* Γ. Variables are typed in singleton contexts, and application combines the disjoint contexts of two terms. Notably, Fiore and Ranchod (2024) recently resolved the long-standing challenge of encoding single-variable linear substitution, once thought impossible by Tanaka.
Given the prominence of linear type theories, extending the familial model to linear abstract syntax is a promising direction for future work– especially since its base category, the free *strict monoidal* category, is even weaker than the symmetric monoidal base used by Tanaka.

Further exploration of Day convolution to model context-wise extension and substitution could also clarify the most suitable formalisation techniques for linear binding.

Polymorphic syntax Another axis for enriching type theories is to introduce variables and binding at the sort level, enabling the modelling of languages with recursive types, Hindley–Milner-style type schemes, full polymorphism, and type constructors. These extensions were integrated into the presheaf model by Hamana (2011) and Fiore and Hamana (2013) through a two-step process. First, the presheaf construction is used to define the presheaf of simply-kinded types $\mathbb{T} \in \widetilde{\mathbb{F}[\mathbb{1}]}$ as the initial algebra of a type signature endofunctor. Then, the category of term contexts is constructed via a Grothendieck construction $\int \mathbb{F}\mathbb{T}(-) \times \mathbb{T}(-)$, whose objects are triples (n, Γ, α): a type variable context $[n]$ (representing n free type variables), a term variable context $\Gamma \in \mathbb{F}[\mathbb{T}(n)]$, and a sort $\alpha \in \mathbb{T}(n)$. This forms the base category for the presheaf of polymorphically-typed terms, where elements $n \mid \Gamma \vdash t \in \alpha$ are terms $t \in \mathcal{P}(n, \Gamma, \alpha)$ of sort α in the variable context Γ. Morphisms in the Grothendieck category act as simultaneous renamings of type and term variables. Hamana (2011) further generalised this framework to higher-kinded polymorphism, and Fiore and Hamana (2013) extended it to a full second-order abstract syntax treatment, incorporating type and term metavariables, type-in-term substitution, polymorphic equational logic, and more. Addressing all these developments in a formalisable setting would be a powerful, if challenging extension: as a polymorphic type system involves metasyntactic computation (substitution), encoding such languages intrinsically is challenging (Chapman et al., 2019).

Multi-context calculi Judgments in modern type theories often encode rich information – indices, annotations, labels, effects, stores, and more. While some of this can be absorbed into the (often unstructured) sort, other aspects require multiple typing contexts to track different categories of variables. For instance, dual-context calculi arise frequently in modal type theories (Kavvos, 2017), mixed linear–nonlinear systems (Benton, 1995), classical type theories (Curien and Herbelin, 2000), and most realistic formal systems beyond toy languages. Supporting such systems involves working with presheaves over product categories of contexts, where each component may have its own structural discipline – linear, cartesian, monoidal, or otherwise – depending on the needs of the theory.

Mutually recursive terms The original presheaf/familial models only support one grammar of terms, but we can extend support to mutually recursive term grammars using mutually recursive signature endofunctors. Fiore (2022) applies this approach for an algebraic analysis of normalisation by evaluation (Berger and Schwichtenberg, 1991) to encode the mutually recursive definition of normal and neutral terms, but the technique should be applicable to other multi-judgment systems like monadic calculi or call-by-push-value (Levy, 1999).

Pattern binding Second-order abstract syntax does not directly support a mechanism widely used in languages with binding: *pattern matching*. It arises naturally whenever we program in a language with compound algebraic data types, replacing elimination forms like projection and explicit case analysis with binding-time term decomposition and pattern-matching

binders. For example, the function $(\alpha \times \beta) + (\alpha \times \gamma) \to \alpha \times (\beta + \gamma)$, written as

$$\lambda p.\ \mathsf{case}\ p\ \mathsf{of}\ (\mathsf{inl}\, x) \mapsto (\mathsf{fst}\, x, \mathsf{inl}\, (\mathsf{snd}\, x)) \ \mid\ (\mathsf{inr}\, y) \mapsto (\mathsf{fst}\, y, \mathsf{inr}\, (\mathsf{snd}\, y))$$

could instead be written in a more concise pattern-matching syntax as

$$\lambda\ \mathsf{inl}(x, y) \mapsto (x, \mathsf{inl}\, y) \ \mid\ \mathsf{inr}(x, z) \mapsto (x, \mathsf{inr}\, z)$$

Abstraction operators in the presheaf and familial models only allow the binding of a finite sequence of variables without any further structure. Fiore (2019) sketched out a proposed extension of the presheaf model with pattern matching, combining a linear grammar of patterns with a cartesian grammar of terms, interpreting the former in the category $\mathbb{B}$ of finite sets and bijections to ensure linear use of pattern variables. It would be worthwhile to flesh out the details, and explore interesting connections to *combinatorial species of structure* (Joyal, 1981; Bergeron et al., 1997) that connect pattern descriptions to algebraic datatypes.

Parametric signatures and translations The framework generates second-order abstract metatheory for a single second-order signature; this poses a limitation for systems that are themselves parametrised by a grammar of symbols, such as first-order logic parametrised over a first-order language of function and relation symbols. Fortunately, second-order abstract syntax is general enough to capture such first-order theories using parametrised metavariables, and a useful extension to the familial model would be a wide-ranging support for parametric signatures, and more generally, signature translations or syntactic translations as introduced by Fiore and Mahmoud (2010).

14.2.2 Structural generalisations

The following avenues of investigation involve significant categorical redevelopments of presheaf models, be they the original approach of Fiore et al. (1999) or the familial model of this thesis. They may be more of theoretical than practical interest, exploring extensions or formulations discovered upon an abstract look at the algebraic theory of syntax.

Promonoidal categories and multicategories A central technical challenge in the familial model is that the substitution tensor product $\oplus \colon \mathbf{Fam}_S \times \mathbf{Fam}_S \to \mathbf{Fam}_S$ fails to lift to pointed modules (i.e. pointed $\diamond$-algebras or $\Box$-coalgebras). Even the skew-monoidal structure requires quotienting, which is not directly available. We addressed this using *synthetic monoidal categories*, but late into the research some useful avenues for generalisation became evident, both addressing the situation of coherently combining objects of a category without the presence of a formal monoidal structure.

Synthetic monoidal structure is likely expressible via *promonoidal categories* (Day, 1970), which replace hom-sets with profunctors $\mathbf{1}^{\mathrm{op}} \times \mathcal{V} \to \mathbf{Set}$ and $(\mathcal{V} \times \mathcal{V})^{\mathrm{op}} \times \mathcal{V} \to \mathbf{Set}$. Though the laws involved are syntactically complex, they reduce to properties similar to those in Section 6.1 and interact well with our definitions of pointed multilinear maps. For the familial model, we likely require a weakening to skew-promonoidal categories (Street, 2013) or skew proactegories (Campbell, 2018), where isomorphisms are replaced by directed transformations.

A related generalisation is the use of *multicategories* (Hermida, 2000; Leinster, 2004), which avoid tensor products entirely by working with morphisms of the form $A_1, \ldots, A_n \to B$. These are especially suitable when a tensor product like $A_1 \otimes \cdots \otimes A_n$ is undefined. We conjecture that the most natural setting for formalising multilinear maps – and hence for axiomatising the familial model – is within a (skew) promonoidal or multicategorical framework.

Finite-order abstract syntax Second-order abstract syntax enables the representation of terms that bind first-order object variables – such as in λ-abstraction, logical quantification, and partial differentiation – with substitution and renaming primarily acting on these first-order variables. While SOAS also accommodates parametrised metavariables, along with metarenaming and metasubstitution, it does not support *binding* of metavariables. That is, second-order signatures restrict binding to variables within operands, precluding native expression of operators that bind other operators. A typical example of such a third-order construct is a control operator like `call/cc` or $\mathcal{C}$, as described by Felleisen and Friedman (1987) and Griffin (1989), which binds a continuation – a unary operator from a target sort to a resumption sort. These kinds of *higher-order signatures* are studied in detail by Arkor (2022, Chapter 4).

We encountered a specific instance of this in *contextual modal type theory* (CMTT) (Nanevski et al., 2008) – an extension of modal type theory (Pfenning and Davies, 2001) with context-indexed modalities – while co-supervising a Master's project with Marcelo Fiore (Chekroun, 2022). In CMTT, parametrised metavariables can be explicitly bound and instantiated as terms sorted in a local parameter context. Extending the familial model to support this requires generalising to categories of terms indexed by sorts, contexts, and metavariable contexts. This leads naturally to a stratified framework for higher-order syntax. Fixing a set of sorts S, we define a tower of base categories:

$$\mathbb{C}_0 \triangleq S \qquad \mathbb{C}_{n+1} \triangleq \mathbb{C}_n^* \times \mathbb{C}_n$$

Presheaves over $\mathbb{C}_n$ then describe $(n+1)^{\text{th}}$-order syntaxes, capable of binding variables, variable-binding terms (i.e. parametrised metavariables), metavariable-binding terms, and so on. We conjecture that generalising the substitution constructions of Chapter 9 to arbitrary levels of this tower will recover the familiar substitution and metasubstitution structure of SOAS in a uniform and elegant way.

Bicategorical syntax The current framework for simply-sorted second-order abstract syntax assumes a single grammar of sorts, used both to type terms and to annotate variables in sorting contexts. Concretely, a sorted presheaf $\mathbf{PSh}_S \coloneqq \widetilde{\mathbb{F}[S]}^S$ can be viewed equivalently as a presheaf over $\mathbb{F}[S] \times S$, where both variables and terms are sorted using the same set S. To handle systems in which the grammar of variable sorts differs from that of term sorts, we can decouple the two by considering presheaves over $\mathbb{F}[S] \times T$, for distinct sort sets S and T. This introduces the complexity of *heterogeneous substitution*: a T-sorted term in an S-sorted context can only be substituted with S-sorted terms, potentially in a different R-sorted context.

Accommodating this structure requires generalising the substitution monoidal structure to a (skew) *bicategorical* framework (Lack and Street, 2014[a]), where the skew monoidal tensor becomes a heterogeneous composition equipped with skew unit and associator 2-cells. To

simultaneously address both bicategorical and multicategorical generalisation, the natural setting may be that of ***fc**-multicategories* (Leinster, 1999) – a direction first proposed by Ohad Kammar in private communication.

14.3 Final remarks

Reading back over the proposal and initial progress reports that preceded this thesis, I was struck by their carefully optimistic, yet grounded tone – describing "the (admittedly rather ambitious) dream of a general framework that can transform a syntactic specification of a second-order language into a comprehensive system for reasoning about the language, without the distractions of defining weakening, substitution, renaming, and proving all the associated correctness lemmas." Clearly, I underestimated the power of syntax and initiality.

Throughout the development of the Agda library, I encountered several critical decision points that required choosing between pragmatic implementation and mathematical fidelity within the limitations of dependently-typed programming. These included axiomatising multilinear maps rather than rewriting the library using setoids to represent the coend of the presheaf substitution tensor product; using standard inductive types to define languages instead of encoding term algebras as containers with sized types (to appease the termination checker); and opting for Python-generated code rather than undertaking the overhead of verified Agda metaprogramming. In many cases, what began as pragmatic "workarounds" turned out to be mathematically natural and conceptually well-motivated, opening fruitful directions in the algebraic theory of presheaf models. These decisions did not compromise the foundational goals; rather, they broadened the theoretical scope and highlighted new possibilities.

This thesis lays the groundwork for a broader programme of research, offering not only concrete constructions but also a toolkit of principles for extending the categorical model of second-order abstract syntax. By bridging abstract mathematics with practical formalisation, our work envisions a future where syntax formalisation is no longer a significant burden – freeing language designers to focus on expressiveness, safety, and usability, with the benefits flowing naturally into the next generation of programming tools.

Bibliography

Abadi, Martin, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy (1991)
Explicit Substitutions
In: *Journal of Functional Programming* 1 (4), pp. 375–416.
doi: 10.1017/S0956796800000186 53, 246 ↑

Abel, Andreas (2010)
MiniAgda: Integrating Sized and Dependent Types.
arxiv: 1012.4896[cs.PL] 264 ↑

Abel, Andreas, Ralph Matthes, and Tarmo Uustalu (2005)
Iteration and Coiteration Schemes for Higher-Order and Nested Datatypes
In: Proceedings of the 6th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2005) 333 (1), pp. 3–66.
doi: 10.1016/j.tcs.2004.10.017 57 ↑

Abramsky, Samson, Dan R. Ghica, Andrzej S. Murawski, C. H. Luke Ong, and Ian D. B. Stark (2004)
Nominal Games and Full Abstraction for the Nu-Calculus
In: *Proceedings of the 19th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2004)*.
doi: 10.1109/lics.2004.1319609 51 ↑

Aczel, Peter (1978)
A General Church–Rosser Theorem. Tech. rep. University of Manchester.
url: www.ens-lyon.fr/LIP/REWRITING/MISC/AGeneralChurch-RosserTheorem.pdf 213, 262 ↑

Ahrens, Benedikt (2012)
Extended Initiality for Typed Abstract Syntax
In: *Logical Methods in Computer Science* 8 (2), p. 1.
doi: 10.2168/LMCS-8(2:1)2012 56 ↑

Ahrens, Benedikt (2015)
Initiality for Typed Syntax and Semantics. PhD thesis. University of Birmingham, pp. 1–155.
doi: 10.6092/issn.1972-5787/4712 56 ↑

Ahrens, Benedikt (2016)
Modules over relative monads for syntax and semantics
In: *Mathematical Structures in Computer Science* 26 (1), pp. 3–37.
doi: 10.1017/S0960129514000103 55, 56 ↑

Ahrens, Benedikt, André Hirschowitz, Ambroise Lafont, and Marco Maggesi (2019)
Reduction Monads and Their Signatures
In: *Proceedings of the ACM on Programming Languages* 4 (POPL), 31:1–31:29.
doi: 10.1145/3371099 56 ↑

Ahrens, Benedikt, André Hirschowitz, Ambroise Lafont, and Marco Maggesi (2021)
Presentable signatures and initial semantics
In: *Logical Methods in Computer Science* Volume 17, Issue 2.
doi: 10.23638/LMCS-17(2:17)2021 55, 56 ↑

Ahrens, Benedikt and Ralph Matthes (2018)
Heterogeneous Substitution Systems Revisited
In: *Proceedings of the 21st International Conference on Types for Proofs and Programs (TYPES 2015)*. Vol. 69. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2:1–2:23.
DOI: 10.4230/LIPIcs.TYPES.2015.2 57 ↑

Ahrens, Benedikt and Julianna Zsido (2011)
Initial Semantics for higher-order typed syntax in Coq.
ARXIV: 1012.1010[cs.LO] 55 ↑

Allais, Guillaume, Robert Atkey, James Chapman, Conor McBride, and James McKinna (2021)
A type- and scope-safe universe of syntaxes with binding: their semantics and proofs
In: *Journal of Functional Programming* 31, e22.
DOI: 10.1017/S0956796820000076 33–35, 39, 120, 167, 247, 250, 253, 254, 261, 264, 267, 268, 278, 279 ↑

Allais, Guillaume, James Chapman, Conor McBride, and James McKinna (2017)
Type-and-Scope Safe Programs and Their Proofs
In: *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)*. ACM Press, pp. 195–207.
DOI: 10.1145/3018610.3018613 33 ↑

Altenkirch, Thorsten (1993)
A Formalization of the Strong Normalization Proof for System F in LEGO
In: *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications (TLCA 1993)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 13–28.
DOI: 10.1007/BFb0037095 52 ↑

Altenkirch, Thorsten, James Chapman, and Tarmo Uustalu (2010)
Monads Need Not Be Endofunctors
In: *Proceedings of the 13th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2010)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 297–311.
DOI: 10.1007/978-3-642-12032-9_21 55, 96, 109, 172 ↑

Altenkirch, Thorsten, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris (2015)
Indexed containers
In: *Journal of Functional Programming* 25, e5.
DOI: 10.1017/S095679681500009X 261, 267 ↑

Altenkirch, Thorsten and Bernhard Reus (1999)
Monadic Presentations of Lambda Terms Using Generalized Inductive Types
In: *Proceedings of the 13th International Workshop on Computer Science Logic (CSL 1999)*. Vol. 1683. Lecture Notes in Computer Science (LNCS). Springer, pp. 453–468.
DOI: 10.1007/3-540-48168-0_32 54, 253 ↑

Anand, Abhishek and Vincent Rahli (2014)
A Generic Approach to Proofs about Substitution
In: *Proceedings of the 9th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2014)*. ACM, pp. 1–8.
DOI: 10.1145/2631172.2631177 52 ↑

Arkor, Nathanael (2018)
Formal Abstract Syntax for Type Theories. MA thesis. University of Cambridge 217 ↑

Arkor, Nathanael (2022)
Monadic and Higher-Order Structure. PhD thesis. University of Cambridge.
DOI: 10.17863/CAM.86347 288 ↑

Arkor, Nathanael and Marcelo Fiore (2020)
Algebraic Models of Simple Type Theories: A Polynomial Approach
In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2020)*.

ACM Press, pp. 88–101.
DOI: 10.1145/3373718.3394771 58, 181, 261↑

Arkor, Nathanael and Dylan McDermott (2021)
Abstract Clones for Abstract Syntax
In: *Proceedings of the 6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*. Vol. 195. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 30:1–30:19.
DOI: 10.4230/LIPIcs.FSCD.2021.30 53, 84↑

Aydemir, Brian, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic (2005)
Mechanized Metatheory for the Masses: The PoplMark Challenge
In: *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2005)*. Springer, pp. 50–65.
DOI: 10.1007/11541868_4 50, 52↑

Aydemir, Brian, Aaron Bohannon, and Stephanie Weirich (2007)
Nominal Reasoning Techniques in Coq
In: *Electronic Notes in Theoretical Computer Science* 174 (5), pp. 69–77.
DOI: 10.1016/j.entcs.2007.01.028 51↑

Aydemir, Brian, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich (2008)
Engineering Formal Metatheory
In: *ACM SIGPLAN Notices* 43 (1), pp. 3–15.
DOI: 10.1145/1328897.1328443 61↑

Aydemir, Brian and Stephanie Weirich (2010)
LNgen: Tool Support for Locally Nameless Representations. Tech. rep. 933.
URL: repository.upenn.edu/handle/20.500.14332/7902 61↑

Backhouse, Roland, Marcel Bijsterveld, Rik van Geldrop, and Jaap van der Woude (1995)
Categorical fixed point calculus
In: *Category Theory and Computer Science*. Springer, pp. 159–179.
DOI: 10.1007/3-540-60164-3_25 71↑

Baez, John C. and James Dolan (1998)
Higher-Dimensional Algebra III. n-Categories and the Algebra of Opetopes
In: *Advances in Mathematics* 135 (2), pp. 145–206.
DOI: https://doi.org/10.1006/aima.1997.1695 110↑

Barendregt, Hendrik Pieter (1985)
The lambda calculus - its syntax and semantics. Vol. 103. Studies in logic and the foundations of mathematics. North-Holland. ISBN: 978-0-444-86748-3 50↑

Beck, Jon (1968)
~~The Tripleableness Theorem~~ Untitled manuscript. Distributed at the *Conference Held at the Seattle Research Center of the Battelle Memorial Institute*.
URL: ncatlab.org/nlab/files/Beck_MonadicityTheorem.pdf 176↑

Beck, Jon (1969)
Distributive Laws
In: *Seminar on Triples and Categorical Homology Theory*. Lecture Notes in Mathematics 80. Republished in *Reprints in Theory and Applications of Categories* 18 (2008). Springer, pp. 96–112.
DOI: 10.1007/BFb0083084 65, 66, 68↑

Bellegarde, Françoise and James Hook (1994)
Substitution: A Formal Methods Case Study Using Monads and Transformations
In: *Science of Computer Programming* 23 (2-3), pp. 287–311.
DOI: 10.1016/0167-6423(94)00022-0 53↑

Bengtson, Jesper and Joachim Parrow (2007)
Formalising the π-Calculus Using Nominal Logic
In: *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2007)*. Vol. 4423. Springer, pp. 63–77.
DOI: 10.1007/978-3-540-71389-0_6 51 ↑

Benke, Marcin, Peter Dybjer, and Patrik Jansson (2003)
Universes for Generic Programs and Proofs in Dependent Type Theory
In: *Nordic Journal of Computing* 10 (4), pp. 265–289 267 ↑

Benton, Nick (1995)
A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models
In: *Computer Science Logic*. Vol. 933, pp. 121–135.
DOI: 10.1007/BFb0022251 286 ↑

Benton, Nick, Chung-Kil Hur, Andrew J Kennedy, and Conor McBride (2012)
Strongly typed term representations in Coq
In: *Journal of Automated Reasoning* 49 (2), pp. 141–159.
DOI: 10.1007/s10817-011-9219-0 34, 167, 250, 253 ↑

Berger, Ulrich and Helmut Schwichtenberg (1991)
An Inverse of the Evaluation Functional for Typed λ-calculus
In: *Proceedings of the 6th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 1991)*. Ludwig-Maximilians-Universität München, pp. 203–211.
DOI: 10.5282/ubm/epub.4261 286 ↑

Bergeron, François, Gilbert Labelle, and Pierre Leroux (1997)
Combinatorial Species and Tree-like Structures. Encyclopedia of Mathematics and its Applications. Cambridge University Press 167, 287 ↑

Berghofer, Stefan (2012)
A Solution to the PoplMark Challenge Using de Bruijn Indices in Isabelle/HOL
In: *Journal of Automated Reasoning* 49 (3), pp. 303–326.
DOI: 10.1007/s10817-011-9231-4 52 ↑

Berghofer, Stefan and Christian Urban (2007)
A Head-to-Head Comparison of de Bruijn Indices and Names
In: *Electronic Notes in Theoretical Computer Science* 174 (5), pp. 53–67.
DOI: 10.1016/j.entcs.2007.01.018 52 ↑

Bird, Richard and Lambert Meertens (1998)
Nested Datatypes
In: *Proceedings of the 4th International Conference on the Mathematics of Program Construction (MPC 1998)*. Vol. 1422. Springer, pp. 52–67.
DOI: 10.1007/BFb0054285 54 ↑

Bird, Richard and Ross Paterson (1999a)
De Bruijn Notation as a Nested Datatype
In: *Journal of Functional Programming* 9 (1), pp. 77–91.
DOI: 10.1017/S0956796899003366 54, 253, 254 ↑

Bird, Richard and Ross Paterson (1999b)
Generalised Folds for Nested Datatypes
In: *Formal Aspects of Computing* 11 (2), pp. 200–222.
DOI: 10.1007/s001650050047 54, 56, 57 ↑

Blanchette, Jasmin Christian, Lorenzo Gheri, Andrei Popescu, and Dmitriy Traytel (2019)
Bindings as Bounded Natural Functors
In: *Proceedings of the ACM on Programming Languages* 3 (POPL).
DOI: 10.1145/3290335 51 ↑

Booker, Thomas and Ross Street (2013)
Tannaka duality and convolution for duoidal categories
In: *Theory and Applications of Categories* 28 (6), pp. 166–205.
URL: www.tac.mta.ca/tac/volumes/28/6/28-06.pdf 137↑

Borthelle, Peio, Tom Hirschowitz, and Ambroise Lafont (2020)
A Cellular Howe Theorem
In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2020)*. ACM Press, pp. 273–286.
DOI: 10.1145/3373718.3394738 96, 124, 207, 217, 219↑

Bosman, Roger, Georgios Karachalias, and Tom Schrijvers (2023)
No Unification Variable Left Behind: Fully Grounding Type Inference for the HDM System
In: *Proceedings of the 14th International Conference on Interactive Theorem Proving (ITP 2023)* 268, 8:1–8:18.
DOI: 10.4230/LIPICS.ITP.2023.8 61↑

Bunge, Marta (1966)
Categories of Set-Valued Functors. PhD thesis. University of Pennsylvania.
URL: ncatlab.org/nlab/files/Bunge-CategoriesOfSetValuedFunctors.pdf 151↑

Busenius, Alex (2011)
Mechanized Formalization of a Transformation from an Extensible Spi Calculus to Java. MA thesis. Saarland University 61↑

Campbell, Alexander (2018)
Skew-Enriched Categories
In: *Applied Categorical Structures* 26 (3), pp. 597–615.
DOI: 10.1007/s10485-017-9504-0 100, 103, 287↑

Capucci, Matteo and Bruno Gavranović (2022)
Actegories for the Working Amthematician
In: *arXiv e-prints*, arXiv:2203.16351.
arXiv: 2203.16351[math.CT] 100↑

Chapman, James, Pierre-Évariste Dagand, Conor McBride, and Peter Morris (2010)
The gentle art of levitation
In: *ACM SIGPLAN Notices* 45 (9), pp. 3–14.
DOI: 10.1145/1932681.1863547 33, 267↑

Chapman, James, Roman Kireev, Chad Nester, and Philip Wadler (2019)
System F in Agda, for Fun and Profit
In: *Mathematics of Program Construction*. Vol. 11825. Springer, pp. 255–297.
DOI: 10.1007/978-3-030-33636-3_10 286↑

Charguéraud, Arthur (2012)
The locally nameless representation
In: *Journal of Automated Reasoning* 49 (3), pp. 363–408.
DOI: 10.1007/s10817-011-9225-2 61↑

Chekroun, Nissim (2022)
Formalising Contextual Modal Typed Calculi. MA thesis. University of Cambridge 288↑

Chlipala, Adam (2008)
Parametric Higher-Order Abstract Syntax for Mechanized Semantics
In: *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP 2008)*. ACM Press, pp. 143–156.
DOI: 10.1145/1411204.1411226 60↑

Choudhury, Pritam (2015)
Constructive Representation of Nominal Sets in Agda. MA thesis. University of Cambridge.
URL: cl.cam.ac.uk/~amp12/agda/choudhury/choudhury-dissertation.pdf 51, 52↑

Church, Alonzo (1932)
A Set of Postulates for the Foundation of Logic
In: *Annals of Mathematics* 33 (2), pp. 346–366.
DOI: 10.2307/1968337. JSTOR: 1968337 50↑

Church, Alonzo (1936)
An Unsolvable Problem of Elementary Number Theory
In: *American Journal of Mathematics* 58 (2), pp. 345–363.
DOI: 10.2307/2371045. eprint: 2371045 50↑

Church, Alonzo (1940)
A Formulation of the Simple Theory of Types
In: *The Journal of Symbolic Logic* 5 (2), pp. 56–68.
DOI: 10.2307/2266170 59↑

Clouston, Ranald A. (2010)
Binding in Nominal Equational Logic
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010) 265, pp. 259–276.
DOI: 10.1016/j.entcs.2010.08.016 51↑

Clouston, Ranald A. and Andrew M. Pitts (2007)
Nominal Equational Logic
In: *Electronic Notes in Theoretical Computer Science*. Computation, Meaning, and Logic: Articles Dedicated to Gordon Plotkin 172, pp. 223–257.
DOI: 10.1016/j.entcs.2007.02.009 51↑

Cockett, J. Robin B. (Oct. 1990)
List-Arithmetic Distributive Categories
In: *Journal of Pure and Applied Algebra* 66 (1), pp. 1–29.
DOI: 10.1016/0022-4049(90)90121-W 41↑

Cockx, Jesper (2021)
1001 Representations of Syntax with Binding.
URL: jesper.sikanda.be/posts/1001-syntax-representations.html (visited on 02/13/2024) 50↑

Copello, Ernesto (2017)
On the Formalisation of the Metatheory of the Lambda Calculus and Languages with Binders. PhD thesis. Universidad ORT Uruguay 51↑

Copello, Ernesto, Nora Szasz, and Álvaro Tasistro (2017)
Formal Metatheory of the Lambda Calculus Using Stoughton's Substitution
In: *Theoretical Computer Science*. Logical and Semantic Frameworks with Applications 685, pp. 65–82.
DOI: 10.1016/j.tcs.2016.08.025 51↑

Copello, Ernesto, Nora Szasz, and Álvaro Tasistro (2018a)
Formalisation in Constructive Type Theory of Barendregt's Variable Convention for Generic Structures with Binders
In: *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2018)*. Vol. 274. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, pp. 11–26.
DOI: 10.4204/EPTCS.274.2 51↑

Copello, Ernesto, Nora Szasz, and Álvaro Tasistro (2018b)
Machine-Checked Proof of the Church-Rosser Theorem for the Lambda Calculus Using the Barendregt Variable Convention in Constructive Type Theory
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 12th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2017) 338, pp. 79–95.
DOI: 10.1016/j.entcs.2018.10.006 51↑

Copello, Ernesto, Nora Szasz, and Álvaro Tasistro (2021)
Formalization of Metatheory of the Lambda Calculus in Constructive Type Theory Using the Barendregt Variable Convention
In: *Mathematical Structures in Computer Science* 31 (3), pp. 341–360.
doi: 10.1017/S0960129521000335 51, 52↑

Copello, Ernesto, Álvaro Tasistro, Nora Szasz, Ana Bove, and Maribel Fernández (2016)
Alpha-Structural Induction and Recursion for the Lambda Calculus in Constructive Type Theory
In: *Electronic Notes in Theoretical Computer Science* 323, pp. 109–124.
doi: 10.1016/j.entcs.2016.06.008 51↑

Curien, Pierre-Louis and Hugo Herbelin (2000)
The Duality of Computation
In: *ACM SIGPLAN Notices* 35 (9), pp. 233–243.
doi: 10.1145/357766.351262 286↑

Day, Brian (1970)
On closed categories of functors
In: *Reports of the Midwest Category Seminar IV*. Vol. 137. Lecture Notes in Computer Science (LNCS). Springer, pp. 1–38.
doi: 10.1007/BFb0060438 126, 160, 287↑

de Bruijn, Nicolaas G. (1972)
Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem
In: *Indagationes Mathematicae*. Vol. 75. 5. Elsevier, pp. 381–392.
doi: 10.1016/1385-7258(72)90034-0 50, 52, 61↑

Despeyroux, Joëlle, Amy Felty, and André Hirschowitz (1995)
Higher-Order Abstract Syntax in Coq
In: *Proceedings of the 2nd International Conference on Typed Lambda Calculi and Applications (TLCA 1995)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 124–138.
doi: 10.1007/BFb0014049 60↑

Despeyroux, Joëlle and André Hirschowitz (1994)
Higher-Order Abstract Syntax with Induction in Coq
In: *Logic Programming and Automated Reasoning*. Vol. 822. Springer, pp. 159–173.
doi: 10.1007/3-540-58216-9_36 59↑

Despeyroux, Joëlle, Frank Pfenning, and Carsten Schürmann (1997)
Primitive Recursion for Higher-Order Abstract Syntax
In: *Typed Lambda Calculi and Applications*, pp. 147–163.
doi: 10.1007/3-540-62688-3_34 60↑

Domínguez, Jesús and Maribel Fernández (2019)
Nominal Syntax with Atom Substitutions: Matching, Unification, Rewriting
In: *Proceedings of the 22nd International Symposium on the Fundamentals of Computation Theory (FCT 2019)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 64–79.
doi: 10.1007/978-3-030-25027-0_5 51↑

Dunn, Lawrence, Val Tannen, and Steve Zdancewic (2023a)
Syntax Monads for the Working Formal Metatheorist.
doi: 10.48550/arXiv.2312.08897.
arxiv: 2312.08897[] 62↑

Dunn, Lawrence, Val Tannen, and Steve Zdancewic (2023b)
Tealeaves: Structured Monads for Generic First-Order Abstract Syntax Infrastructure
In: *Proceedings of the 14th International Conference on Interactive Theorem Proving (ITP 2023)* 268, 14:1–14:20.
doi: 10.4230/LIPICS.ITP.2023.14 62↑

Dybjer, Peter (1994)
Inductive Families
In: *Formal Aspects of Computing* 6 (4), pp. 440–465.
DOI: 10.1007/BF01211308 28 ↑

Dybjer, Peter and Anton Setzer (1999)
A Finite Axiomatization of Inductive-Recursive Definitions
In: *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA 1999)*. Springer, pp. 129–146.
DOI: 10.1007/3-540-48959-2_11 267 ↑

Dyckhoff, Roy (2015)
Cut Elimination, Substitution and Normalisation
In: *Dag Prawitz on Proofs and Meaning*. Springer, pp. 163–187.
DOI: 10.1007/978-3-319-11041-7_7 167 ↑

Eilenberg, Samuel and G. Max Kelly (1966)
Closed Categories
In: *Proceedings of the Conference on Categorical Algebra*. Springer, pp. 421–562.
DOI: 10.1007/978-3-642-99902-4_22 95, 97, 100, 103, 106, 107, 325 ↑

Felleisen, Matthias and Daniel P. Friedman (1987)
Control operators, the SECD-machine, and the λ-calculus
In: *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986*. North-Holland, pp. 193–222.
URL: legacy.cs.indiana.edu/ftp/techreports/TR197.pdf 288 ↑

Felty, Amy and Alberto Momigliano (2012)
Hybrid: A Definitional Two-Level Approach to Reasoning with Higher-Order Abstract Syntax
In: *Journal of Automated Reasoning* 48 (1), pp. 43–105.
DOI: 10.1007/s10817-010-9194-x 61 ↑

Fernández, Maribel and Albert Rubio (2012)
Nominal Completion for Rewrite Systems with Binders
In: *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 201–213.
DOI: 10.1007/978-3-642-31585-5_21 51 ↑

Fiore, Marcelo (2002)
Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus
In: *Proceedings of the 4th International Conference on Principles and Practice of Declarative Programming (PPDP 2002)*. ACM Press, pp. 26–37.
DOI: 10.1145/571157.571161 35, 58 ↑

Fiore, Marcelo (2005)
Mathematical Models of Computational and Combinatorial Structures
In: *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2005)*. Vol. 3441. Springer, pp. 25–46.
DOI: 10.1007/978-3-540-31982-5_2 167 ↑

Fiore, Marcelo (2007)
Towards a Mathematical Theory of Substitution. Invited talk at Category Theory 2007.
URL: cl.cam.ac.uk/~mpf23/talks/CT2007.pdf 167 ↑

Fiore, Marcelo (2008)
Second-Order and Dependently-Sorted Abstract Syntax
In: *Proceedings of the 23rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2008)*, pp. 57–68.
DOI: 10.1109/LICS.2008.38 27, 35, 42–45, 58, 204, 207, 215–218, 232, 233, 284 ↑

Fiore, Marcelo (2012)
Discrete Generalised Polynomial Functors
In: *39th International Colloquium on Automata, Languages and Programming (ICALP 2012)*. Vol. 7392. Lecture Notes in Computer Science. Springer, pp. 214–226.
DOI: 10.1007/978-3-642-31585-5_22 261↑

Fiore, Marcelo (2013)
An Equational Metalogic for Monadic Equational Systems
In: *Theory and Applications of Categories* 27 (18), pp. 464–492.
URL: www.tac.mta.ca/tac/volumes/27/18/27-18.pdf 48, 84, 91, 237↑

Fiore, Marcelo (2017)
New Dimensions in Formal Systems and Computational Models. Research proposal 58↑

Fiore, Marcelo (2019)
Abstract Syntax and Variable Binding. Talk given on reception of the Test of Time Award at LICS 2019 287↑

Fiore, Marcelo (2022)
Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus
In: *Mathematical Structures in Computer Science* 32 (8), pp. 1028–1065.
DOI: 10.1017/S0960129522000263 286↑

Fiore, Marcelo, Nicola Gambino, Martin Hyland, and Glynn Winskel (2008)
The cartesian Closed Bicategory of Generalised Species of Structures
In: *Journal of the London Mathematical Society* 77 (1), pp. 203–220.
DOI: 10.1112/jlms/jdm096 167↑

Fiore, Marcelo and Makoto Hamana (2013)
Multiversal Polymorphic Algebraic Theories: Syntax, Semantics, Translations, and Equational Logic
In: *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*. IEEE Computer Society, pp. 520–529.
DOI: 10.1109/LICS.2013.59 58, 286↑

Fiore, Marcelo and Chung-Kil Hur (2007)
Equational Systems and Free Constructions
In: *Proceedings of the 34th International Colloquium on Automata, Languages, and Programming (ICALP 2007)*. Vol. 4596. Lecture Notes in Computer Science (LNCS). Springer, pp. 607–618.
DOI: 10.1007/978-3-540-73420-8_53 58, 237↑

Fiore, Marcelo and Chung-Kil Hur (2008)
Term equational systems and logics
In: *Electronic Notes in Theoretical Computer Science* 218, pp. 171–192.
DOI: 10.1016/j.entcs.2008.10.011 48, 58, 91, 237↑

Fiore, Marcelo and Chung-Kil Hur (2009)
On the Construction of Free Algebras for Equational Systems.
DOI: 10.1016/j.tcs.2008.12.052 58, 231, 237↑

Fiore, Marcelo and Chung-Kil Hur (2010)
Second-Order Equational Logic
In: *Proceedings of the 24th International Workshop on Computer Science Logic (CSL 2010)*, pp. 320–335.
DOI: 10.1007/978-3-642-15205-4_26 42, 48, 58, 91, 213, 237, 238, 262↑

Fiore, Marcelo and Ola Mahmoud (2010)
Second-Order Algebraic Theories
In: *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*. Vol. 6281. Lecture Notes in Computer Science (LNCS). Springer, pp. 368–380.
DOI: 10.1007/978-3-642-15155-2_33 58, 287↑

Fiore, Marcelo and Ola Mahmoud (2014)
Functorial Semantics of Second-Order Algebraic Theories.

ARXIV: 1401.4697[math.CT].
URL: https://arxiv.org/abs/1401.4697 58 ↑

FIORE, Marcelo, Gordon PLOTKIN, and Daniele TURI (1999)
Abstract Syntax and Variable Binding
In: *Proceedings of the 14th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 1999)*, pp. 193–202.
DOI: 10.1109/LICS.1999.782615 21, 27, 35, 39, 54, 57, 58, 96, 110, 167, 217, 232, 261, 268, 284, 287 ↑

FIORE, Marcelo and Sanjiv RANCHOD (2024)
A finite algebraic presentation of Lawvere theories in the object-classifier topos.
ARXIV: 2408.08980[math.CT].
URL: https://arxiv.org/abs/2408.08980 39, 58, 285 ↑

FIORE, Marcelo and Philip SAVILLE (2017)
List Objects with Algebraic Structure
In: *Proceedings of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*. Vol. 84. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 16:1–16:18.
DOI: 10.4230/LIPIcs.FSCD.2017.16 41, 42, 117, 217 ↑

FIORE, Marcelo and Philip SAVILLE (2018)
Skew monoidal structures on categories of algebras. Talk given at Category Theory 2018.
URL: https://www.mat.uc.pt/~ct2018/slides/P_Saville.pdf 96, 117, 121, 123 ↑

FIORE, Marcelo and Philip SAVILLE (2021)
Skew monoidal categories of algebras. Private communication 118, 123 ↑

FIORE, Marcelo and Dmitrij SZAMOZVANCEV (2022)
Formal Metatheory of Second-Order Abstract Syntax
In: *Proceedings of the ACM on Programming Languages* 6 (POPL), 53:1–53:29.
DOI: 10.1145/3498715 3, 124, 245 ↑

FIORE, Marcelo and Daniele TURI (2001)
Semantics of Name and Value Passing
In: *Proceedings of the 16th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2001)*. IEEE Computer Society, pp. 93–104.
DOI: 10.1109/LICS.2001.932486 170 ↑

FORSTER, Yannick and Kathrin STARK (2020)
Coq à La Carte: A Practical Approach to Modular Syntax with Binders
In: *Proceedings of the 9th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2020)*. ACM Press, pp. 186–200.
DOI: 10.1145/3372885.3373817 53 ↑

FUJII, Soichiro (2016)
A 2-Categorical Study of Graded and Indexed Monads. MA thesis. Graduate School of the University of Tokyo.
ARXIV: 1904.08083[math.CT] 100 ↑

GABBAY, Murdoch J. and Aad MATHIJSSEN (2009)
Nominal (Universal) Algebra: Equational Logic with Names and Binding
In: *Journal of Logic and Computation* 19 (6), pp. 1455–1508.
DOI: 10.1093/logcom/exp033 51 ↑

GABBAY, Murdoch J. and Andrew M. PITTS (1999)
A New Approach to Abstract Syntax Involving Binders
In: *Proceedings of the 14th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 1999)*. IEEE Computer Society, pp. 214–224.
DOI: 10.1109/LICS.1999.782617 51 ↑

Gabbay, Murdoch J. and Andrew M. Pitts (2002)
A New Approach to Abstract Syntax with Variable Binding
In: *Formal Aspects of Computing* 13 (3-5), pp. 341–363.
doi: 10.1007/s001650200016 51↑

Gacek, Andrew (2008)
The Abella Interactive Theorem Prover (System Description)
In: *Automated Reasoning*. Springer, pp. 154–161.
doi: 10.1007/978-3-540-71070-7_13 61↑

Gacek, Andrew, Dale Miller, and Gopalan Nadathur (2011)
Nominal Abstraction
In: *Information and Computation* 209 (1), pp. 48–73.
doi: 10.1016/j.ic.2010.09.004 61↑

Garner, Richard and Michael Shulman (2016)
Enriched categories as a free cocompletion
In: *Advances in Mathematics* 289, pp. 1–94.
doi: doi.org/10.1016/j.aim.2015.11.012 86, 87↑

Gheri, Lorenzo (2019)
A General Theory of Syntax with Bindings. PhD thesis. Middlesex University London.
url: repository.mdx.ac.uk/item/8859v 51↑

Goguen, Joseph A., James W. Thatcher, and E. G. Wagner (1976)
An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types
In: *IBM Research Report* 6487 217↑

Gordon, Andrew D. (1994)
A Mechanisation of Name-Carrying Syntax up to Alpha-Conversion
In: *Higher Order Logic Theorem Proving and Its Applications*. Springer, pp. 413–425.
doi: 10.1007/3-540-57826-9_152 61↑

Greenberg, Michael, Benjamin C. Pierce, and Stephanie Weirich (2010)
Contracts Made Manifest
In: *Proceedings of the 37th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2010)*. ACM Press, pp. 353–364.
doi: 10.1145/1706299.1706341 61↑

Griffin, Timothy G. (1989)
A formulae-as-type notion of control
In: *Proceedings of the 17th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 1989)*. POPL '90. ACM Press, pp. 47–58.
doi: 10.1145/96709.96714 288↑

Hamana, Makoto (2004)
Free Σ-monoids: A higher-order syntax with metavariables
In: *Proceedings of the 2nd Asian Symposium on Programming Languages and Systems (APLAS 2024)*, pp. 348–363.
url: www.cs.gunma-u.ac.jp/~hamana/Papers/free.pdf 41, 43, 58, 215, 232↑

Hamana, Makoto (2011)
Polymorphic Abstract Syntax via Grothendieck Construction
In: *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2011)*. Vol. 6604. Springer, pp. 381–395.
doi: 10.1007/978-3-642-19805-2_26 58, 286↑

Harper, Robert, Furio Honsell, and Gordon Plotkin (1993)
A Framework for Defining Logics
In: *Journal of the ACM* 40 (1), pp. 143–184.
doi: 10.1145/138027.138060 60↑

Hermida, Claudio (2000)
Representable Multicategories
In: *Advances in Mathematics* 151 (2), pp. 164–225.
DOI: 10.1006/aima.1999.1877 126, 288↑

Hermida, Claudio and Bart Jacobs (1995)
An algebraic view of structural induction
In: *Proceedings of the 8th International Workshop on Computer Science Logic (CSL 1994)*. Springer, pp. 412–426.
DOI: 10.1007/BFb0022272 71↑

Hirschowitz, André, Tom Hirschowitz, and Ambroise Lafont (2020)
Modules over Monads and Operational Semantics
In: *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*. Vol. 167. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 12:1–12:23.
DOI: 10.4230/LIPICS.FSCD.2020.12 55, 56↑

Hirschowitz, André, Tom Hirschowitz, Ambroise Lafont, and Marco Maggesi (2022)
Variable binding and substitution for (nameless) dummies
In: *Proceedings of the 25th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2022)*.
URL: hal.science/hal-03547002 53↑

Hirschowitz, André and Marco Maggesi (2007)
Modules over Monads and Linearity
In: *Logic, Language, Information and Computation*. Lecture Notes in Computer Science (LNCS). Springer, pp. 218–237.
DOI: 10.1007/978-3-540-73445-1_16 55↑

Hirschowitz, André and Marco Maggesi (2012)
Initial Semantics for Strengthened Signatures
In: *Proceedings of the 8th Workshop on Fixed Points in Computer Science (FICS 2012)*. Vol. 77. EPTCS, pp. 31–38.
DOI: 10.4204/EPTCS.77.5 55↑

Hofmann, Martin (1999)
Semantical Analysis of Higher-Order Abstract Syntax
In: *Proceedings of the 14th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 1999)*, pp. 204–213.
DOI: 10.1109/LICS.1999.782616 60↑

Hu, Jason Z. S. and Jacques Carette (2021)
Formalizing Category Theory in Agda
In: *Proceedings of the 10th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2021)*. ACM Press, pp. 327–342.
DOI: 10.1145/3437992.3439922 247↑

Huet, Gérard (1989)
The Constructive Engine
In: *A Perspective in Theoretical Computer Science*. World Scientific, pp. 38–69.
DOI: 10.1142/9789814368452_0004 61↑

Huet, Gérard (1994)
Residual Theory in λ-Calculus: A Formal Development
In: *Journal of Functional Programming* 4 (3), pp. 371–394.
DOI: 10.1017/S0956796800001106 52↑

Huffman, Brian and Christian Urban (2010)
A New Foundation for Nominal Isabelle
In: *Proceedings of the 1st International Conference on Interactive Theorem Proving (ITP 2010)*. Lecture

Notes in Computer Science (LNCS). Springer, pp. 35–50.
DOI: 10.1007/978-3-642-14052-5_5 51↑

Hur, Chung-Kil (2010)
Categorical equational systems: algebraic models and equational reasoning. PhD thesis. University of Cambridge.
URL: ropas.snu.ac.kr/~gil.hur/publications/thesis.pdf 58, 237↑

Im, Geun Bin and G. Max Kelly (1986)
A Universal Property of the Convolution Monoidal Structure
In: *Journal of Pure and Applied Algebra* 43 (1), pp. 75–88.
DOI: 10.1016/0022-4049(86)90005-8 160, 161↑

Jacobson, Nathan (2012)
Basic Algebra II. Courier Corporation. ISBN: 978-0-486-13521-2 115↑

Janelidze, George and G. Max Kelly (2001)
A note on actions of a monoidal category
In: *Theory and Applications of Categories* 9 (4), pp. 61–91.
URL: www.tac.mta.ca/tac/volumes/9/n4/n4.pdf 81, 100↑

Jang, Junyoung, Samuel Gélineau, Stefan Monnier, and Brigitte Pientka (2022)
Mœbius: Metaprogramming Using Contextual Types: The Stage Where System F Can Pattern Match on Itself
In: *Proceedings of the ACM on Programming Languages* 6 (POPL), pp. 1–27.
DOI: 10.1145/3498700 61↑

Joyal, André (1981)
Une théorie combinatoire des séries formelles
In: *Advances in Mathematics* 42 (1), pp. 1–82.
DOI: 10.1016/0001-8708(81)90052-9 287↑

Kaiser, Jonas, Steven Schäfer, and Kathrin Stark (2017)
Autosubst 2: Towards Reasoning with Multi-Sorted de Bruijn Terms and Vector Substitutions
In: *Proceedings of the 12th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2017)*. ACM Press, pp. 10–14.
DOI: 10.1145/3130261.3130263 53↑

Kaiser, Jonas, Steven Schäfer, and Kathrin Stark (2018)
Binder Aware Recursion over Well-Scoped de Bruijn Syntax
In: *Proceedings of the 7th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2018)*. ACM Press, pp. 293–306.
DOI: 10.1145/3167098 53↑

Kavvos, G. A. (2017)
Dual-Context Calculi for Modal Logic
In: *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*. IEEE Computer Society, pp. 1–12.
DOI: 10.1109/LICS.2017.8005089 286↑

Kelly, G. Max and John Power (1993)
Adjunctions Whose Counits Are Coequalizers, and Presentations of Finitary Enriched Monads
In: *Journal of Pure and Applied Algebra* 89 (1), pp. 163–179.
DOI: 10.1016/0022-4049(93)90092-8 84↑

Kelly, G. Max and Ross Street (1974)
Review of the Elements of 2-Categories
In: *Category Seminar*. Springer, pp. 75–103.
DOI: 10.1007/BFb0063101 104↑

Kerkhoff, Sebastian, Reinhard Pöschel, and Friedrich Martin Schneider (2014)
A Short Introduction to Clones
In: *Electronic Notes in Theoretical Computer Science* 303, pp. 107–120.
doi: 10.1016/j.entcs.2014.02.006 167 ↑

Keuchel, Steven and Johan T. Jeuring (2012)
Generic Conversions of Abstract Syntax Representations
In: *Proceedings of the 8th ACM SIGPLAN Workshop on Generic Programming*. ACM Press, pp. 57–68.
doi: 10.1145/2364394.2364403 62 ↑

Keuchel, Steven, Stephanie Weirich, and Tom Schrijvers (2016)
Needle & Knot: Binder Boilerplate Tied Up
In: *Proceedings of the 25th European Symposium on Programming (ESOP 2016)*. Vol. 9632. Springer, pp. 419–445.
doi: 10.1007/978-3-662-49498-1_17 53 ↑

Kmett, Edward (2015)
Bound.
url: schoolofhaskell.com/user/edwardk/bound (visited on 02/13/2024) 50 ↑

Kock, Anders (1970a)
Monads on symmetric monoidal closed categories
In: *Archiv der Mathematik* 21, pp. 1–10.
doi: 10.1007/BF01220868 103 ↑

Kock, Anders (1970b)
On double dualization monads
In: *Mathematica Scandinavica* 27 (2), pp. 151–165.
url: jstor.org/stable/24489892 84 ↑

Kock, Anders (1971a)
Bilinearity and cartesian closed monads
In: *Mathematica Scandinavica* 29 (2), pp. 161–174.
url: jstor.org/stable/24491025 117 ↑

Kock, Anders (1971b)
Closed categories generated by commutative monads
In: *Journal of the Australian Mathematical Society* 12 (4), pp. 405–424.
doi: 10.1017/S1446788700010272 103 ↑

Kock, Anders (1972)
Strong functors and monoidal monads
In: *Archiv der Mathematik* 23, pp. 113–120.
doi: 10.1007/BF01304852 103 ↑

Kock, Joachim, André Joyal, Michael Batanin, and Jean-François Mascari (2010)
Polynomial Functors and Opetopes
In: *Advances in Mathematics* 224 (6), pp. 2690–2737.
doi: 10.1016/j.aim.2010.02.012 167 ↑

Lack, Stephen and Ross Street (2012a)
A Skew-Duoidal Eckmann-Hilton Argument and Quantum Categories
In: *Applied Categorical Structures* 22.
doi: 10.1007/s10485-013-9356-1 95 ↑

Lack, Stephen and Ross Street (2012b)
Skew monoidales, skew warpings and quantum categories
In: *Theory and Applications of Categories* 26 (15), pp. 385–402.
url: tac.mta.ca/tac/volumes/26/15/26-15.pdf 95, 112, 137 ↑

Lack, Stephen and Ross Street (2014a)
On monads and warpings

In: *Cahiers de Topologie et Géométrie Différentielle Catégoriques* LV (4), pp. 244–266.
URL: cahierstgdc.com/wp-content/uploads/2017/05/LackStreet_55-4.pdf 95, 100, 288 ↑

Lack, Stephen and Ross Street (2014[b])
Triangulations, orientals, and skew monoidal categories
In: *Advances in Mathematics* 258, pp. 351–396. ISSN: 0001-8708.
DOI: 10.1016/j.aim.2014.03.003 95 ↑

Lack, Stephen and Ross Street (2015)
Skew-monoidal reflection and lifting theorems
In: *Theory and Applications of Categories* 30 (28), pp. 985–1000.
URL: www.tac.mta.ca/tac/volumes/30/28/30-28.pdf 95, 137 ↑

Lambek, Joachim (1986)
Cartesian closed categories and typed λ-calculi
In: *Combinators and Functional Programming Languages*. Springer, pp. 136–175.
DOI: 10.1007/3-540-17184-3_44 270 ↑

Lamiaux, Thomas and Benedikt Ahrens (2025)
A Unified Framework for Initial Semantics.
ARXIV: 2502.10811[cs.LO] 50, 55, 56 ↑

Laplaza, Miguel L. (1977)
Embedding of Closed Categories Into Monoidal Closed Categories
In: *Transactions of the American Mathematical Society* 233, pp. 85–91.
DOI: 10.2307/1997823. JSTOR: 1997823 97 ↑

Lawvere, F. William (1963)
Functorial Semantics of Algebraic Theories. PhD thesis. Columbia University.
DOI: 10.1007/BFb0077116 58 ↑

Lee, Gyesik, Bruno C. D. S. Oliveira, Sungkeun Cho, and Kwangkeun Yi (2012)
GMeta: A Generic Formal Metatheory Framework for First-Order Representations
In: *Proceedings of the 21st European Symposium on Programming (ESOP 2012)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 436–455.
DOI: 10.1007/978-3-642-28869-2_22 62 ↑

Lehner, Marina Christina (2014)
"All Concepts are Kan Extensions": Kan Extensions as the Most Universal of the Universal Constructions. Bachelor's Thesis. Harvard College.
URL: legacy-www.math.harvard.edu/theses/senior/lehner/lehner.pdf 156 ↑

Leinster, Tom (1999)
fc-Multicategories.
ARXIV: math/9903004[] 289 ↑

Leinster, Tom (2004)
Higher Operads, Higher Categories. London Mathematical Society Lecture Note Series. Cambridge University Press.
DOI: 10.1017/CBO9780511525896 126, 167, 288 ↑

Leroy, Xavier (2007)
A Locally Nameless Solution to the POPLmark Challenge. Technical report. INRIA, p. 54
61 ↑

Levy, Paul Blain (1999)
Call-by-Push-Value: A Subsuming Paradigm
In: *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA 1999)*. Springer, pp. 228–242.
DOI: 10.1007/3-540-48959-2_17 286 ↑

Loregian, Fosco (2019)
A Fubini rule for ∞-coends

In: *arXiv e-prints*, arXiv:1902.06086.
ARXIV: 1902.06086[math.CT] 155↑

LOREGIAN, Fosco (2021)
(Co)end Calculus. London Mathematical Society Lecture Note Series. Cambridge University Press.
DOI: 10.1017/9781108778657 151, 153, 155, 160↑

MAC LANE, Saunders (1971)
Categories for the Working Mathematician.
Graduate Text in Mathematics. Springer. ISBN: 978-0-387-98403-2.
DOI: 10.1007/978-1-4757-4721-8 75, 95, 156↑

MAC LANE, Saunders and Garrett BIRKHOFF (1967)
Algebra. The Macmillan Company 116↑

MACLANE, Saunders and Ieke MOERDIJK (2012)
Sheaves in geometry and logic: A first introduction to topos theory. Springer 151↑

MAHMOUD, Ola (2011)
Second-Order Algebraic Theories. PhD thesis. University of Cambridge.
URL: www.repository.cam.ac.uk/bitstream/handle/1810/241035/Thesis.pdf 58↑

MARMOLEJO, Francisco and Richard J. WOOD (2010)
Monads as Extension Systems – No Iteration is Necessary
In: *Theory and Applications of Categories* 24 (4), pp. 84–113.
URL: tac.mta.ca/tac/volumes/24/4/24-04.pdf 84↑

MATACHE, Cristina (2017)
Formalisation of the λμT-calculus in Isabelle/HOL. Undergraduate dissertation. MA thesis. University of Cambridge.
URL: homepages.inf.ed.ac.uk/cmatache/documents/diss.pdf 52↑

MATTHES, Ralph (2011)
Map Fusion for Nested Datatypes in Intensional Type Theory
In: *Science of Computer Programming*. Special Issue on the Mathematics of Program Construction (MPC 2008) 76 (3), pp. 204–224.
DOI: 10.1016/j.scico.2010.05.008 57↑

MATTHES, Ralph and Tarmo UUSTALU (2004)
Substitution in Non-Wellfounded Syntax with Variable Binding
In: 327 (1), pp. 155–174.
DOI: 10.1016/j.tcs.2004.07.025 56, 57, 217↑

MATTHES, Ralph, Kobe WULLAERT, and Benedikt AHRENS (2023)
Substitution for Non-Wellfounded Syntax with Binders through Monoidal Categories.
ARXIV: 2308.05485[cs.PL] 57↑

MCBRIDE, Conor (2005)
Type-preserving renaming and substitution. Unpublished note.
URL: http://strictlypositive.org/ren-sub.pdf 32, 39, 167, 247, 250, 253, 257↑

MCBRIDE, Conor and James MCKINNA (2004)
Functional Pearl: I Am Not a Number – I Am a Free Variable
In: *Proceedings of the 2004 ACM SIGPLAN Workshop on Haskell*. Haskell '04. ACM Press, pp. 1–9.
DOI: 10.1145/1017472.1017477 61↑

MCBRIDE, Conor and Ross PATERSON (2008)
Applicative Programming with Effects
In: *Journal of Functional Programming* 18 (01).
DOI: 10.1017/S0956796807006326 62↑

MCDERMOTT, Dylan and Tarmo UUSTALU (2022)
What Makes a Strong Monad?
In: *Proceedings of the 9th Mathematically Structured Functional Programming (MSFP 2022)*. Vol. 360.

Electronic Proceedings in Theoretical Computer Science, pp. 113–133.
DOI: 10.4204/EPTCS.360.6 81, 82, 103 ↑

McKinna, James and Robert Pollack (1993)
Pure Type Systems Formalized
In: *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications (TLCA 1993)*. Springer, pp. 289–305.
DOI: 10.1007/BFb0037113 61 ↑

Mendler, Nax Paul (1991)
Inductive Types and Type Constraints in the Second-Order Lambda Calculus
In: *Annals of Pure and Applied Logic* 51 (1), pp. 159–172.
DOI: 10.1016/0168-0072(91)90069-X 57 ↑

Miculan, Marino and Ivan Scagnetto (2003)
A framework for typed HOAS and semantics
In: *Proceedings of the 5th International Conference on Principles and Practice of Declarative Programming (PPDP 2003)*. PPDP '03. ACM Press, pp. 184–194.
DOI: 10.1145/888251.888269 58 ↑

Miller, Dale (2000)
Abstract Syntax for Variable Binders: An Overview
In: *First International Conference on Computational Logic (CL 2000)*. Vol. 1861. Springer, pp. 239–253.
DOI: 10.1007/3-540-44957-4_16 61 ↑

Moggi, Eugenio (1991)
Notions of computation and monads
In: *Information and Computation* 93 (1), pp. 55–92.
DOI: 10.1016/0890-5401(91)90052-4 53 ↑

Murawski, Andrzej S. and Nikos Tzevelekos (2016)
Nominal Game Semantics
In: *Foundations and Trends® in Programming Languages* 2 (4), pp. 191–269.
DOI: 10.1561/2500000017 51 ↑

Nanevski, Aleksandar, Frank Pfenning, and Brigitte Pientka (2008)
Contextual Modal Type Theory
In: *ACM Transactions on Computational Logic* 9 (3), pp. 1–49.
DOI: 10.1145/1352582.1352591 61, 288 ↑

Pagano, Miguel and José E. Solsona (2023)
Nominal Sets in Agda – A Fresh and Immature Mechanization
In: *Electronic Proceedings in Theoretical Computer Science* 376, pp. 67–80.
DOI: 10.4204/EPTCS.376.7.
arXiv: 2303.13252[cs] 51, 52 ↑

Paranhos, Fabrízio S. (2022)
Uma formalização da teoria nominal em Coq. MA thesis. Universidade Federal de Goiás (UFG), Brasil.
URL: repositorio.bc.ufg.br/tede/handle/tede/12314 51 ↑

Paranhos, Fabrízio S. and Daniel Ventura (2022)
Towards a Formalization of Nominal Sets in Coq. Presented at the 8th International Workshop on Coq for Programming Languages (CoqPL 2022).
URL: inf.ufg.br/~daniel/papers/coqpl22-final52.pdf 51 ↑

Paulson, Lawrence (2015)
A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle
In: *Journal of Automated Reasoning* 55, pp. 1–37.
DOI: 10.1007/s10817-015-9322-8 51 ↑

Paulson, Lawrence (2023)
Large-Scale Formal Proof for the Working Mathematician – Lessons Learnt from the

ALEXANDRIA Project.
ARXIV: 2305.14407[math] 17↑

PFENNING, Frank and Rowan DAVIES (2001)
A judgmental reconstruction of modal logic
In: *Mathematical Structures in Computer Science* 11 (4), pp. 511–540.
DOI: 10.1017/S0960129501003322 288↑

PFENNING, Frank and Conal ELLIOTT (1988)
Higher-Order Abstract Syntax
In: *Proceedings of the 1st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 1988)*. ACM Press, pp. 199–208.
DOI: 10.1145/53990.54010 59↑

PFENNING, Frank and Carsten SCHÜRMANN (1999)
System Description: Twelf — A Meta-Logical Framework for Deductive Systems
In: *Proceedings of the 16th International Conference on Automated Deduction (CADE 1999)*. Springer, pp. 202–206.
DOI: 10.1007/3-540-48660-7_14 60↑

PICKERING, Matthew, Gergő ÉRDI, Simon PEYTON JONES, and Richard EISENBERG (2016)
Pattern Synonyms
In: *Proceedings of the 9th ACM SIGPLAN International Symposium on Haskell (Haskell 2016)*. ACM Press, pp. 80–91.
DOI: 10.1145/2976002.2976013 263↑

PIENTKA, Brigitte and Jana DUNFIELD (2008)
Programming with Proofs and Explicit Contexts
In: *Proceedings of the 10th International Conference on Principles and Practice of Declarative Programming (PPDP 2008)*. ACM, pp. 163–173.
DOI: 10.1145/1389449.1389469 61↑

PIENTKA, Brigitte and Ulrich SCHÖPP (2020)
Semantical Analysis of Contextual Types
In: *Proceedings of the 23rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2020)*. Springer, pp. 502–521.
DOI: 10.1007/978-3-030-45231-5_26 61↑

PIENTKA, Brigitte, David THIBODEAU, Andreas ABEL, Francisco FERREIRA, and Rebecca ZUCCHINI (2021)
A Type Theory for Defining Logics and Proofs
In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*. LICS '19. IEEE Computer Society, pp. 1–13.
DOI: 10.5555/3470152.3470181 61↑

PITTS, Andrew M. (2003)
Nominal Logic, a First Order Theory of Names and Binding
In: *Information and Computation* 186 (2), pp. 165–193.
DOI: 10.1016/S0890-5401(03)00138-X 51↑

PITTS, Andrew M. (2006)
Alpha-structural recursion and induction
In: *Journal of the ACM* 53 (3), pp. 459–506.
DOI: 10.1145/1147954.1147961 51↑

PITTS, Andrew M. (2011)
Structural Recursion with Locally Scoped Names
In: *Journal of Functional Programming* 21 (3), pp. 235–286.
DOI: 10.1017/S0956796811000116 51↑

Pitts, Andrew M. (2013)
Nominal Sets: Names and Symmetry in Computer Science. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press 51↑

Pitts, Andrew M. (2014)
An Equivalent Presentation of the Bezem-Coquand-Huber Category of Cubical Sets.
arXiv: 1401.7807[cs.LO] 51↑

Pitts, Andrew M. (2015)
Nominal Presentation of Cubical Sets Models of Type Theory
In: *Proceedings of the 20th International Conference on Types for Proofs and Programs (TYPES 2014)*. Vol. 39. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 202–220.
DOI: 10.4230/LIPIcs.TYPES.2014.202 51↑

Pitts, Andrew M. (2019)
Initial algebra for a strictly positive endofunctor constructed using sized types and quotient types.
URL: www.cl.cam.ac.uk/~amp12/agda/initial-T-algebras/InitialTAlgebra.html (visited on 08/31/2024) 264↑

Pitts, Andrew M. and Murdoch J. Gabbay (2000)
A Metalanguage for Programming with Bound Names Modulo Renaming
In: *Proceedings of the 5th International Conference on the Mathematics of Program Construction (MPC 2000)*. Vol. 1837. Springer, pp. 230–255.
DOI: 10.1007/10722010_15 51↑

Plotkin, Gordon (2020)
A Complete Equational Axiomatisation of Partial Differentiation
In: *Electronic Notes in Theoretical Computer Science* 352, pp. 211–232.
DOI: j.entcs.2020.09.011 274↑

Pollack, Randy, Masahiko Sato, and Wilmer Ricciotti (2012)
A Canonical Locally Named Representation of Binding
In: *Journal of Automated Reasoning* 49 (2), pp. 185–207.
DOI: 10.1007/s10817-011-9229-y 51↑

Polonowski, Emmanuel (2013)
Automatically Generated Infrastructure for De Bruijn Syntaxes
In: *Proceedings of the 4þ International Conference on Interactive Theorem Proving (ITP 2013)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 402–417.
DOI: 10.1007/978-3-642-39634-2_29 53↑

Popescu, Andrei (2023)
Rensets and Renaming-Based Recursion for Syntax with Bindings
In: *Journal of Automated Reasoning* 67 (3), p. 23.
DOI: 10.1007/s10817-023-09672-4 50, 51↑

Popescu, Andrei (2024)
Nominal Recursors as Epi-Recursors
In: *Proceedings of the ACM on Programming Languages* 8 (POPL), 15:425–15:456.
DOI: 10.1145/3632857 52↑

Poswolsky, Adam and Carsten Schürmann (2009)
System Description: Delphin – A Functional Programming Language for Deductive Systems
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 1st International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2008) 228, pp. 113–120.
DOI: 10.1016/j.entcs.2008.12.120 60↑

Pottier, François (2014)
DBLib: Coq library for working with de Bruijn indices.
URL: github.com/coq-community/dblib (visited on 02/13/2024) 53↑
Power, John (2007)
Abstract Syntax: Substitution and Binders
In: *Electronic Notes in Theoretical Computer Science* 173, pp. 3–16.
DOI: 10.1016/j.entcs.2007.02.024 58, 167↑
Power, John and Miki Tanaka (2008)
Category Theoretic Semantics for Typed Binding Signatures with Recursion
In: *Fundamenta Informaticae* 84 (2), pp. 221–240.
URL: researchportal.bath.ac.uk/en/publications/category-theoretic-semantics-for-typed-binding-signatures-with-re 58↑
Rasmussen, Ole (1995)
The Church–Rosser theorem in Isabelle: a proof porting experiment. Tech. rep. UCAM-CL-TR-364. University of Cambridge, Computer Laboratory.
DOI: 10.48456/tr-364. 52↑
Ratkovic, Kruna Sergt (2012)
Morita theory in enriched context. PhD thesis. Université de Nice Sophia Antipolis.
URL: theses.hal.science/tel-00785301/document 102↑
Reyes, Marie La Palme, Gonzalo Reyes, and Houman Zolfaghari (2004)
Generic figures and their glueings: A constructive approach to functor categories. Polimetrica 151↑
Riehl, Emily (2017)
Category theory in context. Courier Dover Publications 156↑
Rose, Kristoffer (1996)
Explicit Substitution: Tutorial & Survey. Technical report LS-96-3. BRICS Lecture Series. University of Århus.
URL: researchgate.net/publication/228386201_Explicit_substitution_tutorial_survey 53↑
Rossberg, Andreas, Claudio Russo, and Derek Dreyer (2014)
F-ing modules
In: *Journal of Functional Programming* 24 (5), pp. 529–607.
DOI: 10.1017/S0956796814000264 61↑
Sabok, Marcin, Sam Staton, Dario Stein, and Michael Wolman (2021)
Probabilistic Programming Semantics for Name Generation
In: *Proceedings of the ACM on Programming Languages* 5 (POPL), 11:1–11:29.
DOI: 10.1145/3434292 51↑
Sato, Masahiko, Takafumi Sakurai, Yukiyoshi Kameyama, and Atsushi Igarashi (2003)
Calculi of meta-variables
In: *Proceedings of the 17th International Workshop on Computer Science Logic (CSL 2003)*, pp. 484–497.
DOI: 10.1007/978-3-540-45220-1_39 58↑
Schäfer, Steven, Tobias Tebbi, and Gert Smolka (2015)
Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions
In: *Proceedings of the 6th International Conference on Interactive Theorem Proving (ITP 2015)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 359–374.
DOI: 10.1007/978-3-319-22102-1_24 53↑
Sewell, Peter, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, et al. (2010)
Ott: Effective tool support for the working semanticist
In: *Journal of Functional Programming* 20 (1), pp. 71–122.
DOI: 10.1145/1291220.1291155 61↑

Shankar, Natarajan (1988)
A Mechanical Proof of the Church–Rosser Theorem
In: *Journal of the ACM* 35 (3), pp. 475–522.
DOI: 10.1145/44483.44484 52↑

Shinwell, Mark R. (2006)
Fresh O'Caml: Nominal Abstract Syntax for the Masses
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the ACM-SIGPLAN Workshop on ML (ML 2005) 148 (2), pp. 53–77.
DOI: 10.1016/j.entcs.2005.11.040 51↑

Shinwell, Mark R., Andrew M. Pitts, and Murdoch J. Gabbay (2003)
FreshML: Programming with Binders Made Simple
In: *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003)*. ACM Press, pp. 263–274.
DOI: 10.1145/944705.944729 51↑

Stark, Kathrin (2020)
Mechanising Syntax with Binders in Coq. PhD thesis. Saarland University, p. 206.
URL: ps.uni-saarland.de/Publications/documents/Stark_2020_Mechanising.pdf 53↑

Stoughton, Allen (1988)
Substitution Revisited
In: *Theoretical Computer Science* 59 (3), pp. 317–325.
DOI: 10.1016/0304-3975(88)90149-1 51↑

Street, Ross (2013)
Skew-closed categories
In: *Journal of Pure and Applied Algebra* 217 (6), pp. 973–988.
DOI: 10.1016/j.jpaa.2012.09.020 95, 97, 100, 103, 287↑

Szlachányi, Kornél (2012)
Skew-monoidal categories and bialgebroids
In: *Advances in Mathematics* 231 (3-4), pp. 1694–1730.
DOI: 10.1016/j.aim.2012.06.027 95↑

Tanaka, Miki (2000)
Abstract Syntax and Variable Binding for Linear Binders
In: *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS 2000)*. Vol. 1893. Lecture Notes in Computer Science (LNCS). Springer, pp. 670–679.
DOI: 10.1007/3-540-44612-5_62 58, 285↑

Tanaka, Miki (2005)
Pseudo-Distributive Laws and a Unified Framework for Variable Binding. PhD thesis. Laboratory for Foundations of Computer Science, University of Edinburgh.
URL: www.lfcs.inf.ed.ac.uk/reports/04/ECS-LFCS-04-438/index.html 58, 66↑

Tanaka, Miki and John Power (2006)
Pseudo-Distributive Laws and Axiomatics for Variable Binding
In: *Higher-Order and Symbolic Computation* 19 (2), pp. 305–337.
DOI: 10.1007/s10990-006-8750-x 58↑

Tao, Terence Chi-Shen (2025)
Machine-Assisted Proof
In: *Notices of the American Mathematical Society* 72 (01), p. 1.
DOI: 10.1090/noti3041 17↑

Tasistro, Álvaro, Ernesto Copello, and Nora Szasz (2015)
Formalisation in Constructive Type Theory of Stoughton's Substitution for the Lambda Calculus
In: *Electronic Notes in Theoretical Computer Science* 312, pp. 215–230.
DOI: 10.1016/j.entcs.2015.04.013 51↑

Taylor, Walter (1993)
Abstract Clone Theory
In: *Algebras and Orders*. Springer, pp. 507–530.
DOI: 10.1007/978-94-017-0697-1_11 53, 84↑

Tiu, Alwen (2009)
On the Role of Names in Reasoning about λ-Tree Syntax Specifications
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 1st International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2008) 228, pp. 135–150.
DOI: 10.1016/j.entcs.2008.12.122 61↑

Tobin-Hochstadt, Sam and Matthias Felleisen (2008)
The Design and Implementation of Typed Scheme
In: *Proceedings of the 35th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2008)*. ACM Press, pp. 395–406.
DOI: 10.1145/1328438.1328486 51↑

Trimble, Todd (2013)
Towards a doctrine of operads.
URL: ncatlab.org/toddtrimble/published/Towards+a+doctrine+of+operads 168↑

Urban, Christian (2008)
Nominal Techniques in Isabelle/HOL
In: *Journal of Automated Reasoning* 40 (4), pp. 327–356.
DOI: 10.1007/s10817-008-9097-2 51↑

Urban, Christian, James Cheney, and Stefan Berghofer (2011)
Mechanizing the metatheory of LF
In: *ACM Transactions on Computational Logic* 12 (2).
DOI: 10.1145/1877714.1877721 51↑

Urban, Christian and Cezary Kaliszyk (2011)
General Bindings and Alpha-Equivalence in Nominal Isabelle
In: *Proceedings of the 20th European Symposium on Programming (ESOP 2011)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 480–500.
DOI: 10.1007/978-3-642-19718-5_25 51↑

Urban, Christian and Michael Norrish (2005)
A Formal Treatment of the Barendregt Variable Convention in Rule Inductions
In: *Proceedings of the 3rd ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding (MERLIN 2005)*. ACM Press, pp. 25–32.
DOI: 10.1145/1088454.1088458 51↑

Urban, Christian, Andrew M. Pitts, and Murdoch J. Gabbay (2003)
Nominal Unification
In: *Proceedings of the 17th International Workshop on Computer Science Logic (CSL 2003)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 513–527.
DOI: 10.1007/978-3-540-45220-1_41 51↑

Urciuoli, Sebastián, Álvaro Tasistro, and Nora Szasz (2020)
Strong Normalization for the Simply-Typed Lambda Calculus in Constructive Type Theory Using Agda
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 15th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2020) 351, pp. 187–203.
DOI: 10.1016/j.entcs.2020.08.010 51↑

Uustalu, Tarmo (2010)
Strong relative monads
In: *10th International Workshop on Coalgebraic Methods in Computer Science (CMCS 2010)*. SEN-1004.
URL: event.cwi.nl/cmcs10/slides/18_cmcs10-slides.pdf 129↑

Uustalu, Tarmo, Niccolò Veltri, and Noam Zeilberger (2020)
Eilenberg–Kelly Reloaded
In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the 36th Mathematical Foundations of Programming Semantics Conference (MFPSC 2020) 352, pp. 233–256.
DOI: 10.1016/j.entcs.2020.09.012 81, 95, 97, 103, 106, 107 ↑

Voevodsky, Vladimir (2014)
The Origins and Motivations of Univalent Foundations
In: *The Institute Letter* Summer 2014, pp. 8–9.
URL: ias.edu/sites/default/files/pdfs/publications/letter-2014-summer.pdf 17 ↑

Vouillon, Jérôme (2011)
A Solution to the PoplMark Challenge Based on de Bruijn Indices
In: *Journal of Automated Reasoning* 49, pp. 327–362.
DOI: 10.1007/s10817-011-9230-5 52 ↑

Wadler, Philip (1990)
Comprehending Monads
In: *Proceedings of the 1990 ACM Conference on LISP and Functional Programming (LFP)*. ACM Press, pp. 61–78.
DOI: 10.1145/91556.91592 53 ↑

Wadler, Philip (1992)
The Essence of Functional Programming
In: *Proceedings of the 19th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 1992)*. ACM Press, pp. 1–14.
DOI: 10.1145/143165.143169 53 ↑

Wan, Xinyi and Qinxiang Cao (2024)
Formalization of Lambda Calculus with Explicit Names as a Nominal Reasoning Framework
In: *Proceedings of the 9th International Symposium on Dependable Software Engineering. Theories, Tools, and Applications (SETTA 2024)*. Lecture Notes in Computer Science (LNCS). Springer, pp. 262–278.
DOI: 10.1007/978-981-99-8664-4_15 52 ↑

Washburn, Geoffrey and Stephanie Weirich (2003)
Boxes go bananas: encoding higher-order abstract syntax with parametric polymorphism
In: *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003)*. ICFP '03. ACM Press, pp. 249–262.
DOI: 10.1145/944705.944728 60 ↑

Weirich, Stephanie and Chris Casinghino (2012)
Generic Programming with Dependent Types
In: *Generic and Indexed Programming: International Spring School, SSGIP 2010, Oxford, UK, March 22-26, 2010, Revised Lectures*. Springer, pp. 217–258.
DOI: 10.1007/978-3-642-32202-0_5 33 ↑

Zsido, Julianna (2010)
Typed Abstract Syntax. PhD thesis. Université Nice Sophia Antipolis.
URL: theses.hal.science/tel-00535944 55, 56, 58, 129 ↑

APPENDIX A

Detailed proofs

We list longer proofs of propositions below.

§ Proof of Proposition 3.3.1 on page 73

Proposition 3.3.1 *Let $C\colon \mathcal{C} \to \mathcal{C}$ be a comonad and $F\colon \mathcal{C} \to \mathcal{C}$ an endofunctor with $\widehat{F}$ a strict lifting to C-**Coalg**$(\mathcal{C})$. Then, an initial F-algebra $(\mathrm{F}, f) \in F$-**alg**$(\mathcal{C})$ lifts to an initial $\widehat{F}$-algebra $(\hat{\mathrm{F}}, \hat{f}) \in \widehat{F}$-**alg**$(C$-**Coalg**$(\mathcal{C}))$ satisfying $U(\hat{\mathrm{F}}) = \mathrm{F}$.*

Proof Assume the following situation:

$$\begin{array}{ccc} C\text{-}\mathbf{Coalg}(\mathcal{C}) & \xrightarrow{\widehat{F}} & C\text{-}\mathbf{Coalg}(\mathcal{C}) \\ {\scriptstyle U}\downarrow & & \downarrow{\scriptstyle U} \\ \mathcal{C} & \xrightarrow[F]{} & \mathcal{C} \end{array}$$

We show that the initial F-algebra $(\mathrm{F}, f\colon F\mathrm{F} \to \mathrm{F})$ induces an initial $\widehat{F}$-algebra $(\hat{\mathrm{F}}, \hat{f}\colon \widehat{F}(\hat{\mathrm{F}}) \to (\hat{\mathrm{F}})) \in C$-**Coalg**$(\mathcal{C})$ such that $U\hat{\mathrm{F}} = \mathrm{F}$ – that is, the carrier of the initial $\widehat{F}$-algebra has the form $\hat{\mathrm{F}} = (\mathrm{F}, c)$ for a C-coalgebra structure $c\colon \mathrm{F} \to C\mathrm{F}$. The four steps are: ① equipping F with this C-coalgebra structure c to obtain an object $(\mathrm{F}, c) \in C$-**Coalg**$(\mathcal{C})$, ② showing that it is an algebra for $\widehat{F}\colon C$-**Coalg**$(\mathcal{C}) \to C$-**Coalg**$(\mathcal{C})$, ③ showing that there is a $\widehat{F}$-algebra homomorphism from (F, c) into any other $\widehat{F}$-algebra, and ④ proving that this homomorphism is unique.

① *C-coalgebra structure* By the dual of Proposition 3.1.5, the lifting of F to C-coalgebras gives rise to a lifting $\widehat{C}\colon F$-**alg** $\to F$-**alg** that has comonad structure. Since the lifting of the initial F-algebra $\widehat{C}(\mathrm{F}, f\colon F\mathrm{F} \to \mathrm{F}) = (C\mathrm{F}, FC\mathrm{F} \xrightarrow{\varphi_{\mathrm{F}}} C\mathrm{F}\mathrm{F} \xrightarrow{Cf} C\mathrm{F})$ is also an F-algebra, by initiality we have a unique F-algebra homomorphism $c\colon (\mathrm{F}, f) \to \widehat{C}(\mathrm{F}, f)$ satisfying

$$\begin{array}{ccccc} FF & & \xrightarrow{\quad f \quad} & & F \\ {\scriptstyle Fc}\downarrow & & & & \downarrow{\scriptstyle c} \\ FCF & \xrightarrow[\varphi_F]{} & CFF & \xrightarrow[Cf]{} & CF \end{array} \tag{†}$$

which will give the C-coalgebra structure $c\colon F \to CF$ on F. This homomorphism diagram shows both that c is an F-algebra homomorphism from F to CF, and that f is a C-coalgebra homomorphism from FF to F.

② *$\widehat{F}$-algebra structure* We prove that we have an $\widehat{F}$-algebra structure map on the object $(F, c) \in C\text{-}\mathbf{Coalg}(\mathcal{C})$. Since $\widehat{F}$ is a lifting of F to C-coalgebras, an $\widehat{F}$-algebra structure map is $\widehat{F}(F, c) = (FF, \hat{c}) \to (F, c)$, given by $f\colon FF \to F$ which is a C-algebra homomorphism by (†). Thus, $\big((F, F \xrightarrow{c} CF), FF \xrightarrow{f} F\big)$ is an $\widehat{F}$-algebra.

③ *Initial morphism* Let $\big((A, d\colon A \to CA) \in C\text{-}\mathbf{Coalg}, a\colon \widehat{F}(A, d) \to (A, d)\big)$ be an $\widehat{F}$-algebra in C-$\mathbf{Coalg}(\mathcal{C})$, with a an F-algebra $\widehat{F}A \to A$ that preserves the C-coalgebra structure:

$$\begin{array}{ccccc} FA & & \xrightarrow{\quad a \quad} & & A \\ {\scriptstyle Fd}\downarrow & & & & \downarrow{\scriptstyle d} \\ FCA & \xrightarrow[\varphi_A]{} & CFA & \xrightarrow[Ca]{} & CA \end{array} \tag{‡}$$

As before, the diagram also makes $d\colon (A, a) \to \widehat{C}(A, a)$ into an F-algebra homomorphism. To prove initiality, we need to find a unique $\widehat{F}$-algebra homomorphism $k\colon ((F, c), f) \to ((A, d), a)$, i.e. a morphism $k\colon F \to A$ that is both ⑤ a C-coalgebra homomorphism $(F, c) \to (A, d)$ and ⑥ an F-algebra homomorphism $(F, f) \to (A, a)$. Since F is the initial F-algebra, we can induce k by initiality as the unique F-algebra homomorphism $k\colon (F, f) \to (A, a)$, satisfying ⑥. The C-coalgebra homomorphism condition ⑤ for k is the following diagram:

$$\begin{array}{ccc} F & \xrightarrow{k} & A \\ {\scriptstyle c}\downarrow & & \downarrow{\scriptstyle d} \\ CF & \xrightarrow[Ck]{} & CA \end{array}$$

Since CA is the carrier of the F-algebra $\widehat{C}(A, a)$ and $(F, f) \in F\text{-}\mathbf{alg}(\mathcal{C})$ is initial, it suffices to show that this is a diagram in F-$\mathbf{alg}(\mathcal{C})$ and therefore the two composites out of the initial object are equal by uniqueness. For this, we can use the assumptions that $d\colon (A, a) \to \widehat{C}(A, a) = (CA, \hat{a})$ is an F-algebra homomorphism by (‡), and so is $k\colon (F, f) \to (A, a)$ by ⑥.

$$(F, f) \xrightarrow{k} (A, a) \xrightarrow{d} \widehat{C}(A, a) \;=\; (F, f) \xrightarrow{c} \widehat{C}(F, f) \xrightarrow{\widehat{C}k} \widehat{C}(A, a)$$

④ *Uniqueness* Uniqueness of $k\colon ((F, c), f) \to ((A, d), a)$ follows from the uniqueness of k as an F-algebra homomorphism: any other $\widehat{F}$-algebra homomorphism would in particular be a morphism of F-algebras so must be equal to k.

□

§ Proof of Lemma 4.1.1 on page 82

Lemma 4.1.1 *For a biclosed $\mathcal{V}$-modular category $\mathcal{C}$ and all $A \in \mathcal{V}$, $X, Y \in \mathcal{C}$, we have a natural family of maps that is compatible with the biclosed structure:*

$$\ell^X_{Y,A} \colon \langle X, Y\rangle \otimes A \to \langle A \multimap X, Y\rangle \colon \mathcal{V} \times \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V}$$

$$\begin{array}{ccc} I \otimes A & \xrightarrow{j_X \otimes A} & \langle X, X\rangle \otimes A \\ {\scriptstyle \lambda_A}\downarrow & & \downarrow{\scriptstyle \ell^X_{X,A}} \\ A & \xrightarrow[\overline{\beta}^X_A]{} & \langle A \multimap X, X\rangle \end{array} \qquad (\ell\lambda)$$

$$\begin{array}{ccc} \langle X, Y\rangle \otimes I & \xrightarrow{\ell^X_{Y,I}} & \langle I \multimap X, Y\rangle \\ & {\scriptstyle \rho_{\langle X,Y\rangle}}\searrow \quad \swarrow{\scriptstyle \langle i_X, Y\rangle} & \\ & \langle X, Y\rangle & \end{array} \qquad (\ell\rho)$$

$$\begin{array}{ccccc} (\langle X, Y\rangle \otimes A) \otimes B & \xrightarrow{\ell^X_{Y,A} \otimes B} & \langle A \multimap X, Y\rangle \otimes B & \xrightarrow{\ell^{A \multimap X}_{Y,B}} & \langle B \multimap (A \multimap X), Y\rangle \\ {\scriptstyle \alpha_{\langle X,Y\rangle,A,B}}\downarrow & & & & \downarrow{\scriptstyle \langle c^{A,B}_X, Y\rangle} \\ \langle X, Y\rangle \otimes (A \otimes B) & & \xrightarrow[\ell^X_{Y,A\otimes B}]{} & & \langle (A \otimes B) \multimap X, Y\rangle \end{array} \qquad (\ell\alpha)$$

Proof We prove that currying and the transformation t are mates under the following adjunction:

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{B \multimap (-)} & \mathcal{C} \\ {\scriptstyle \langle -, Y\rangle}\left(\dashv\right){\scriptstyle (-) \multimap Y} & & {\scriptstyle \langle -, Y\rangle}\left(\dashv\right){\scriptstyle (-) \multimap Y} \\ \mathcal{V}^{\mathrm{op}} & \xrightarrow[(-)\otimes B]{} & \mathcal{V}^{\mathrm{op}} \end{array}$$

This establishes an isomorphism between uncurrying $\overline{c}^{A,B}_Y \colon B \multimap (A \multimap Y) \to (A \otimes B) \multimap Y$ and the transformation $\ell^X_{Y,B} \colon \langle X, Y\rangle \otimes B \to \langle B \multimap X, Y\rangle$. The naturality squares become, for $f \colon B \to \langle X, Y\rangle$ and $g \colon X \to (B \multimap Y)$

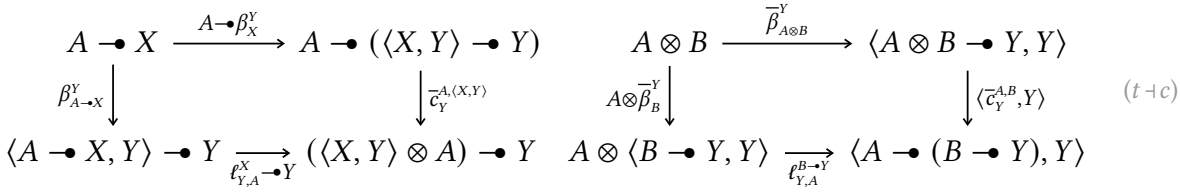
$$\begin{array}{ccc} A \multimap X & \xrightarrow{A \multimap \beta(f)} & A \multimap (B \multimap Y) \\ {\scriptstyle \beta(\ell^X_{Y,A})}\downarrow & & \downarrow{\scriptstyle \overline{c}^{A,\langle X,Y\rangle}_Y} \\ (\langle X, Y\rangle \otimes A) \multimap Y & \xrightarrow[(f\otimes A) \multimap Y]{} & (B \otimes A) \multimap Y \end{array}$$

$$\begin{array}{ccc} B \otimes A & \xrightarrow{\overline{\beta}(\overline{c}^{A,B}_Y)} & \langle A \multimap (B \multimap Y), Y\rangle \\ {\scriptstyle \overline{\beta}(g)\otimes A}\downarrow & & \downarrow{\scriptstyle \langle A \multimap g, Y\rangle} \\ \langle X, Y\rangle \otimes A & \xrightarrow[\ell^X_{Y,A}]{} & \langle A \multimap X, Y\rangle \end{array}$$

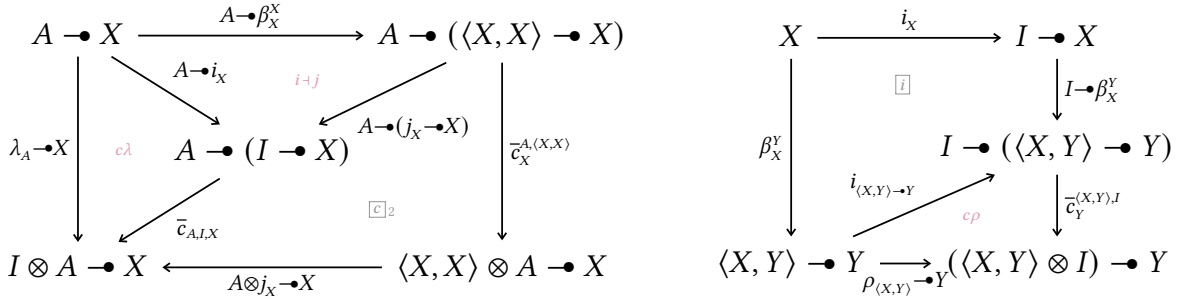
Explicitly, ℓ is defined as the $\overline{\beta}$-transpose of the composite

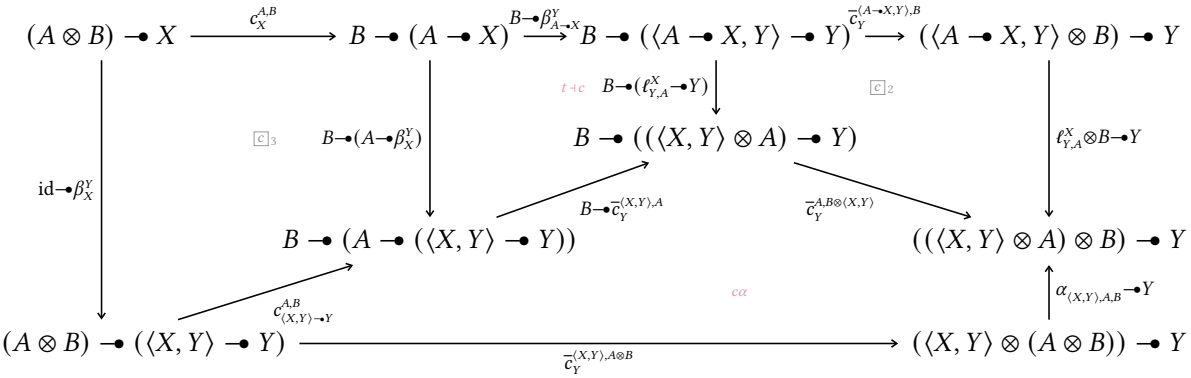
$$(A \multimap X) \xrightarrow{A \multimap \beta^Y_X} A \multimap (\langle X, Y\rangle \multimap Y) \xrightarrow{\overline{c}^{A,\langle X,Y\rangle}_Y} (\langle X, Y\rangle \otimes A) \multimap Y$$

and the naturality conditions at f and g become the transposition laws



The laws $\overline{\beta}$-transpose to





The transformation may also be derived from the enrichment as the composite

$$\langle X,Y\rangle \otimes A \xrightarrow{\mathrm{id}\otimes\overline{\beta}^{X}_{A}} \langle X,Y\rangle \otimes \langle A \multimap X, X\rangle \xrightarrow{M^{X}_{A\multimap X,Y}} \langle A \multimap X, Y\rangle$$

with the laws derived from axioms of M. □

---

§ Proof of Theorem 4.3.2 on page 91

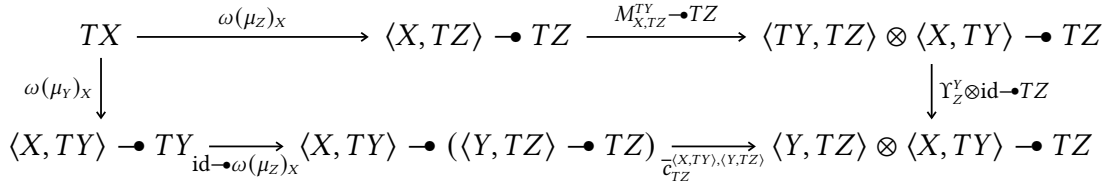
Theorem 4.3.2

Every powered monad gives rise to an enriched Kleisli triple, and every algebra for a powered monad induces an algebra for the corresponding enriched Kleisli triple.

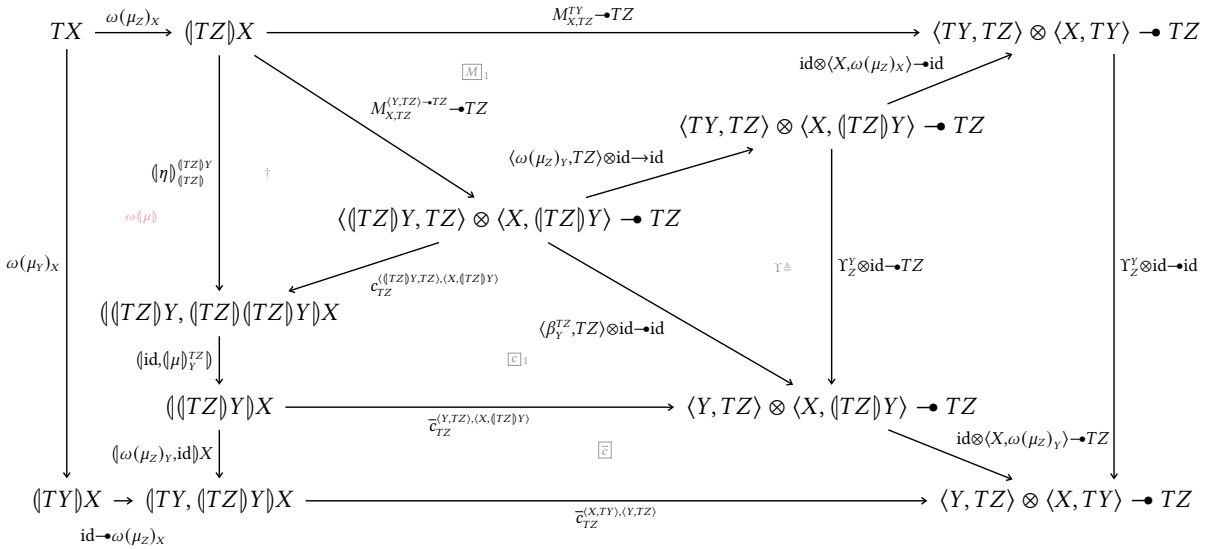
Proof Let (T,η,μ) be a monad with $p_{A,X}\colon T(A \multimap X) \to A \multimap TX$ a powering. We define $\Upsilon^{X}_{Y}\colon \langle X,TY\rangle \to \langle TX,TY\rangle$ as the β-transpose of $\omega(\mu_Y)_X\colon TX \Longrightarrow (\!|TY|\!)X$ that expands to

$$TX \xrightarrow{T(\!|\eta|\!)^{TY}_{X}} T(\langle X,TY\rangle \multimap TY) \xrightarrow{((\!|p|\!))_{TY,TY})_X} \langle X,TY\rangle \multimap TTY \xrightarrow{(\!|\mathrm{id},\mu_Y|\!)(X)} \langle X,TY\rangle \multimap TY$$

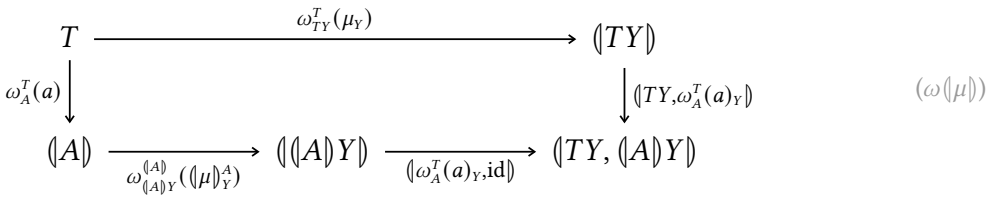
One unit law ($\eta\Upsilon$) transposes to the unit-preservation of the monad morphism $\omega(\mu_Y)$ from Theorem 4.3.1, while the other ($\Upsilon\eta$) transposes to one of the zig-zag identities of the adjunction $(-)(TY) \dashv (\!|TY, -|\!)$ applied to the powered monad T. The associativity law ($\Upsilon\Upsilon$) β-transposes to



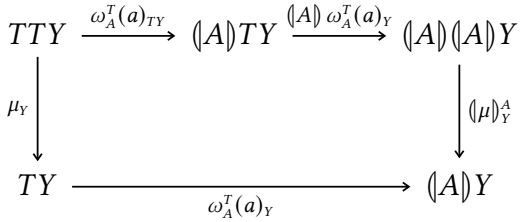
which commutes as follows:



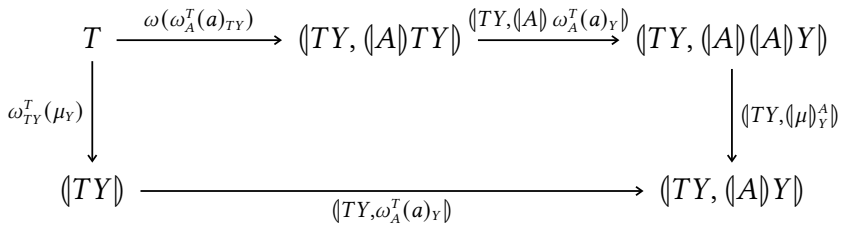
where † reduces to the equivalence of ℓ (that defines $p^{(\!|TZ|\!)}$ and therefore $(\!|\eta|\!)^{(\!|TZ|\!)Y}_{(\!|TZ|\!)}$) and M, and ($\omega(\!|\mu|\!)$) is an instance of the diagram



established as follows. Given $a\colon TA \to A$, the transpose $\omega^T_A(a)\colon T \Longrightarrow (\!|A|\!)$ is a monad morphism, so it satisfies the two monads' multiplication-preservation condition ($\varphi\lfloor\mu\rceil$):

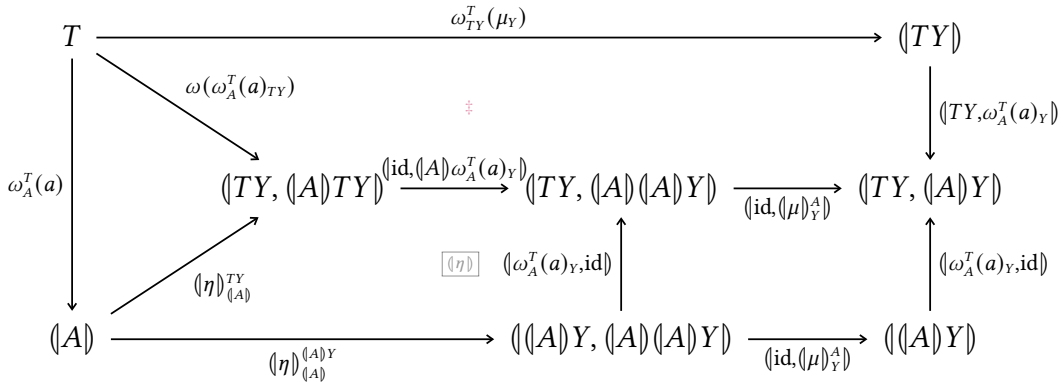


Transposing this using the adjunction $\omega\colon (-)(TY) \dashv (\!|TY, -|\!)$ gives the following:



(‡)

We use this diagram in the proof of ($\omega(\!|\mu|\!)$):



where the bottom composite amounts to the ω-transpose of $(\!|\mu|\!)_Y^A$. The reasoning for algebras for a powered monad is similar to the above, using the algebra structure instead of μ_Z. □

---

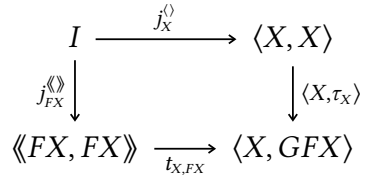
§ Proof of Lemma 5.1.1 on page 105

Lemma 5.1.1 *Given an adjunction $\tau\colon F \dashv G\colon \mathcal{C} \to \mathcal{D}$ between left skew-monoidal closed $\mathcal{V}$-modular categories, the following conditions are equivalent:*

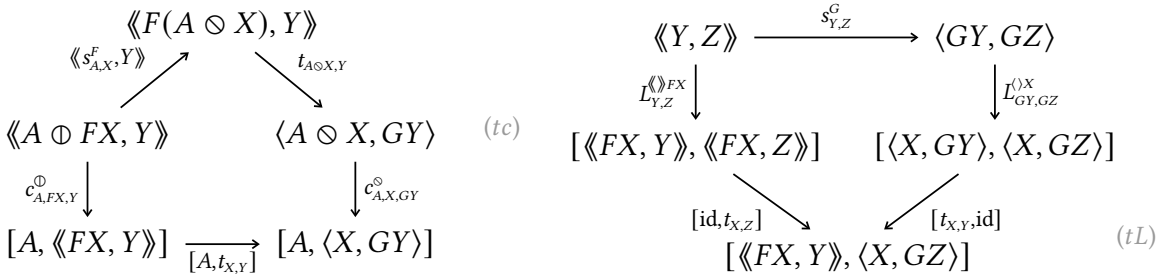
1. *F is an oplax left skew $\mathcal{V}^{\otimes}$-modular functor with strength $s^F_{A,Y}\colon F(A \otimes Y) \to A \oplus FY$*
2. *G is a lax left skew $\mathcal{V}^{[]}$-functor with strength $s^G_{Y,Z}\colon \langle\!\langle Y, Z\rangle\!\rangle \to \langle GY, GZ\rangle$;*
3. *τ has a* lax internal transpose

$$t_{X,Y}\colon \langle\!\langle FX, Y\rangle\!\rangle \to \langle X, GY\rangle\colon \mathcal{C} \times \mathcal{D} \to \mathcal{V}$$

that satisfies the unit and associativity axioms



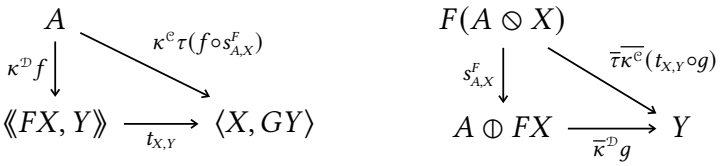
(tj)



Proof The bijection between $s^{\otimes}_{A,Y} \colon F(A \otimes X) \to A \oplus FY$ and $t_{X,Y} \colon \langle\!\langle FX, Y \rangle\!\rangle \to \langle X, GY \rangle$ is the Yoneda transpose

$$\begin{array}{ccc} \mathcal{D}\big(A \oplus FX, Y\big) & \xrightarrow{\mathcal{D}(s_{A,X},Y)} & \mathcal{D}(F(A \otimes X), Y) \\ \Big\downarrow{\scriptstyle \kappa^{\mathcal{D}}} & & \Big\downarrow{\scriptstyle \tau} \\ & & \mathcal{C}(A \otimes X, GY) \\ & & \Big\downarrow{\scriptstyle \kappa^{\mathcal{C}}} \\ \mathcal{V}\big(A, \langle\!\langle FX, Y \rangle\!\rangle\big) & \xrightarrow[\mathcal{V}(A,t_{X,Y})]{} & \mathcal{V}\big(A, \langle X, GY \rangle\big) \end{array}$$

that, evaluated at $f \colon A \oplus FX \to Y \in \mathcal{D}$ and $g \colon A \to \langle\!\langle FX, Y \rangle\!\rangle \in \mathcal{V}$ gives the following transposition schemes:

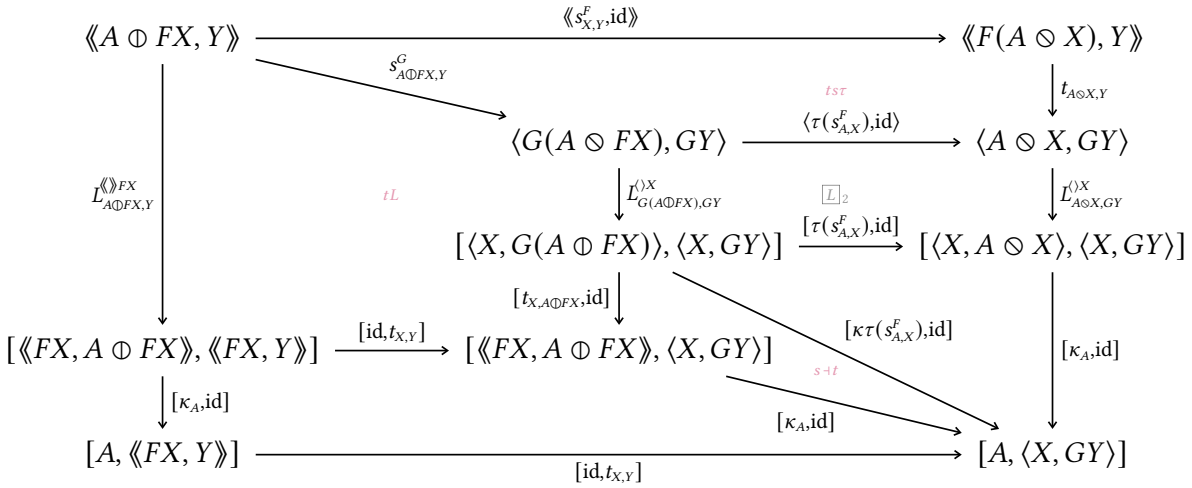


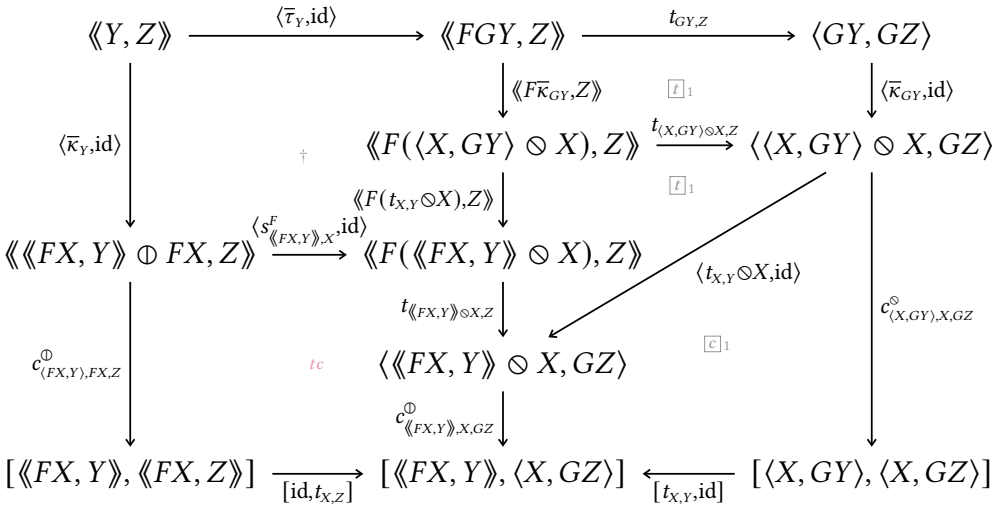
The transformations $s^{G}_{X,Y} \colon \langle\!\langle X, Y \rangle\!\rangle \to \langle GX, GY \rangle$ and t are mates under the functors on the left, inducing the diagram on the right for $f \colon FX \to Y \in \mathcal{D}$

We can transpose the oplax version of Diagram ($s\lambda\otimes$) using the transposition schemes to obtain the unit axiom for t from that of s^F:

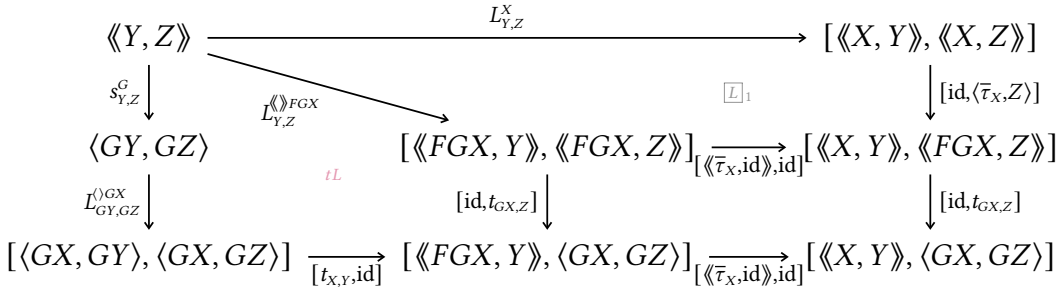
$$\frac{A \xrightarrow{\kappa^{\mathcal{C}}\tau(\lambda^{\otimes}_{FX} \circ s_{I,X})} \langle X, GFX \rangle}{\dfrac{A \xrightarrow{\kappa^{\mathcal{D}}(\lambda^{\oplus}_{FX})} \langle\!\langle FX, FX \rangle\!\rangle \xrightarrow{t_{X,X}} \langle X, GFX \rangle}{A \xrightarrow{j_{FX}} \langle\!\langle FX, FX \rangle\!\rangle \xrightarrow{t_{X,X}} \langle X, GFX \rangle}} \qquad \frac{F(I \otimes Y) \xrightarrow{F\lambda^{\otimes}_{Y}} FY \longrightarrow FY}{\dfrac{I \otimes Y \xrightarrow{\lambda^{\otimes}_{Y}} Y \xrightarrow{\tau_Y} GFY}{I \xrightarrow{j^{\langle\rangle}_{Y}} \langle Y, Y \rangle \xrightarrow{\langle Y, \tau_Y \rangle} \langle Y, GFY \rangle}}$$

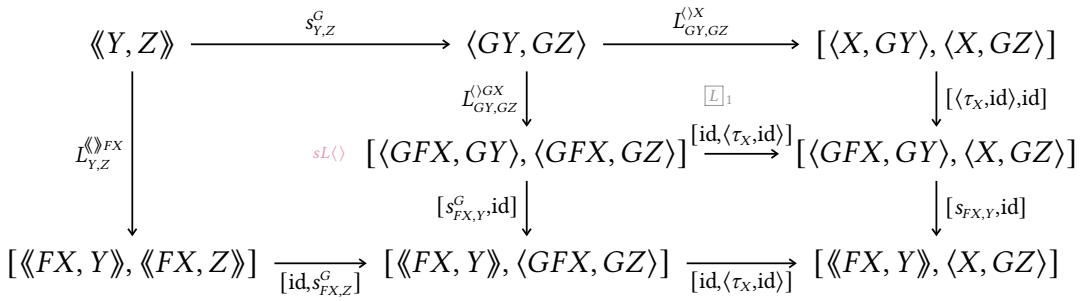
The associativity axiom of s^F transposes to the first associativity axiom of t. The equivalence of two associativity axioms for t is shown as follows:





where † is an instance of Diag. $(s \dashv t)$ with $g = \mathrm{id}\colon \langle FX, Y\rangle \to \langle FX, Y\rangle$, with the transposition $\overline{\tau\kappa}(t_{X,Y})\colon F(\langle\!\langle FX, Y\rangle\!\rangle \otimes X) \to Y$ equalling the composite $\overline{\tau}_Y \circ F\overline{\kappa}_{GY} \circ F(t_{X,Y} \otimes X)$ by naturality of $\overline{\tau}$ and $\overline{\kappa}$. Finally, the second associativity axiom of t is equivalent to the associativity of $s^{\langle\rangle}$:



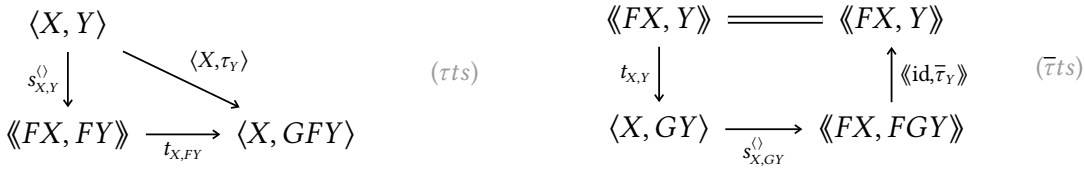


□

---

§ Proof of Lemma 5.1.2 on page 106

Lemma 5.1.2 *Given a strong $\mathcal{V}^{\otimes}$-modular functor F with right adjoint $\tau\colon F \dashv G\colon \mathcal{C} \to \mathcal{D}$, the induced $\mathcal{V}^{\langle\rangle}$-strength $s_{X,Y}\colon \langle X, Y\rangle \to \langle\!\langle FX, FX\rangle\!\rangle$ and internal transpose $t_{X,Y}\colon \langle\!\langle FX, Y\rangle\!\rangle \to \langle X, GY\rangle$ are compatible in the sense of the following diagrams:*

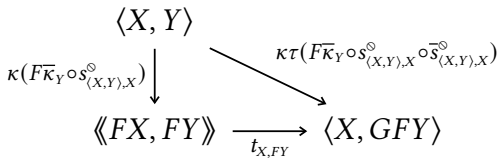


Proof A strong strength $s^{\otimes}_{A,X}\colon A \oplus FX \cong F(A \otimes X)$ decomposes into a lax and oplax direction: the former $s^{\otimes}_{A,X}\colon A \oplus FX \to F(A \otimes X)$ is equivalent to the $\mathcal{V}^{\langle\rangle}$-strength $s^{\langle\rangle}_{X,Y}\colon \langle X, Y\rangle \to \langle\!\langle FA, FB\rangle\!\rangle$, and the latter $\bar{s}^{\otimes}_{A,X}\colon F(A \otimes X) \to A \oplus FX$ induces the transpose $t_{X,Y}\colon \langle\!\langle FX, Y\rangle\!\rangle_{\mathcal{D}} \to \langle X, GY\rangle_{\mathcal{C}}$ by Lemma 5.1.1.

The proof proceeds by expressing the transformations back in terms of the two directions of $s^{\otimes}$. Firstly, $s^{\langle\rangle}_{X,Y}\colon \langle X, Y\rangle \to \langle\!\langle FX, FY\rangle\!\rangle$ expands as the κ-transpose of

$$\langle X, Y\rangle \oplus FX \xrightarrow{s^{\otimes}_{\langle X,Y\rangle,X}} F(\langle\!\langle X, Y\rangle\!\rangle \otimes X) \xrightarrow{F\bar{\kappa}_Y} FY$$

and, instantiating the transposition scheme Diag. ($s \dashv t$) with the composite above to get



where the inverse compositions of $s^{\otimes}$ cancel, it is sufficient to show that the $\kappa\tau$-transpose of

$$F(\langle X, Y\rangle \otimes X) \xrightarrow{F\bar{\kappa}_Y} FY$$

equals $\langle X, Y\rangle \xrightarrow{\langle X,\tau_Y\rangle} \langle X, GFY\rangle$, which is immediate by naturality.

The second diagram $\bar{\kappa}$-transposes to the counit $\bar{\kappa}_Y\colon \langle\!\langle FX, Y\rangle\!\rangle \oplus FX \to Y$ equalling the transpose of the bottom composite

$$\langle\!\langle FX, Y\rangle\!\rangle \oplus FX \xrightarrow{t_{X,FY}\otimes\mathrm{id}} \langle X, GY\rangle \oplus FX \xrightarrow{s^{\otimes}_{\langle X,GY\rangle,X}} F(\langle X, GY\rangle \otimes X) \xrightarrow{F(\bar{\kappa}_{GY})} FGY \xrightarrow{\bar{\tau}_Y} Y$$

which, by naturality of $s^{\oslash}$, equals

$$\langle\!\langle FX, Y \rangle\!\rangle \oplus FX \xrightarrow{s^{\oslash}_{\langle FX,Y\rangle,X}} F(\langle\!\langle FX, Y \rangle\!\rangle \oslash X) \xrightarrow{\overline{\tau\kappa}(t_{X,Y})} Y$$

By the transposition scheme Diag. ($s \dashv t$) with $g \triangleq \mathrm{id}$, the second map $\overline{\tau\kappa}(t_{X,Y})$

$$\begin{array}{ccc} F(\langle\!\langle FX, Y \rangle\!\rangle \oslash X) & & \\ \downarrow{\scriptstyle \overline{s}^{\oslash}_{\langle FX,Y\rangle\!\rangle,X}} & \searrow^{\overline{\tau\kappa}(t_{X,Y})} & \\ \langle\!\langle FX, Y \rangle\!\rangle \oplus FX & \xrightarrow[\overline{\kappa}_Y]{} & Y \end{array}$$

and the inverse compositions of $s^{\oslash}$ cancel, leaving $\langle\!\langle FX, Y \rangle\!\rangle \oplus FX \xrightarrow{\overline{\kappa}_Y} Y$, as required. □

---

§ Proof of Proposition 6.1.1 on page 128

Proposition 6.1.1 *If $\mathcal{A}_{|\mathcal{V}}$ and $\mathcal{B}_{|\mathcal{W}}$ are representable synthetic monoidal categories, and a monoidal functor $K\colon \mathcal{V} \to \mathcal{W}$ lifts to a $L\colon \mathcal{A} \to \mathcal{B}$ with $K\underline{A}_{\mathcal{A}} = \underline{LA}_{\mathcal{B}}$, then L is representable synthetic monoidal $\mathcal{A}_{|\mathcal{V}} \to \mathcal{B}_{|\mathcal{W}}$.*

Proof Suppose we have a lifting of a monoidal K to L with $K\underline{A}_{\mathcal{A}} = \underline{LA}_{\mathcal{B}}$. We show that L is representable synthetic monoidal, with unit, multiplication, and laws as follows

$$u[f\colon I \to \underline{C}_{\mathcal{A}}] \triangleq J \xrightarrow{u^K} KI \xrightarrow{Kf} K\underline{C}_{\mathcal{A}} = \underline{LC}_{\mathcal{B}}$$

$$m[g\colon \underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}} \to \underline{C}_{\mathcal{A}}] \triangleq \underline{LA}_{\mathcal{B}} \oplus \underline{LB}_{\mathcal{B}} = K\underline{A}_{\mathcal{A}} \oplus K\underline{B}_{\mathcal{A}} \xrightarrow{m^K_{\underline{A},\underline{B}}} K(\underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}}) \xrightarrow{Kg} K\underline{C}_{\mathcal{A}} = \underline{LC}_{\mathcal{B}}$$

- The diagram on the left makes the bottom right of the left unit law commute:

$$\begin{array}{ccc} I \otimes \underline{B}_{\mathcal{A}} & \xrightarrow{\underline{f}\otimes\mathrm{id}} & \underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}} \\ {\scriptstyle \lambda^{\otimes}_{\underline{B}}}\downarrow & & \downarrow{\scriptstyle \underline{g}} \\ \underline{B}_{\mathcal{A}} & \xleftarrow[\underline{h}]{} & \underline{C}_{\mathcal{A}} \end{array}$$

$$\begin{array}{ccccc} J \oplus K\underline{B}_{\mathcal{A}} & \xrightarrow{u^K\oplus\mathrm{id}} & KI \oplus K\underline{B}_{\mathcal{A}} & \xrightarrow{K\underline{f}\oplus\mathrm{id}} & K\underline{A}_{\mathcal{A}} \oplus K\underline{B}_{\mathcal{A}} \\ & uh & \downarrow{\scriptstyle m^K_{I,\underline{B}}} & \boxed{m}_1 & \downarrow{\scriptstyle m^K_{\underline{A},\underline{B}}} \\ {\scriptstyle \lambda^{\oplus}_{K\underline{B}}}\downarrow & & K(I \otimes \underline{B}_{\mathcal{A}}) & \xrightarrow[K(\underline{f}\otimes\mathrm{id})]{} & K(\underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}}) \\ & \searrow^{K\lambda^{\otimes}_{\underline{B}}} & & & \downarrow{\scriptstyle K\underline{g}} \\ K\underline{B}_{\mathcal{A}} & & \xleftarrow[K\underline{h}]{} & & K\underline{C}_{\mathcal{A}} \end{array}$$

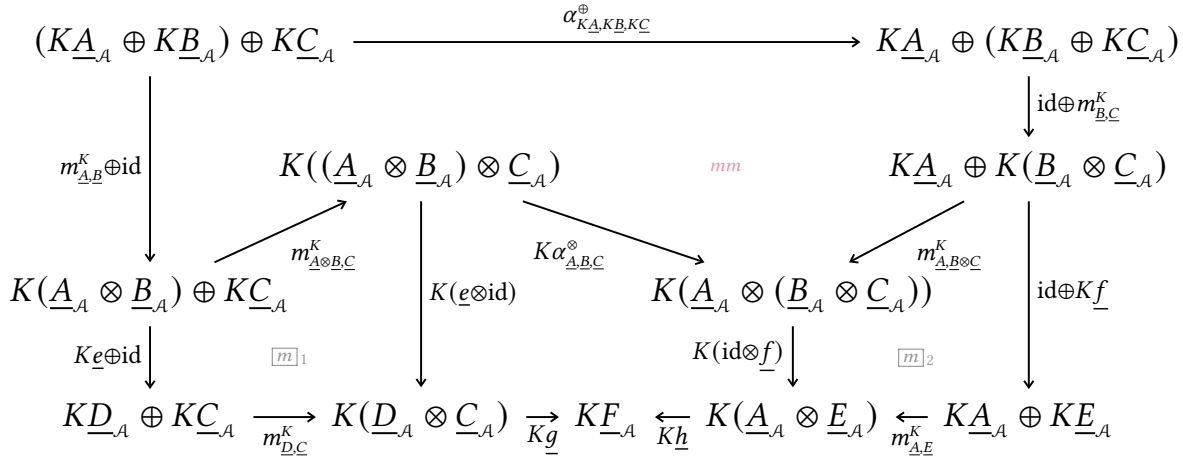
- The diagram on the left makes the top half of the right unit law commute:

$$\begin{array}{ccc} \underline{A}_{\mathcal{A}} & \xrightarrow{\underline{h}} & \underline{C}_{\mathcal{A}} \\ {\scriptstyle \rho^{\otimes}_{\underline{A}}}\downarrow & & \uparrow{\scriptstyle \underline{g}} \\ \underline{A}_{\mathcal{A}} \otimes I & \xrightarrow[\mathrm{id}\otimes f]{} & \underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}} \end{array}$$

$$\begin{array}{ccccc} K\underline{A}_{\mathcal{A}} & & \xrightarrow{K\underline{h}} & & K\underline{C}_{\mathcal{A}} \\ & \searrow^{K\rho^{\otimes}_{\underline{A}}} & & & \uparrow{\scriptstyle K\underline{g}} \\ {\scriptstyle \rho^{\oplus}_{K\underline{A}}}\downarrow & & K(\underline{A}_{\mathcal{A}} \otimes I) & \xrightarrow[K(\mathrm{id}\otimes\underline{f})]{} & K(\underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}}) \\ & mu & \uparrow{\scriptstyle m^K_{\underline{A},I}} & \boxed{m}_2 & \uparrow{\scriptstyle m^K_{\underline{A},\underline{B}}} \\ K\underline{A}_{\mathcal{A}} \oplus J & \xrightarrow[\mathrm{id}\oplus u^K]{} & K\underline{A}_{\mathcal{A}} \oplus KI & \xrightarrow[\mathrm{id}\oplus K\underline{f}]{} & K\underline{A}_{\mathcal{A}} \oplus K\underline{B}_{\mathcal{A}} \end{array}$$

- The diagram on the top makes the middle bottom pentagon of the associativity law below it commute:

$$\begin{array}{ccccc}
(\underline{A}_{\mathcal{A}} \otimes \underline{B}_{\mathcal{A}}) \otimes \underline{C}_{\mathcal{A}} & & \xrightarrow{\alpha^{\otimes}_{\underline{A},\underline{B},\underline{C}}} & & \underline{A}_{\mathcal{A}} \otimes (\underline{B}_{\mathcal{A}} \otimes \underline{C}_{\mathcal{A}}) \\
\downarrow{\scriptstyle \underline{e}\otimes\mathrm{id}} & & & & \downarrow{\scriptstyle \mathrm{id}\otimes\underline{f}} \\
\underline{D}_{\mathcal{A}} \otimes \underline{C}_{\mathcal{A}} & \xrightarrow{\underline{g}} & \underline{F}_{\mathcal{A}} & \xleftarrow{\underline{h}} & \underline{A}_{\mathcal{A}} \otimes \underline{E}_{\mathcal{A}}
\end{array}$$



□

---

§ Proof of Theorem 7.1.3 on page 141

Theorem 7.1.3

Suppose we have an adjunction $F \dashv G \colon \mathcal{C} \to \mathcal{V}$ for $(\mathcal{V}, I, \otimes, [-,=])$ skew-monoidal closed and $(\mathcal{C}, \oslash, \langle -,= \rangle)$ a left skew-monoidal closed action. Then, F is a skew-monoidal warping riding $\oslash$ if and only if G is a skew-closed warping riding $\langle -,= \rangle$.

Proof We follow a similar proof pattern to Eilenberg and Kelly (1966) in establishing an equivalence between monoidal and closed structure over a category: namely, by finding intermediate forms for the transformations and axioms that can be shown equivalent. We write $\tau \colon \mathcal{V}(FX, B) \cong \mathcal{C}(X, GB)$, $\xi \colon \mathcal{V}(A \otimes B, C) \cong \mathcal{V}(A, [B, C])$ and $\kappa \colon \mathcal{C}(A \oslash Y, Z) \cong \mathcal{V}(A, \langle Y, Z \rangle)$ for the adjoint transpose isomorphisms.

Relating the data

- The skew-monoidal and skew-closed warping functors are adjoint by assumption: $F \dashv G$.
- The unit objects $J \in \mathcal{C}$ are identical.
- The morphism $v \colon FJ \to I \in \mathcal{V}$ and $u \colon J \to GI \in \mathcal{C}$ are adjoint transposes: $u = \tau v$.
- The natural transformations $k_X \colon X \to FX \oslash J$ and $l_X \colon G\langle J, X \rangle \to X$ are in bijection through the Yoneda embedding as shown on the left, and evaluating it on $f \colon FX \oslash J \to Y$ gives us the transposition scheme on the right:

$$\begin{array}{ccc} \mathcal{C}(FX \oslash J, Y) & \xrightarrow{\mathcal{C}(k_X,Y)} & \mathcal{C}(X,Y) \\ {\scriptstyle\kappa}\big| & & \| \\ \mathcal{V}(FX, \langle F, Y\rangle) & & \| \\ {\scriptstyle\tau}\big| & & \| \\ \mathcal{C}(X, G\langle J, Y\rangle) & \xrightarrow[\mathcal{C}(X,l_X)]{} & \mathcal{C}(X,Y) \end{array} \qquad \begin{array}{ccc} X & \xrightarrow{\tau\kappa f} & G\langle J, Y\rangle \\ & {\scriptstyle f\circ k_X}\searrow & \big\downarrow{\scriptstyle l_Y} \\ & & Y \end{array} \qquad (k \dashv l)$$

- The natural transformation $p_{X,Y} \colon F(FX \oslash Y) \to FX \otimes FY \colon \mathcal{C} \times \mathcal{C} \to \mathcal{V}$ is in bijection with the *warped transpose* $t_{X,B} \colon G[FX, B] \to G\langle X, GB\rangle \colon \mathcal{C} \times \mathcal{V} \to \mathcal{C}$ via Yoneda:

$$\begin{array}{ccc} \mathcal{V}(FX \otimes FY, B) & \xrightarrow{\mathcal{V}(p_{X,Y},B)} & \mathcal{V}(F(FX \oslash Y), B) \\ {\scriptstyle\check{\xi}}\big| & & \big|{\scriptstyle\tau} \\ \mathcal{V}(FX, [FY, B]) & & \mathcal{C}(FX \oslash Y, GB) \\ {\scriptstyle\tau}\big| & & \big|{\scriptstyle\kappa} \\ \mathcal{C}(X, G[FY, B]) & \xrightarrow[\mathcal{C}(X,t_{Y,B})]{} & \mathcal{V}(FX, \langle Y, GB\rangle) \\ & & \big|{\scriptstyle\tau} \\ & & \mathcal{C}(X, G\langle Y, GB\rangle) \end{array}$$

Evaluating this at $f \colon FX \otimes FY \to C$ and $g \colon X \to G[FY, B]$ gives the transposition schemes

$$\begin{array}{ccc} X & \xrightarrow{\tau\check{\xi}f} & G[FY, C] \\ & {\scriptstyle\tau\kappa\tau(f\circ p_{X,Y})}\searrow & \big\downarrow{\scriptstyle t_{Y,C}} \\ & & G\langle Y, GC\rangle \end{array} \qquad \begin{array}{ccc} F(FX \oslash Y) & \xrightarrow{p_{X,Y}} & FX \otimes FY \\ & {\scriptstyle\overline{\tau\kappa\tau}(t_{Y,B}\circ g)}\searrow & \big\downarrow{\scriptstyle\overline{\check{\xi}}\overline{\tau}g} \\ & & B \end{array} \qquad (p \dashv t)$$

The transformations $t_{X,B} \colon G[FX, B] \to G\langle X, GB\rangle$ and $q_{A,B} \colon G[A, B] \to G\langle GA, GB\rangle$ are mates under the adjunction on the left, such that for all $z \colon FX \to A$, the right square commutes:

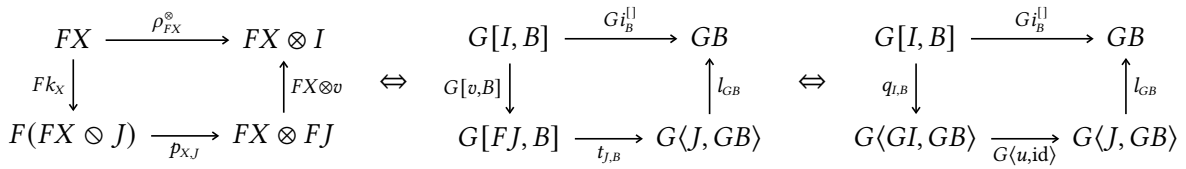
$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{G\langle -, GB\rangle} & \mathcal{C}^{\mathrm{op}} \\ F \big(\dashv \big) G & & \| \\ \mathcal{V} & \xrightarrow[G[-,B]]{} & \mathcal{C}^{\mathrm{op}} \end{array} \qquad \begin{array}{ccc} G[A, B] & \xrightarrow{q_{A,B}} & G\langle GA, GB\rangle \\ {\scriptstyle G[z,B]}\big\downarrow & & \big\downarrow{\scriptstyle G\langle \tau z, \mathrm{id}\rangle} \\ G[FX, B] & \xrightarrow[t_{X,B}]{} & G\langle X, GB\rangle \end{array} \qquad (tq\tau)$$

Instantiating this at the identity, we have expansions of t and q in terms of each other:

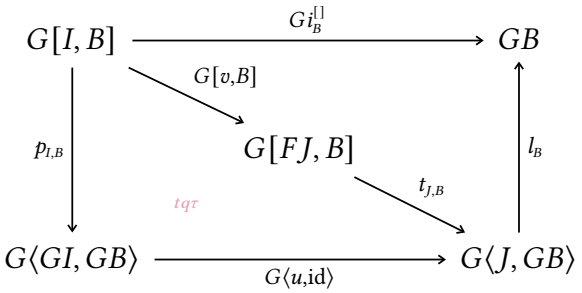
$$\begin{array}{ccc} G[A, B] & \xrightarrow{G[\overline{\tau},B]} & G[FGA, B] \\ & {\scriptstyle q_{A,B}}\searrow & \big\downarrow{\scriptstyle t_{GA,B}} \\ & & G\langle GA, GB\rangle \end{array} \qquad \begin{array}{ccc} G[FY, B] & \xrightarrow{q_{FY,B}} & G\langle GFY, GB\rangle \\ & {\scriptstyle t_{Y,B}}\searrow & \big\downarrow{\scriptstyle G\langle \tau, B\rangle} \\ & & G\langle Y, GB\rangle \end{array} \qquad (t \dashv q)$$

Relating the axioms Again, we connect the skew-monoidal and skew-closed axioms via properties of t defined above.

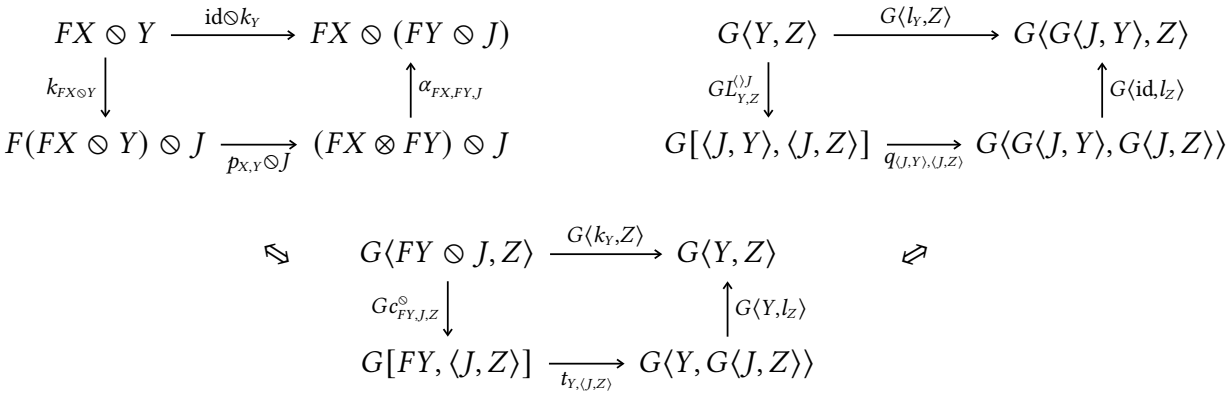
$(\rho p) \Leftrightarrow (iq)$



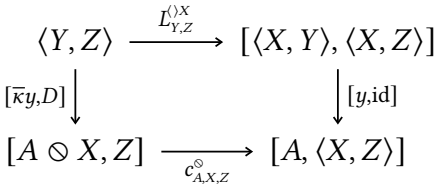
The Yoneda embedding of the left diagram transposes to the middle one. Then, instantiating Diagram $(tq\tau)$ with $z \triangleq v \colon FJ \to I$ gives the equivalence between the middle and right axioms.



$(\alpha k) \Leftrightarrow (Ll)$

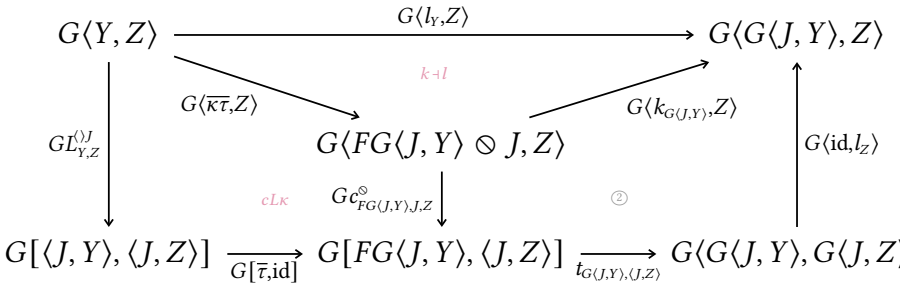


The Yoneda embedding of the left diagram transposes to the middle one. Then, assuming the middle diagram ②, the right diagram ③ is derived by expanding $q_{[J,Y],[J,Z]}$ as $t_{G\langle J,Y\rangle,\langle J,Z\rangle} \circ G[\overline{\tau}, \mathrm{id}]$, instantiating the transposition scheme between c and L obtained as mates:

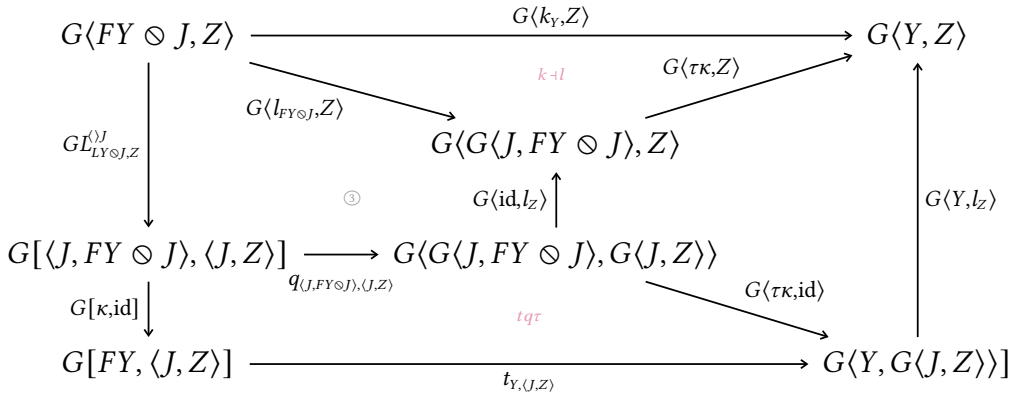


$(cL\kappa)$

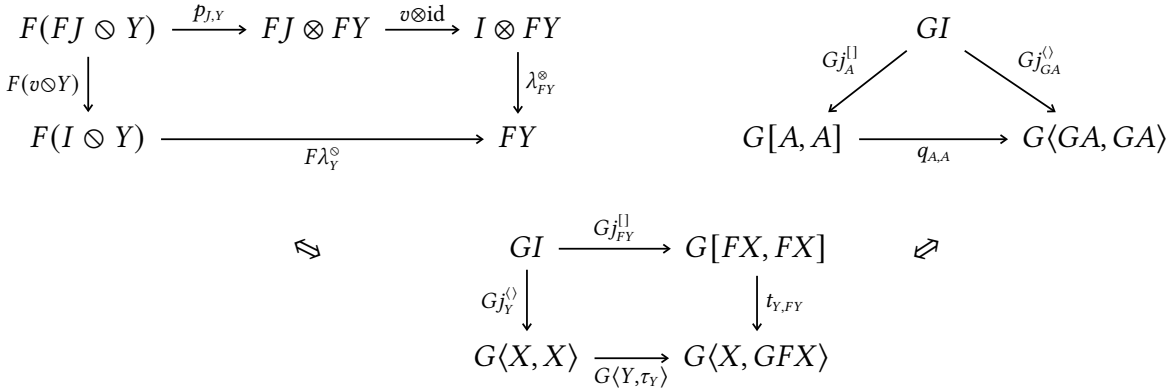
with $y \triangleq \overline{\tau}(\mathrm{id}_{G\langle J,Y\rangle}) \colon FG\langle J,Y\rangle \to \langle J,Y\rangle$, and $(k \dashv l)$ with $f \triangleq \overline{\kappa\tau}(\mathrm{id}_{G\langle J,Y\rangle}) \colon FG\langle J,Y\rangle \otimes J \to Y$:



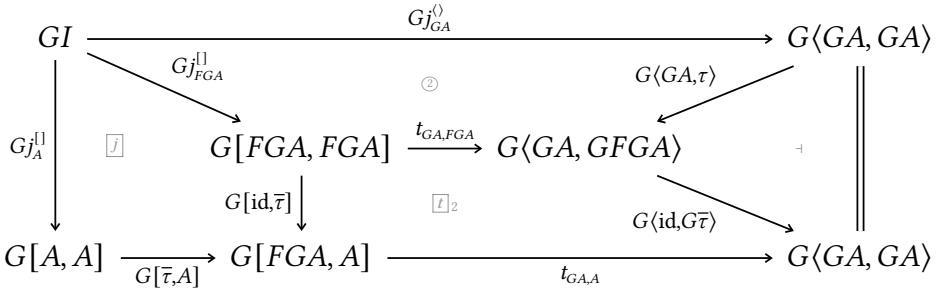
Conversely, axiom ③ implies axiom ② after expanding $c^{\otimes}_{FY,J,Z} \colon \langle FY \otimes J,Z\rangle \to [FY,\langle J,Z\rangle]$ as the composite $\langle FY \otimes J,Z\rangle \xrightarrow{L^{J}_{FY\otimes J,Z}} [\langle J,FY \otimes J\rangle,\langle J,Z\rangle] \xrightarrow{[\kappa,\mathrm{id}]} [FY,\langle J,Z\rangle]$, instantiating Diag. $(k \dashv l)$ with $f \triangleq \mathrm{id} \colon FY \otimes J \to FY \otimes J$, and Diag. $(tq\tau)$ with $\kappa(\mathrm{id}_{FY\otimes J}) \colon FY \to \langle J,FY \otimes J\rangle$:



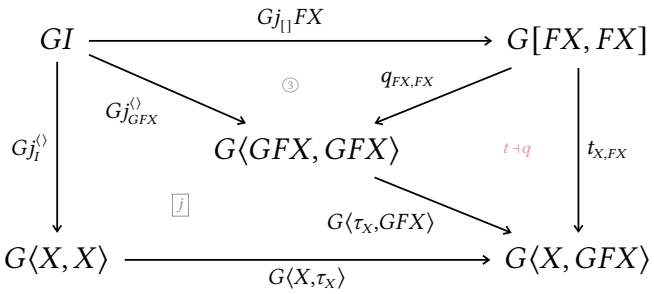
$(\lambda p) \Leftrightarrow (jq)$



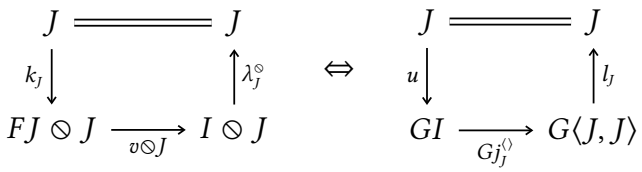
The top composite of the left diagram ① is an instance of Diag. $(p \dashv t)$, with the composite $f \triangleq \lambda^{\otimes}_{FY} \circ (v \otimes FY) \colon FJ \otimes FY \to FY$, so we have that $\tau\kappa\tau(\lambda^{\otimes}_{FY} \circ (v \otimes FY) \circ p_{J,Y}) = t_{J,FY} \circ \tau\xi(\lambda^{\otimes}_{FY} \circ (v \otimes FY))$. Then, $\lambda^{\otimes}_{FY} \circ (v \otimes FY)$ transposes to $Gj^{[]}_{FY} \circ u$, and the bottom composite $F\lambda^{\otimes}_{Y} \circ F(v \otimes Y)$ of ① transposes to $G\langle Y,\tau\rangle \circ Gj_{\langle\rangle}Y \circ u$, by naturality of τ, κ and ξ. Their equality in the middle diagram ② is implied by the equality of ①. To show that ② implies ③, we expand $q_{A,A}$ as $t_{GA,A} \circ G[\overline{\tau},A]$, and apply the zig-zag identity to show that $G\overline{\tau}(\mathrm{id}_{FGA}) \circ \tau(\mathrm{id}_{FGA}) = G\varepsilon_A \circ \eta_{GA} = \mathrm{id}$:



Conversely, assuming ③, we construct axiom ② using Diagram $(t \dashv q)$:



$(\lambda v) \Leftrightarrow (ju)$

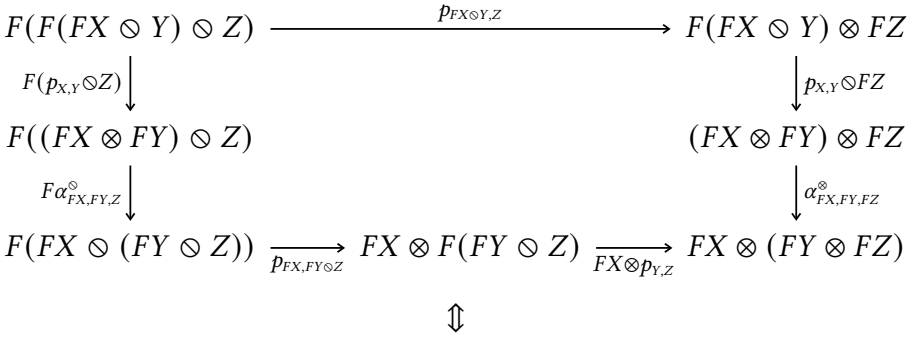


The transposition scheme Diagram $(k \dashv l)$ states that

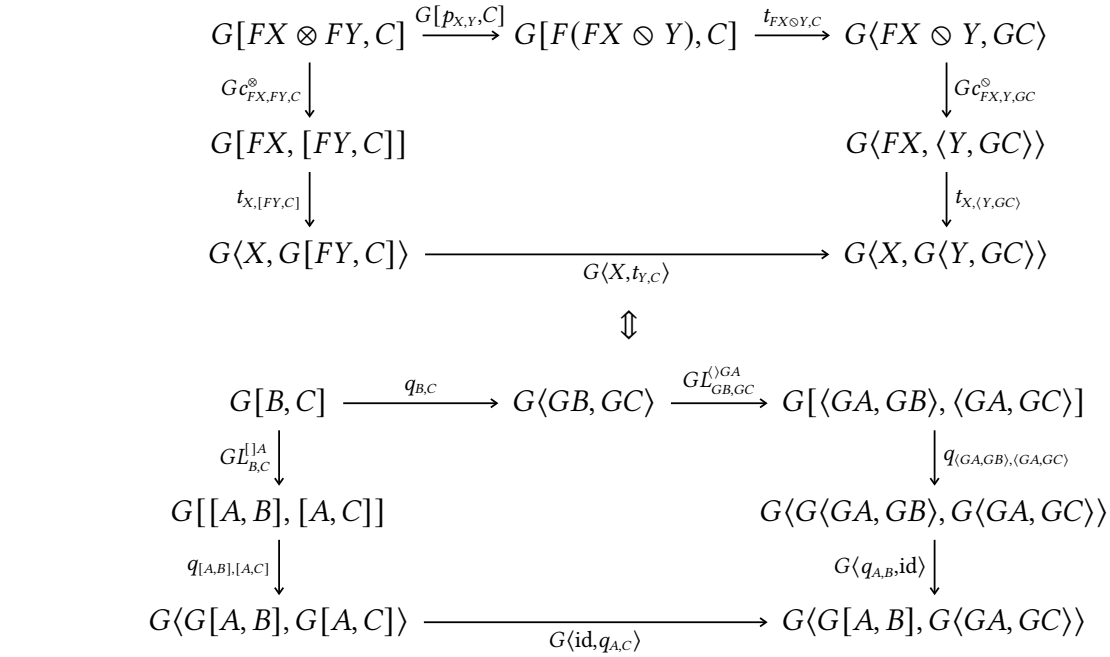
$$(\lambda_J \circ (v \otimes J)) \circ k_J = l_J \circ \tau\kappa\big(\lambda_J \circ (v \otimes J)\big)$$

and the transpose of $\lambda_J \circ (v \otimes J)$ is $J \xrightarrow{u} GI \xrightarrow{Gj_J} G\langle J, J\rangle$ by naturality.

$(\alpha p) \Leftrightarrow (Lq)$



⇕

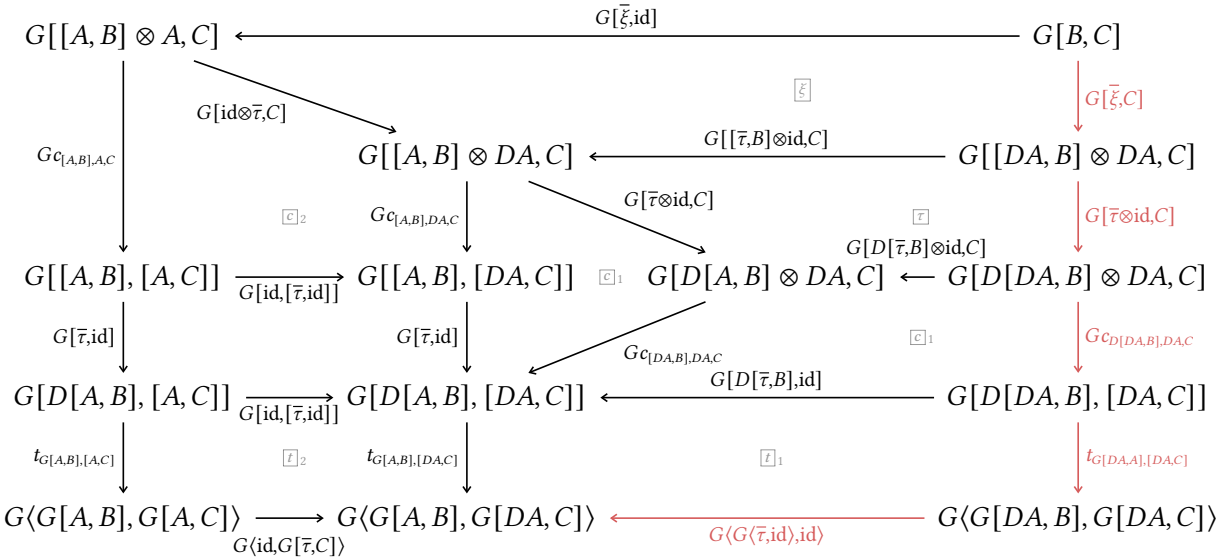


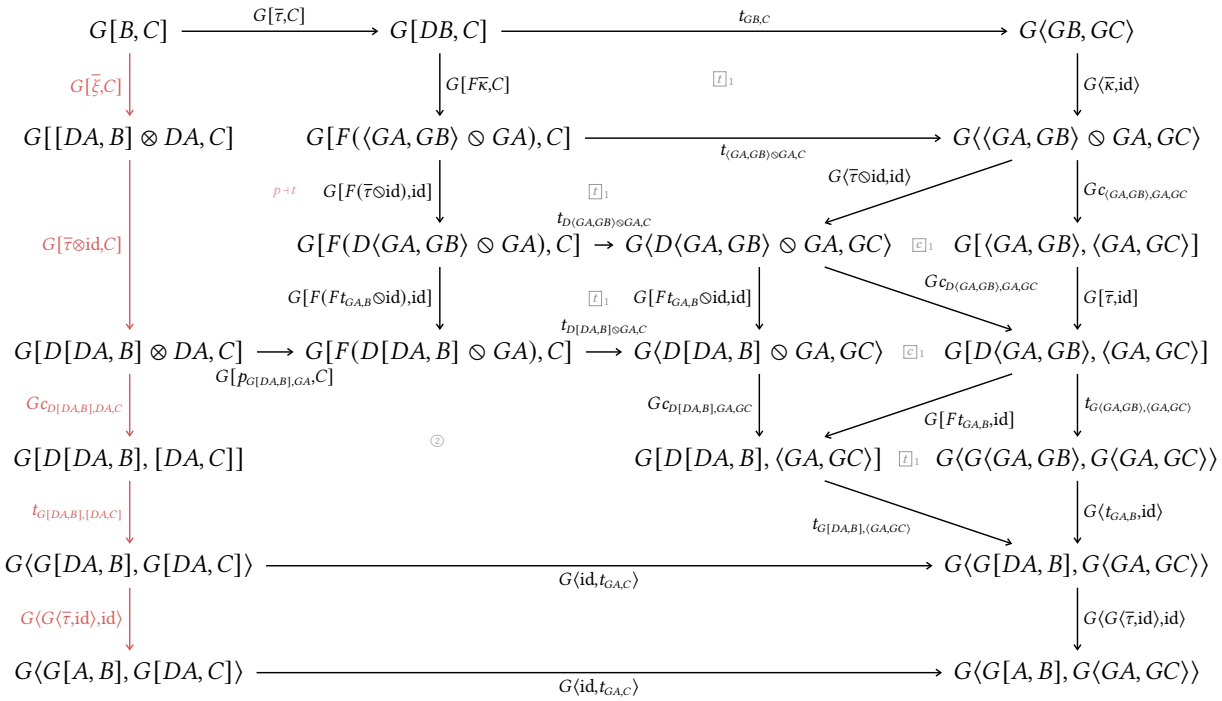
The first diagram ① transposes to the second one ② via Yoneda. Axiom ③ follows from ② using the expansions of L in terms of c, and q in terms of t:

$$[B, C] \xrightarrow{[\overline{\zeta}, C]} [[A, B] \otimes A, C] \xrightarrow{c_{[A,B],A,C}} [[A, B], [A, C]]$$

$$G[B, C] \xrightarrow{G[\overline{\tau}, C]} G[FGB, C] \xrightarrow{t_{GB,C}} G\langle GB, GC\rangle$$

The proof of ③ is made up of the following diagrams, joined together by the red composites. We abbreviate $FG\colon \mathcal{V} \to \mathcal{V}$ as D and $GF\colon \mathcal{C} \to \mathcal{C}$ as T.

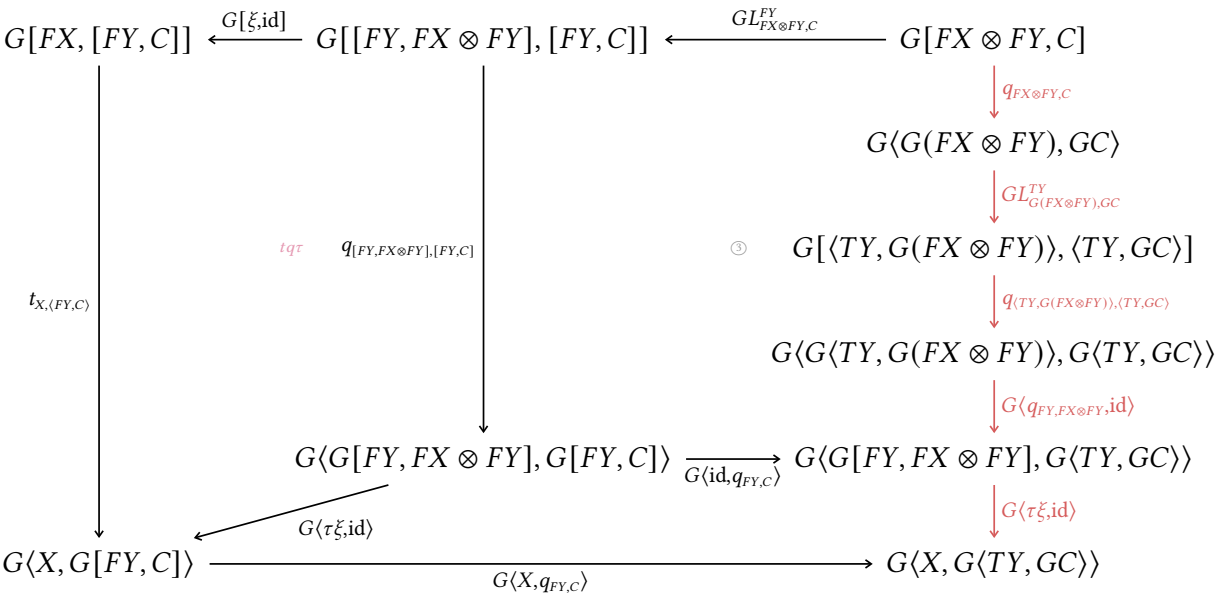


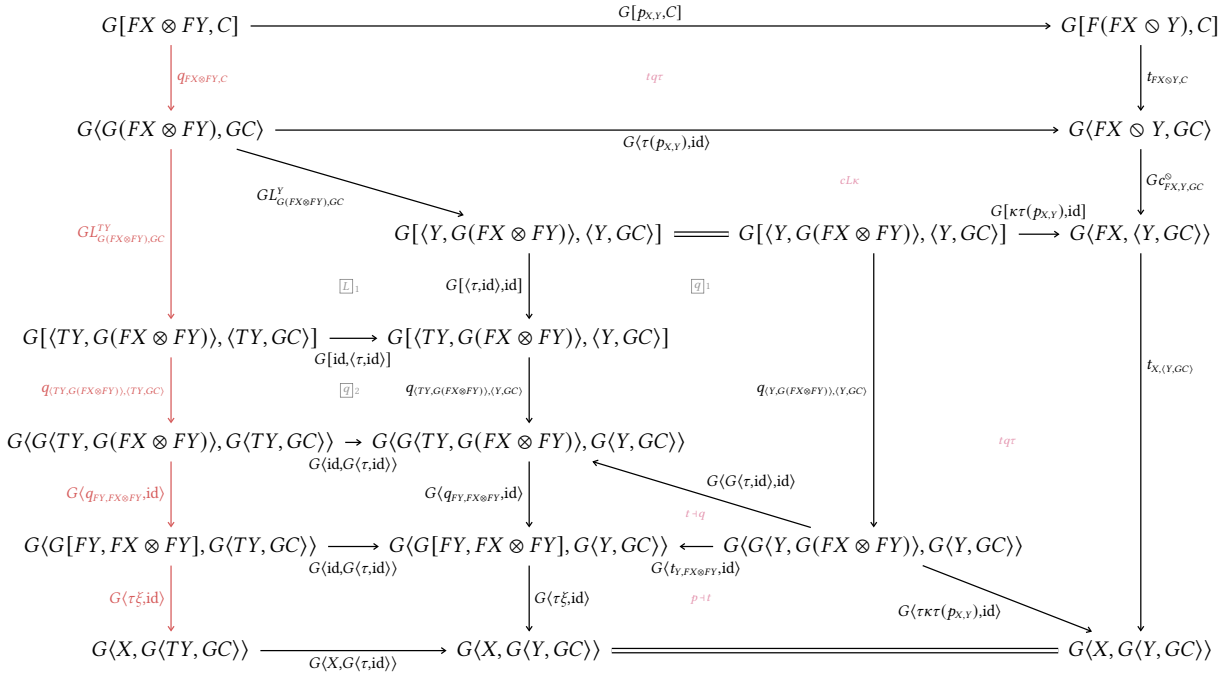


Conversely, from ③ we derive ② by the opposite expansions of c and t in terms of L and q:

$$[A\otimes B,C] \xrightarrow{L^B_{A\otimes B,C}} [[B,A\otimes B],[B,C]] \xrightarrow{[\overline{\xi},\mathrm{id}]} [A,[B,C]]$$

$$G[FA,B] \xrightarrow{q_{FA,B}} G\langle GFA,GB\rangle \xrightarrow{G\langle\tau,\mathrm{id}\rangle} G\langle A,GB\rangle$$





□

---

§ Proof of Theorem 7.1.4 on page 142

Theorem 7.1.4

For an adjoint triple $F \dashv G \dashv H \colon \mathcal{C} \to \mathcal{V}$, if $G \colon \mathcal{V} \to \mathcal{C}$ is a strong $\mathcal{V}^{\circledcirc}$-module functor, then F and G form adjoint warpings.

Proof Let $\tau \colon F \dashv G \colon \mathcal{V} \to \mathcal{C}$ and $\pi \colon G \dashv H \colon \mathcal{C} \to \mathcal{V}$ be the two adjunctions, and suppose G is a strong $\mathcal{V}^{\circledcirc}$-module functor from $(\mathcal{V}, \otimes)$ to $(\mathcal{C}, \circledcirc)$, with strength $A \circledcirc GB \cong G(A \otimes B)$. From Lemma 5.1.2, we have induced and compatible natural transformations $s_{A,B} \colon [A, B] \to \langle GA, GB \rangle$ and $t_{A,X} \colon \langle GA, X \rangle \to [A, HX]$. With these, we construct a skew-closed warping structure on $G \colon \mathcal{V} \to \mathcal{C}$, which, by the previous theorem, equivalently induces a warping on F.

- The warping functor and object are $G \colon \mathcal{V} \to \mathcal{C}$ and $GI \colon \mathcal{C}$;
- The morphism $u \colon GI \to GI \in \mathcal{C}$ is the identity;
- the natural transformation $l_X \colon G\langle J, X \rangle \to X$ is the transpose of

$$\langle GI, X \rangle \xrightarrow{t_{I,X}} [I, HX] \xrightarrow{i_X^{[]}} HX$$

- the natural transformation $q_{A,B} \colon G[A, B] \to G\langle GA, GB \rangle$ is

$$G[A, B] \xrightarrow{Gs_{A,B}} G\langle GA, GB \rangle$$

The warping axioms are as follows. Diagram (*iq*) π-transposes to

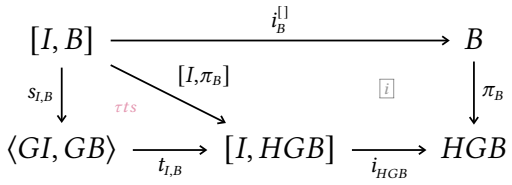


Diagram (*jq*) is the strength law Diagram (*sj⟨⟩*). The axiom (*ju*) π-transposes to:

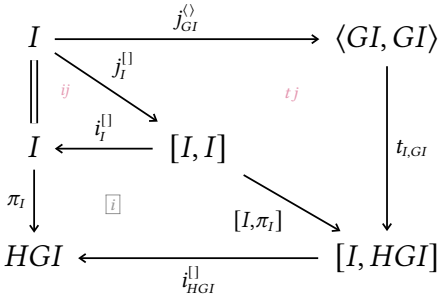
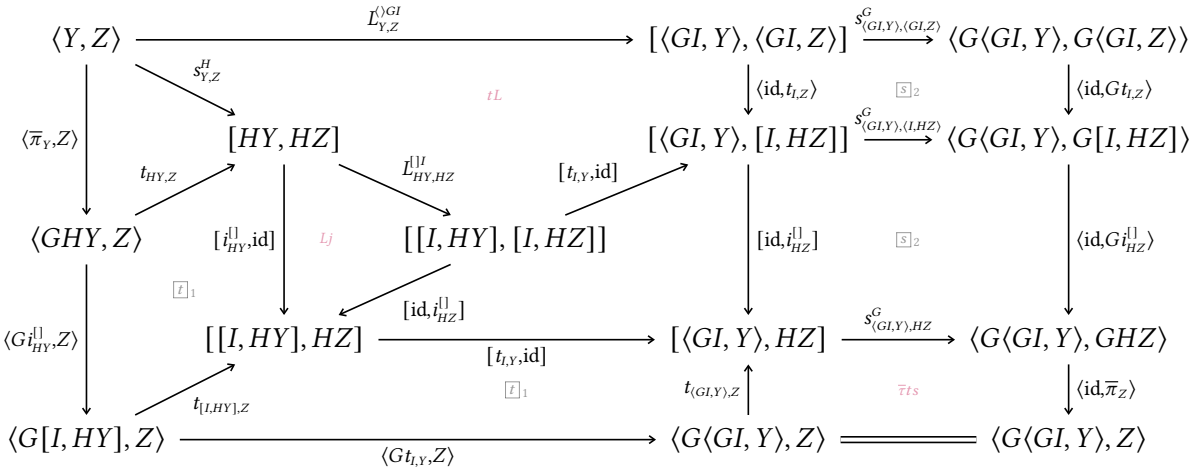
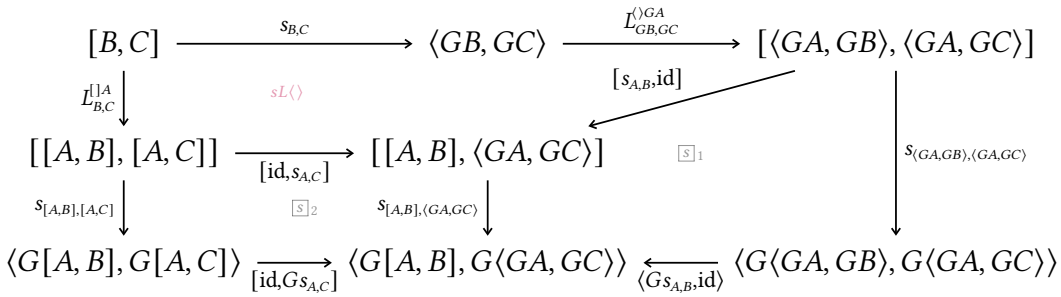


Diagram (*Ll*) is the G-application of:



Finally, Diagram (*Lq*) is simply the G-application of:



□

§ Proof of Theorem 7.2.1 on page 146

Theorem 7.2.1

If two unital endofunctors $(\Sigma\colon \mathcal{C} \to \mathcal{C}, \eta^\Sigma\colon \mathrm{Id} \Longrightarrow \Sigma)$ *and* $(\Omega\colon \mathcal{V} \to \mathcal{V}, \eta^\Omega\colon \mathrm{Id} \Longrightarrow \Omega)$ *form 1-cells and 2-cells in* **SynMod**

$$(\mathrm{id}, \eta^\Sigma)\colon (\mathrm{Id}, \mathrm{Id}) \Longrightarrow (\mathrm{Id}, \Sigma) : (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}}) \to (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}})$$
$$(\mathrm{id}, \eta^\Omega)\colon (\mathrm{Id}, \mathrm{Id}) \Longrightarrow (\mathrm{Id}, \Omega) : (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}}) \to (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}})$$

$(\boldsymbol{K}, G)\colon (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}}) \to (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}})$ *is a strong elevator between them, and* $\boldsymbol{L}\colon \mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}} \to \boldsymbol{\mathcal{V}}_{|\mathcal{V}}$ *is synthetic monoidal, then the categories of* Ω*-monoids in* $\mathcal{V}$ *and* Σ*-monoids in* $\mathcal{C}$ *are equivalent.*

Proof We first expand the assumptions. The 2-cells between synthetic module functors are natural transformations satisfying

$$\begin{array}{ccc} X \ominus \boldsymbol{Y} & \xrightarrow{g} & Z \\ {\scriptstyle \eta^\Sigma_X \ominus \boldsymbol{Y}}\downarrow & & \downarrow{\scriptstyle \eta^\Sigma_Z} \\ \Sigma X \ominus \boldsymbol{Y} & \xrightarrow[s^\Sigma[g]]{} & \Sigma Z \end{array} \qquad \begin{array}{ccc} A \oslash \boldsymbol{B} & \xrightarrow{h} & C \\ {\scriptstyle \eta^\Omega_A \oslash \boldsymbol{B}}\downarrow & & \downarrow{\scriptstyle \eta^\Omega_C} \\ \Omega A \oslash \boldsymbol{B} & \xrightarrow[s^\Omega[h]]{} & \Omega C \end{array}$$

The 1-cell $(\boldsymbol{K}, G)\colon (\boldsymbol{\mathcal{V}}_{|\mathcal{V}}, \mathcal{V}_{|\mathcal{V}}) \to (\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}, \mathcal{C}_{|\mathcal{C}})$ comes with a natural transformation of hom-sets $\mathcal{C}(X \ominus \boldsymbol{Y}, Z) \to \mathcal{V}(GX \oslash \boldsymbol{KY}, GZ)$ which, as $\underline{\boldsymbol{KY}}_{\mathcal{C}} = GY \in \mathcal{C}$, has components derived from the monoidal functor multiplication $m^G_{A,B}\colon GA \oplus GB \to G(A \otimes B)$. It is an elevator between the endofunctors, so satisfies the equivalent diagrams below for all $f\colon A \oslash \boldsymbol{B} \to C$:

$$\begin{array}{ccc} \Sigma GA \ominus \boldsymbol{KB} & \xrightarrow{s^\Sigma[s^{G,\boldsymbol{K}}[f]]} & \Sigma GC \\ \big| & & \big| \\ G\Omega A \ominus \boldsymbol{KB} & \xrightarrow[s^{G,\boldsymbol{K}}[s^\Sigma[f]]]{} & G\Omega C \end{array} \quad = \quad \begin{array}{ccc} \Sigma GA \oplus GB & \xrightarrow{s^\Sigma[m^G[f]]} & \Sigma GC \\ \big| & & \big| \\ G\Omega A \oplus GB & \xrightarrow[m^G[s^\Sigma[f]]]{} & G\Omega C \end{array}$$

Furthermore, as G is monoidal and $\underline{\boldsymbol{KA}}_{\mathcal{C}} = G\underline{A}_{\mathcal{V}}$, Proposition 6.1.1 applies to show that $\boldsymbol{K}\colon \boldsymbol{\mathcal{V}}_{|\mathcal{V}} \to \mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}$ is a synthetic monoidal functor, so the diagrams are further equivalent to:

$$\begin{array}{ccc} \underline{\boldsymbol{K\Omega A}}_{\mathcal{C}} \ominus \boldsymbol{KB} & \xrightarrow{m^{\boldsymbol{K}}[s^\Omega[f]]} & \underline{\boldsymbol{K\Omega C}}_{\mathcal{C}} \\ \big| & & \big| \\ \underline{\boldsymbol{\Sigma KA}}_{\mathcal{C}} \ominus \boldsymbol{KB} & \xrightarrow[s^\Sigma[m^{\boldsymbol{K}}[f]]]{} & \underline{\boldsymbol{\Sigma GC}}_{\mathcal{C}} \end{array} \tag{†}$$

Ω-monoid to Σ-monoid Instantiating Corollary 6.3.2 with Σ, Ω and $\boldsymbol{K}$ defined above, we get that a synthetic Ω-monoid $\boldsymbol{M}$ in $\boldsymbol{\mathcal{V}}_{|\mathcal{V}}$ maps to a synthetic Σ-monoid $\boldsymbol{KM}$ in $\mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}}$. The underlying object $\underline{\boldsymbol{M}}_{\mathcal{V}} = M$ is a Ω-monoid in $\mathcal{V}$, and so is $\underline{\boldsymbol{KM}}_{\mathcal{V}} = GM$ a Σ-monoid in $\mathcal{C}$.

Σ-monoid to Ω-monoid By assumption, $\boldsymbol{L}$ is a synthetic monoidal functor, so the monoid structure of the algebraic monoids is transformed as in Proposition 7.2.3. To apply Corollary 6.3.2, we need to furthermore show that $\boldsymbol{L}\colon \mathcal{C}^{\boldsymbol{T}}{}_{|\mathcal{C}} \to \boldsymbol{\mathcal{V}}_{|\mathcal{V}}$ lifts the synthetic module endo-

functor $\Sigma\colon \mathcal{C}^T{}_{|\mathcal{C}} \to \mathcal{C}^T{}_{|\mathcal{C}}$ to $\Omega\colon \mathcal{V}_{|\mathcal{V}} \to \mathcal{V}_{|\mathcal{V}}$ in **SynMod**. For this, we need to define an elevator $\boldsymbol{\psi}\colon \Omega\boldsymbol{L} \Longrightarrow \boldsymbol{L}\Sigma$ that satisfies, for all pointed multilinear maps $g \in \mathbf{PMLin}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{Z})$,

$$\begin{array}{ccc} \underline{\Omega\boldsymbol{L}\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{Y} & \xrightarrow{s^{\Omega}[m^{\boldsymbol{L}}[g]]} & \underline{\Omega\boldsymbol{L}\boldsymbol{X}}_{\mathcal{V}} \\ \downarrow{\scriptstyle \underline{\boldsymbol{\psi}_{\boldsymbol{X}}}\oslash \mathrm{id}} & & \downarrow{\scriptstyle \underline{\boldsymbol{\psi}_{\boldsymbol{Z}}}} \\ \underline{\boldsymbol{L}\Sigma\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{Y} & \xrightarrow[m^{\boldsymbol{L}}[s^{\Sigma}[g]]]{} & \underline{\boldsymbol{L}\Sigma\boldsymbol{X}}_{\mathcal{V}} \end{array} \tag{‡}$$

To define $\boldsymbol{\psi}\colon \Omega\boldsymbol{L} \Longrightarrow \boldsymbol{L}\Sigma$, we first construct an elevator $\boldsymbol{\varphi}\colon \boldsymbol{K}\Omega \Longrightarrow \Sigma\boldsymbol{K}\colon \mathcal{C}^{\boldsymbol{T}} \to \boldsymbol{\mathcal{V}}$ with components:

$$\underline{\boldsymbol{K}\Omega\boldsymbol{A}}_{\mathcal{C}} = G\Omega A \cong \Sigma G A = \underline{\Sigma\boldsymbol{K}\boldsymbol{A}}_{\mathcal{C}}$$

and strength-preservation given by the same property of the isomorphism $\Sigma G \cong G\Omega$. From this, $\boldsymbol{\psi}\colon \Omega\boldsymbol{L} \Longrightarrow \boldsymbol{L}\Sigma$ is given by:

$$\Omega\boldsymbol{L} \cong \boldsymbol{L}\boldsymbol{K}\Omega\boldsymbol{L} \xrightarrow{\boldsymbol{L}\boldsymbol{\varphi}\boldsymbol{L}} \boldsymbol{L}\Sigma\boldsymbol{K}\boldsymbol{L} \cong \boldsymbol{L}\Sigma$$

We can now instantiate (†) with $f \triangleq m^{\boldsymbol{L}}[g]\colon \underline{\boldsymbol{L}\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{Y} \to \underline{\boldsymbol{L}\boldsymbol{Z}}_{\mathcal{V}}$ for $g \in \mathbf{PMLin}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{Z})$:

$$\begin{array}{ccc} \underline{\boldsymbol{K}\Omega\boldsymbol{L}\boldsymbol{X}}_{\mathcal{C}} \ominus \boldsymbol{K}\boldsymbol{L}\boldsymbol{Y} & \xrightarrow{m^{\boldsymbol{K}}[s^{\Omega}[m^{\boldsymbol{L}}[g]]]} & \underline{\boldsymbol{K}\Omega\boldsymbol{L}\boldsymbol{Z}}_{\mathcal{C}} \\ \downarrow{\scriptstyle \underline{\boldsymbol{\varphi}_{\boldsymbol{L}\boldsymbol{X}}}\ominus \mathrm{id}} & & \downarrow{\scriptstyle \underline{\boldsymbol{\varphi}_{\boldsymbol{L}\boldsymbol{Z}}}} \\ \underline{\Sigma\boldsymbol{K}\boldsymbol{L}\boldsymbol{X}}_{\mathcal{C}} \ominus \boldsymbol{K}\boldsymbol{L}\boldsymbol{Y} & \xrightarrow[s^{\Sigma}[m^{\boldsymbol{K}}[m^{\boldsymbol{L}}[g]]]]{} & \underline{\Sigma\boldsymbol{K}\boldsymbol{L}\boldsymbol{Z}}_{\mathcal{C}} \end{array}$$

Applying the operator $m^{\boldsymbol{L}}[-]$ to the diagram, and extracting $\boldsymbol{\varphi}$ via naturality, we get:

$$\begin{array}{ccc} \underline{\boldsymbol{L}\boldsymbol{K}\Omega\boldsymbol{L}\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{K}\boldsymbol{L}\boldsymbol{Y} & \xrightarrow{m^{\boldsymbol{L}}[m^{\boldsymbol{K}}[s^{\Omega}[m^{\boldsymbol{L}}[g]]]]} & \underline{\boldsymbol{L}\boldsymbol{K}\Omega\boldsymbol{L}\boldsymbol{Z}}_{\mathcal{V}} \\ \downarrow{\scriptstyle \underline{\boldsymbol{L}\boldsymbol{\varphi}_{\boldsymbol{L}\boldsymbol{X}}}\oslash \mathrm{id}} & & \downarrow{\scriptstyle \underline{\boldsymbol{L}\boldsymbol{\varphi}_{\boldsymbol{L}\boldsymbol{Z}}}} \\ \underline{\boldsymbol{L}\Sigma\boldsymbol{K}\boldsymbol{L}\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{K}\boldsymbol{L}\boldsymbol{Y} & \xrightarrow[m^{\boldsymbol{L}}[s^{\Sigma}[m^{\boldsymbol{K}}[m^{\boldsymbol{L}}[g]]]]]{} & \underline{\boldsymbol{L}\Sigma\boldsymbol{K}\boldsymbol{L}\boldsymbol{Z}}_{\mathcal{V}} \end{array}$$

As $\boldsymbol{L}\boldsymbol{K}$ and $\boldsymbol{K}\boldsymbol{L}$ are naturally monoidally isomorphic to the identity, and $\boldsymbol{L}\boldsymbol{\varphi}_{\boldsymbol{L}\boldsymbol{X}} = \boldsymbol{\psi}_{\boldsymbol{X}}$ by definition, the diagram simplifies to

$$\begin{array}{ccc} \underline{\Omega\boldsymbol{L}\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{Y} & \xrightarrow{s^{\Omega}[m^{\boldsymbol{L}}[g]]} & \underline{\Omega\boldsymbol{L}\boldsymbol{Z}}_{\mathcal{V}} \\ \downarrow{\scriptstyle \underline{\boldsymbol{\psi}_{\boldsymbol{X}}}\oslash \mathrm{id}} & & \downarrow{\scriptstyle \underline{\boldsymbol{\psi}_{\boldsymbol{Z}}}} \\ \underline{\boldsymbol{L}\Sigma\boldsymbol{X}}_{\mathcal{V}} \oslash \boldsymbol{L}\boldsymbol{Y} & \xrightarrow[m^{\boldsymbol{L}}[s^{\Sigma}[g]]]{} & \underline{\boldsymbol{L}\Sigma\boldsymbol{Z}}_{\mathcal{V}} \end{array}$$

which is exactly the strength-preservation (‡) we sought. The functor $\boldsymbol{L}$ with natural transformation $\boldsymbol{\psi}$ is thus a lifting of Σ to Ω in the category **SynMod**, so Corollary 6.3.2 can be applied to deduce the corresponding mapping of synthetic Σ-monoids $\boldsymbol{N}$ in $\mathcal{C}^T{}_{|\mathcal{C}}$ to synthetic Ω-monoids $\boldsymbol{L}\boldsymbol{N}$ in $\mathcal{V}_{|\mathcal{V}}$. The underlying Σ-monoid $N \in \mathcal{C}$ is mapped to $L(N, n\colon N \oplus J \xrightarrow{N\oplus\eta} N \oplus N \xrightarrow{\mu} N)$,

an Ω-monoid in $\mathcal{V}$, with unit, multiplication, and Ω-algebra structure

$$u[\eta] : I \to L(N,n) \qquad \underline{m^L[\mu]}_{\mathcal{V}} : L(N,n) \otimes L(N,n) \to L(N,n)$$
$$\Omega(L(N,n)) = \underline{\boldsymbol{\Omega LN}}_{\mathcal{V}} \to \underline{\boldsymbol{L\Sigma N}}_{\mathcal{V}} \to \underline{\boldsymbol{LN}}_{\mathcal{V}} = L(N,n)$$

Invertibility Since the underlying functors of the mappings are the monoidally inverse functors $\boldsymbol{K}$ and $\boldsymbol{L}$, the round trips on objects are isomorphic. Similarly, the multiplication operations $m^L[-]$ and $m^K[-]$ compose to the isomorphism of sets $\mathcal{C}^T(\boldsymbol{J}, \boldsymbol{Z}) \cong \mathcal{C}^T(\boldsymbol{J}, \boldsymbol{KLZ})$ and $\mathbf{PMLin}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{Z}) \cong \mathbf{PRLin}(\boldsymbol{KLX}, \boldsymbol{KLY}; \boldsymbol{KLZ})$, and similarly for the other direction. Thus, the equivalence of categories $\Sigma\text{-}\mathbf{Mon}(\mathcal{C}) \simeq \Omega\text{-}\mathbf{Mon}(\mathcal{V})$ is established.

□

---

§ Proof of Theorem 10.3.1 on page 201

Theorem 10.3.1

For all $W \in \mathbf{Fam}$, the lifted functor $(-)^W : \mathcal{I}/\mathbf{Mod}_S \to \mathcal{I}/\mathbf{Mod}_S$ is a synthetic monoidal endofunctor on $\mathcal{I}/\mathbf{Mod}_S$.

Proof The unit transformation $u : \mathcal{I} \to \mathcal{I}^W$ is the component $\kappa^W_{\mathcal{I}}$. The multiplication operator is defined as:

$$m[-] : \mathbf{PMLin}(\mathcal{X}, \mathcal{Y}; \mathcal{Z}) \to \mathbf{PMLin}(\mathcal{X}^W, \mathcal{Y}^W; \mathcal{Z}^W)$$
$$m[f : \mathcal{X} \oplus \mathcal{Y} \to \mathcal{Z}](\Gamma, l \in \mathcal{X}^W(\Gamma), \omega : {}^{\Gamma}(\mathcal{Y}^W)_{\Delta})_{\Theta}\ (w \in W(\Theta)) \triangleq f\big(\Gamma + \Theta, l\, w, \omega\lfloor w \rfloor \rtimes \iota_1^{\Delta,\Theta}\big)$$

Explicitly, $l\, x \in \mathcal{X}(\Gamma + \Theta)$ has to be associated with a substitution rule ${}^{\Gamma+\Theta}\mathcal{Y}_{\Delta+\Theta}$, constructed as the right widening of the substitution and renaming rules:

$$\mathcal{I}_\alpha\Gamma \xrightarrow{\omega} (\mathcal{Y}^W)_\alpha(\Delta) \xrightarrow{-w} \mathcal{Y}_\alpha(\Delta + \Theta) \qquad \mathcal{I}_\alpha\Theta \xrightarrow{\iota_1^{\Delta,\Theta}} \mathcal{I}_\alpha(\Delta + \Theta)$$

With the notational conventions for partially applied parametrised maps, the definition is:

$$m[f]\{\omega\}\lfloor w \rfloor(l) \triangleq f\{\omega\lfloor w \rfloor \rtimes \mathrm{inr}\}(l\, w)$$

We next show the synthetic monoidal axioms.

- For all pointed multilinear maps $g : \mathcal{X} \oplus \mathcal{Y} \to \mathcal{Z}$ and pointed module homomorphisms $h : \mathcal{Z} \to \mathcal{Y}$, if the diagram on the left commutes, so must the diagram on the right:

$$\begin{array}{ccc} \mathcal{I} \oplus \mathcal{Y} & \xrightarrow{\eta_{\mathcal{X}} \oplus \mathcal{Y}} & \mathcal{X} \oplus \mathcal{Y} \\ {\scriptstyle \lambda_{\mathcal{Y}}}\downarrow & & \downarrow{\scriptstyle g} \\ \mathcal{Y} & \xleftarrow[h]{} & \mathcal{Z} \end{array} \qquad \begin{array}{ccccc} \mathcal{I} \oplus \mathcal{Y}^W & \xrightarrow{\eta_{\mathcal{X}} \oplus \mathrm{id}} & \mathcal{X} \oplus \mathcal{Y}^W & \xrightarrow{\kappa^W_{\mathcal{X}} \oplus \mathrm{id}} & \mathcal{X}^W \oplus \mathcal{Y}^W \\ {\scriptstyle \lambda_{\mathcal{Y}^W}}\downarrow & & & & \downarrow{\scriptstyle m[g]} \\ \mathcal{Y}^W & & \xleftarrow[h^W]{} & & \mathcal{Z}^W \end{array}$$

Assume that for all $v \in \mathcal{I}_\alpha\Gamma, \sigma \in {}^{\Gamma}\mathcal{Y}_\Delta$, ① $h(g\{\sigma\}(\eta\, v)) = \sigma\, v$. Then, for all $u \in \mathcal{I}_\alpha\Gamma, \omega \in {}^{\Gamma}(\mathcal{Y}^W)_\Delta$ and $w \in W(\Theta)$, we have:

$$\begin{aligned}
&h^W\lfloor w\rfloor\big(m[g]((\kappa \oplus \mathrm{id})(\eta\, u, \omega))\big)\\
&= h\big(m[g]\{\omega\}\lfloor w\rfloor(\kappa(\eta\, u))\big)\\
&= h\big(g\{\omega\lfloor w\rfloor \rtimes \mathrm{inr}\}(\kappa\,(\eta\, u)\, w)\big) && (m[g] \triangleq)\\
&= h\big(g\{\omega\lfloor w\rfloor \rtimes \mathrm{inr}\}(\mathcal{X}\langle\mathrm{inl}\rangle(\eta\, u))\big) && (\kappa \triangleq)\\
&= h\big(g\{(\omega\lfloor w\rfloor \rtimes \mathrm{inr}) \circ \mathrm{inl}\}(\eta\, u)\big) && (g \text{ right linear})\\
&= h\big(g\{\omega\lfloor w\rfloor\}(\eta\, u)\big) && (\text{widening property})\\
&= \omega\lfloor w\rfloor u && (① \text{ with } \sigma \triangleq \omega\lfloor w\rfloor)
\end{aligned}$$

- For all pointed multilinear maps $g\colon \mathcal{X}\oplus\mathcal{Y} \to \mathcal{Z}$ and pointed module homomorphisms $h\colon \mathcal{X} \to \mathcal{Z}$, if the diagram on the left commutes, so must the diagram on the right:

$$\begin{array}{ccc}
\mathcal{X} & \xrightarrow{\;h\;} & \mathcal{Z}\\
{\scriptstyle\rho_{\mathcal{X}}}\downarrow & & \uparrow{\scriptstyle g}\\
\mathcal{X}\oplus\mathcal{I} & \xrightarrow[\mathcal{X}\oplus\eta_{\mathcal{Y}}]{} & \mathcal{X}\oplus\mathcal{Y}
\end{array}
\qquad\qquad
\begin{array}{ccccc}
\mathcal{X}^W & & \xrightarrow{\;h^W\;} & & \mathcal{Z}^W\\
{\scriptstyle\rho_{\mathcal{X}^W}}\downarrow & & & & \uparrow{\scriptstyle m[g]}\\
\mathcal{X}^W\oplus\mathcal{I} & \xrightarrow[\mathrm{id}\oplus\eta_{\mathcal{Y}}]{} & \mathcal{X}^W\oplus\mathcal{Y} & \xrightarrow[\mathrm{id}\oplus\kappa^W_{\mathcal{Y}}]{} & \mathcal{X}^W\oplus\mathcal{Y}^W
\end{array}$$

Assume that for all $t \in \mathcal{X}_\alpha\Gamma$, $g((\mathcal{X}\oplus\eta_{\mathcal{Y}})(\rho_{\mathcal{X}}\, t)) = h\, t$, or equivalently, ② $g\{\eta\}\, t = h\, t$. Then, for all $l \in (\mathcal{X}^W)_\alpha\Gamma$ and $w \in W(\Theta)$, we have:

$$\begin{aligned}
&m[g]\{\kappa \circ \eta\}\lfloor w\rfloor(l)\\
&= g\{(\kappa \circ \eta)\lfloor w\rfloor \rtimes \mathrm{inr}\}(l\, w) && (m[g] \triangleq)\\
&= g\{(\mathcal{X}\langle\mathrm{inl}\rangle \circ \eta) \rtimes \mathrm{inr}\}(l\, w) && (\kappa \triangleq)\\
&= g\{(\eta \circ \mathrm{inl}) \rtimes \mathrm{inr}\}(l\, w) && (\eta \text{ naturality})\\
&= g\{\eta\}(l\, w) && (\text{widening property})\\
&= h(l\, w) && (② \text{ with } t \triangleq l\, w)\\
&= h^W\lfloor w\rfloor l
\end{aligned}$$

- For all pointed multilinear maps $e\colon \mathcal{U}\oplus\mathcal{V} \to \mathcal{X}$, $f\colon \mathcal{V}\oplus\mathcal{W} \to \mathcal{Y}$, $g\colon \mathcal{X}\oplus\mathcal{W} \to \mathcal{Z}$, $h\colon \mathcal{U}\oplus\mathcal{Y} \to \mathcal{Z}$, if the diagram on the top commutes, so must the diagram on the bottom:

$$\begin{array}{ccccc}
(\mathcal{U}\oplus\mathcal{V})\oplus\mathcal{W} & & \xrightarrow{\;\alpha_{\mathcal{U},\mathcal{V},\mathcal{W}}\;} & & \mathcal{U}\oplus(\mathcal{V}\oplus\mathcal{W})\\
{\scriptstyle e\oplus\mathcal{W}}\downarrow & & & & \downarrow{\scriptstyle \mathcal{U}\oplus f}\\
\mathcal{X}\oplus\mathcal{W} & \xrightarrow[g]{} & \mathcal{Z} & \xleftarrow[h]{} & \mathcal{U}\oplus\mathcal{Y}
\end{array}$$

$$\begin{array}{ccccc}
(\mathcal{U}^W\oplus\mathcal{V}^W)\oplus\mathcal{W}^W & & \xrightarrow{\;\alpha_{\mathcal{U}^W,\mathcal{V}^W,\mathcal{W}^W}\;} & & \mathcal{U}^W\oplus(\mathcal{V}^W\oplus\mathcal{W}^W)\\
{\scriptstyle m[e]\oplus\mathrm{id}}\downarrow & & & & \downarrow{\scriptstyle \mathrm{id}\oplus m[f]}\\
\mathcal{X}^W\oplus\mathcal{W}^W & \xrightarrow[m[g]]{} & \mathcal{Z}^W & \xleftarrow[m[h]]{} & \mathcal{U}^W\oplus\mathcal{Y}^W
\end{array}$$

Assume that for all $t \in \mathcal{U}_\alpha\Gamma$ and substitution rules $\sigma \in {}^{\Gamma}\mathcal{V}_{\Delta}, \varsigma \in {}^{\Delta}\mathcal{W}_{\Theta}$, we have the top diagram identity $g(e \oplus \mathcal{V})((t,\sigma),\varsigma) = h((\mathcal{U} \oplus f)(\alpha((t,\sigma),\varsigma)))$ or equivalently, ③ $g\{\varsigma\}(e\{\sigma\}t) = h\{f\{\varsigma\} \circ \sigma\}t$. We show that for all $l \in (\mathcal{U}^W)_\alpha(\Gamma)$, $\omega \in {}^{\Gamma}(\mathcal{V}^W)_{\Delta}$, $\varpi \in {}^{\Delta}(\mathcal{W}^W)_{\Theta}$ and $w \in W(\Xi)$, we have:

$$
\begin{aligned}
& m[g]\{\varpi\}\lfloor w\rfloor\big(m[e]\{\omega\}\,l\big) \\
&= g\{\varpi\lfloor w\rfloor \rtimes \mathrm{inr}\}(m[e]\{\omega\}\lfloor w\rfloor\, l) && (m[g]\triangleq) \\
&= g\{\varpi\lfloor w\rfloor \rtimes \mathrm{inr}\}(e\{\omega\lfloor w\rfloor \rtimes \mathrm{inr}\}(l\,w)) && (m[e]\triangleq) \\
&= h\{f\{\varpi\lfloor w\rfloor \rtimes \mathrm{inr}\} \circ (\omega\lfloor w\rfloor \rtimes \mathrm{inr})\}(l\,w) && (③) \\
&= h\{(f\{\varpi\lfloor w\rfloor \rtimes \mathrm{inr}\} \circ \omega\lfloor w\rfloor) \rtimes \mathrm{inr}\}(l\,w) && (\text{Lem. } 10.3.2) \\
&= h\{(m[f]\{\varpi\} \circ \omega)\lfloor w\rfloor \rtimes \mathrm{inr}\}(l\,w) && (m[f]\triangleq) \\
&= m[h]\{m[f]\{\varpi\} \circ \omega\}\lfloor w\rfloor\, l && (m[h]\triangleq)
\end{aligned}
$$

where the application of Lemma 10.3.2 is preconditioned on $(\varpi\lfloor w\rfloor \rtimes \iota_1^{\Theta,\Xi}) \circ \iota_1^{\Delta,\Xi} = \eta_W \circ \iota_1^{\Theta,\Xi}$, which follows by a reduction axiom of widening.

Thus, we conclude that $(-)^W$ is a synthetic monoidal functor on $\mathcal{I}/\mathbf{Mod}_S$. □

---

§ Proof of Proposition 10.3.10 on page 205

Proposition 10.3.10 *For $U, W \in \mathbf{Fam}$, there is an elevator in* **SynMod** *from the $(U \bullet\!\!-)$-relative $\mathcal{I}/\mathbf{Mod}_S$-module endofunctor $(U \circ\!\!-) : \mathbf{Fam} \to \mathbf{Fam}$ to the $(W \multimap\!\bullet)$-relative $\mathcal{I}/\mathbf{Mod}_S$-module endofunctor $(W \multimap\!\circ) : \mathbf{Fam} \to \mathbf{Fam}$.*

Proof The underlying elevator from $((U \circ\!\!-), (U \bullet\!\!-))$ to $((W \multimap\!\circ), (W \multimap\!\bullet))$ consists of a synthetic monoidal transformation $e_W^{U\bullet-} : U \bullet\!\!- (W \multimap\!\bullet (-)) \Longrightarrow W \multimap\!\bullet (U \bullet\!\!- (-))$ and a synthetic module transformation $e_W^{U\circ-} : U \circ\!\!- (W \multimap\!\circ (-)) \Longrightarrow W \multimap\!\circ (U \circ\!\!- (-))$. The multiplication-preservation of the former implies that of the latter, similarly to how the strength laws of Proposition 10.3.8 derive from the monoidal laws of Theorem 10.3.1.

We therefore show that the exchange preserves the unit and multiplication operators associated with the composite near-genuine functors $U \bullet\!\!- (W \multimap\!\bullet (-))$ and $W \multimap\!\bullet (U \bullet\!\!- (-))$. For clarity we will write κ and $m[-]$ for the unit and multiplication of $(W \multimap\!\bullet)$, and $\varkappa$ and $n[-]$ for those of $(U \bullet\!\!-)$, using exponentiation notation with the hom direction implicit. The laws instantiate to the following, with $f : \mathcal{X} \oplus \mathcal{Y} \to \mathcal{Z}$ a pointed multilinear map:

$$
\begin{array}{ccccc}
\mathcal{I} & \xrightarrow{\varkappa_{\mathcal{I}}} & \mathcal{I}^U & \xrightarrow{(\kappa_{\mathcal{I}})^U} & (\mathcal{I}^W)^U \\
& \searrow^{\kappa_{\mathcal{I}}} & & & \downarrow{\scriptstyle e_{\mathcal{I}}^{U,W}} \\
& & \mathcal{I}^W & \xrightarrow[(\varkappa_{\mathcal{I}})^W]{} & (\mathcal{I}^U)^W
\end{array}
\qquad
\begin{array}{ccc}
(\mathcal{X}^W)^U \oplus (\mathcal{Y}^W)^U & \xrightarrow{n[m[f]]} & (\mathcal{Z}^W)^U \\
{\scriptstyle e_{\mathcal{X}}^{U,W} \oplus e_{\mathcal{Y}}^{U,W}}\downarrow & & \downarrow{\scriptstyle e_{\mathcal{Z}}^{U,W}} \\
(\mathcal{X}^U)^W \oplus (\mathcal{Y}^U)^W & \xrightarrow[m[n[f]]]{} & (\mathcal{Z}^U)^W
\end{array}
$$

Taking $v \in \mathcal{I}_\alpha\Gamma$, $w : W(\Delta)$ and $u : U(\Theta)$, the unit law reduces as follows:

$$
\begin{aligned}
& e\,(\kappa^U(\varkappa\, v))\,w\,u \\
&= \alpha_{\Theta,\Gamma,\Delta}((\kappa^U\,(\varkappa\, v))\,u\,w) && (e\triangleq)
\end{aligned}
$$

$$
\begin{aligned}
&= \alpha_{\Theta,\Gamma,\Delta}(\kappa(\varkappa\, v\, u)\, w) \\
&= \alpha_{\Theta,\Gamma,\Delta}(\iota_2^{\Theta+\Gamma,\Delta}(\iota_1^{\Theta,\Gamma}\, v)) && (\kappa, \varkappa \triangleq) \\
&= \iota_1^{\Theta,\Gamma+\Delta}(\iota_2^{\Gamma,\Delta}\, v) && \text{(monoidal coherence)} \\
&= \varkappa(\kappa\, v\, w)\, u && (\kappa, \varkappa \triangleq) \\
&= \varkappa^W(\kappa\, v)\, w\, u
\end{aligned}
$$

For the multiplication, we take $l \in ((\mathcal{X}^W)^U)_\alpha\Gamma$, $\omega \in {}^\Gamma\big((\mathcal{Y}^W)^U\big)_\Delta$, $w\colon W(\Theta)$, $u\colon U(\Xi)$, and a pointed multilinear map $f\colon \mathcal{X} \multimap \boxed{\mathcal{Y}} \to \mathcal{Z}$:

$$
\begin{aligned}
& e\,(n[m[f]]\{\omega\}\, l)\, w\, u \\
&= \mathcal{Z}\langle\alpha_{\Xi,\Delta,\Theta}\rangle\big(n[m[f]]\{\omega\}\lfloor u\rfloor\, l\, w\big) && (e \triangleq) \\
&= \mathcal{Z}\langle\alpha_{\Xi,\Delta,\Theta}\rangle\big(m[f]\{\iota_2^{\Xi,\Delta} \ltimes \omega\lfloor u\rfloor\}\lfloor w\rfloor(l\, u)\big) && (n[m[f]] \triangleq) \\
&= \mathcal{Z}\langle\alpha_{\Xi,\Delta,\Theta}\rangle\big(f\{(\iota_2^{\Xi,\Delta} \ltimes \omega\lfloor u\rfloor)\lfloor w\rfloor \rtimes \iota_1^{\Xi+\Delta,\Theta}\}(l\, u\, w)\big) && (m[f] \triangleq) \\
&= f\{\mathcal{Y}\langle\alpha_{\Xi,\Delta,\Theta}\rangle \circ ((\iota_2^{\Xi,\Delta} \ltimes \omega\lfloor u\rfloor)\lfloor w\rfloor \rtimes \iota_1^{\Xi+\Delta,\Theta})\}(l\, u\, w) && (f \text{ multilinear}) \\
&= f\{\big(\iota_2^{\Xi,\Delta+\Theta} \ltimes ((e \circ \omega)\lfloor w\rfloor \rtimes \iota_1^{\Delta,\Theta})\lfloor u\rfloor\big) \circ \alpha_{\Xi,\Gamma,\Theta}\}(l\, u\, w) && (\dagger) \\
&= f\{\iota_2^{\Xi,\Delta+\Theta} \ltimes ((e \circ \omega)\lfloor w\rfloor \rtimes \iota_1^{\Delta,\Theta})\lfloor u\rfloor\}(\mathcal{X}\langle\alpha_{\Xi,\Gamma,\Theta}\rangle(l\, u\, w)) && (f \text{ multilinear}) \\
&= f\{\iota_2^{\Xi,\Delta+\Theta} \ltimes ((e \circ \omega)\lfloor w\rfloor \rtimes \iota_1^{\Delta,\Theta})\lfloor u\rfloor\}(e\, l\, w\, u) && (e \triangleq) \\
&= n[f]\{(e \circ \omega)\lfloor w\rfloor \rtimes \iota_1^{\Delta,\Theta}\}(e\, l\, w)\, u && (n[f] \triangleq) \\
&= m[n[f]]\{e \circ \omega\}\lfloor w\rfloor\, (e\, l)\, u && (m[n[f]] \triangleq)
\end{aligned}
$$

where the inner equality of widenings at † computes as follows:

$$
\begin{aligned}
& \mathcal{Y}\langle\alpha_{\Xi,\Delta,\Theta}\rangle \circ ((\iota_2^{\Xi,\Delta} \ltimes \omega\lfloor u\rfloor)\lfloor w\rfloor \rtimes \iota_1^{\Xi+\Delta,\Theta}) \\
&= \mathcal{Y}\langle\alpha_{\Xi,\Delta,\Theta}\rangle \circ (((\iota_2^{\Xi+\Delta,\Theta} \circ \iota_2^{\Xi,\Delta}) \ltimes \omega\lfloor u\rfloor\lfloor w\rfloor) \rtimes \iota_1^{\Xi+\Delta,\Theta}) && \text{(Lem. 10.3.1)} \\
&= ((\alpha_{\Xi,\Delta,\Theta} \circ \iota_2^{\Xi+\Delta,\Theta} \circ \iota_2^{\Xi,\Delta}) \ltimes (\mathcal{Y}\langle\alpha_{\Xi,\Delta,\Theta}\rangle \circ \omega\lfloor u\rfloor\lfloor w\rfloor)) \rtimes (\alpha_{\Xi,\Delta,\Theta} \circ \iota_1^{\Xi+\Delta,\Theta}) && \text{(widening lemma)} \\
&= ((\iota_2^{\Xi,\Delta+\Theta} \ltimes (\mathcal{Y}\langle\alpha_{\Xi,\Delta,\Theta}\rangle \circ \omega\lfloor u\rfloor\lfloor w\rfloor)) \rtimes (\iota_1^{\Xi,\Delta+\Theta} \circ \iota_1^{\Delta,\Theta})) && \text{(cocartesian coherence)} \\
&= \big(\iota_2^{\Xi,\Delta+\Theta} \ltimes ((\mathcal{Y}\langle\alpha_{\Xi,\Delta,\Theta}\rangle \circ \omega\lfloor u\rfloor\lfloor w\rfloor) \rtimes (\iota_1^{\Xi,\Delta+\Theta} \circ \iota_1^{\Delta,\Theta}))\big) \circ \alpha_{\Xi,\Gamma,\Theta} && \text{(reassociation)} \\
&= \big(\iota_2^{\Xi,\Delta+\Theta} \ltimes ((e \circ \omega)\lfloor w\rfloor\lfloor u\rfloor \rtimes \iota_1^{\Xi,\Delta+\Theta} \circ \iota_1^{\Delta,\Theta})\big) \circ \alpha_{\Xi,\Gamma,\Theta} && (e \triangleq) \\
&= \big(\iota_2^{\Xi,\Delta+\Theta} \ltimes ((e \circ \omega)\lfloor w\rfloor \rtimes \iota_1^{\Delta,\Theta})\lfloor u\rfloor\big) \circ \alpha_{\Xi,\Gamma,\Theta} && \text{(Lem. 10.3.1)}
\end{aligned}
$$

□