Backpropagation for Effectful Languages I: Finite Probability and Discrete Output Algebraic Effects

Diogo Simm
Utrecht University, The Netherlands
d.l.simmsallesvianna@uu.nl

Fernando Lucatelli Nunes
Centre for Mathematics, University of Coimbra, Portugal
fln@uc.pt

Matthijs Vákár
Utrecht University, The Netherlands
m.i.l.vakar@uu.nl

Abstract

We analyse *reverse-mode* automatic differentiation (AD) for discrete probabilistic programs. Our construction is formulated in the framework of Combinatory Homomorphic Automatic Differentiation (CHAD), treating AD as a structure-preserving transformation of programs, guided by a denotational semantics.

The main case study is the finite atomic distribution monad, whose computations have finite support and differentiable weights. The key point is that differentiating probabilistic programs requires cotangents to flow backwards not only through deterministic computations, but also through the probabilistic structure itself. We define the corresponding reverse-mode code transformation and prove its correctness, for handled real-output programs, by a categorical logical-relations argument.

Although the paper focuses on finite discrete probability, the construction gives a reusable pattern for differentiating discrete-output algebraic effects, including finite multiset non-determinism (e.g., from fork-join parallelism), exceptions, and writer-style accumulation (e.g., for in-place accumulation of high-dimensional vectors). More broadly, we view this work as a foundational step towards extending CHAD to richer probabilistic languages and to other algebraic effects with handlers.

Keywords. reverse-mode automatic differentiation, probabilistic programming, algebraic effects, categorical semantics.

1 Introduction

Probabilistic programming languages provide constructs for specifying probabilistic models directly and for manipulating their statistical properties within the language. Their promise is to make common tasks in probabilistic data analysis, such as sampling, inference, and the optimisation of expected objectives, more compositional and closer to the structure of the models themselves.

Reverse-mode automatic differentiation (AD), often called backpropagation, underlies much of modern optimisation and machine learning. Its practical importance comes from the need to compute gradients of scalar or low-dimensional objectives with respect to large numbers of parameters Baydin et al. (2018). Compared to the simpler forward-mode of AD, reverse-mode AD inverts data flow: forward-mode propagates sensitivities along the computation, whereas reverse-mode propagates and accumulates cotangents against the direction of computation. This makes the design and semantics of reverse-mode automatic differentiation substantially more delicate.

Combinatory Homomorphic Automatic Differentiation (CHAD) gives a particularly clean categorical account of reverse-mode AD. In the CHAD framework, automatic differentiation is not specified as an *ad hoc* syntactic traversal, but as a semantics-driven program transformation: a structure-preserving map out of the syntactic categorical model of the source language Lucatelli Nunes and Vákár (2023); Lucatelli Nunes et al. (2025). This account is not merely explanatory, but it guides us in how to differentiate more complex forms of computation. Once the relevant semantic structure has been identified, the corresponding program transformation is obtained by following the universal structure of the language. In the deterministic setting, this has led both to correctness proofs by categorical logical relations and to efficient implementations via familiar functional-programming techniques, such as sparse representations and defunctionalisation or closure conversion Smeding and Vákár (2024).

The goal of the present paper is to explain how the reverse-mode CHAD transformation applies to programs with finite-support probabilistic effects and differentiable weights. In this impure setting, backpropagation is not only a matter of propagating cotangents through the deterministic input-output behaviour of a program. The weights of probabilistic traces, and more generally the effectful structure of computations, become part of the reverse derivative communication channel. We formulate this using monadic types and handlers: monadic operations generate probabilistic computations, while handlers consume them, for instance by taking expected values.

There is already foundational work on forward-mode automatic differentiation for probabilistic programming, notably ADEV Lew et al. (2023). The present paper addresses the corresponding reverse-mode problem. The semantic challenges of the two modes are different: forward-mode propagates sensitivities with the computation, whereas reverse mode must accumulate cotangent contributions backwards through both ordinary program structure and probabilistic weights. Our contribution is to give a compositional reverse-mode transformation for this finite probabilistic setting, formulated using monadic types and handlers, and to prove its correctness by a categorical logical-relations argument. The CHAD framework is essential here: it derives the transformation from the semantic structure rather than retrofitting a correctness proof after the fact, while keeping the construction close to implementation.

1.1 Motivating example: expected-profit optimisation under finite probabilistic choice

To make the problem concrete, we begin with a small finite probabilistic optimisation example. The model is deliberately simple, but it contains the features that drive the paper: parameter-dependent probabilistic weights, deterministic arithmetic, and a handler that consumes a probabilistic computation by taking an expected value. Reverse-mode AD for such a program must propagate cotangents through all of these components, not only through the deterministic arithmetic. We first describe the model mathematically, and later return to its encoding in the source language and to the corresponding code transformation (Section 5).

We consider the problem of introducing a product to a population. To obtain a finite probabilistic model, we make the simplifying assumption that the population is highly trend-following. There are then two possible outcomes: either the trend does not catch on and no one buys the product, or it catches on and everyone who can afford the product buys one.

The parameters of the model are the per-product profit p and the marketing budget m. Marketing influences the probability of the successful-trend scenario, while the per-product profit determines how many people can purchase the product if that scenario occurs.

In symbols, the probability of zero total sales is

$$\mathrm{sig}(-s_m \cdot (m - m_0))\,,$$

where $\mathrm{sig}(x) = (1 + e^{-x})^{-1}$ is the sigmoid function and $s_m, m_0 \in \mathbb{R}$ are constants: s_m represents the model's sensitivity to marketing, while m_0 is the marketing budget at which the success and failure scenarios are equally likely. If, instead, the campaign is successful, the number of purchases is

$$P \cdot \mathrm{sig}(-s_p \cdot (p - p_0))\,,$$

where P is the total population, s_p is a known constant controlling the sensitivity to changes in p, and p_0 is the per-product profit at which the product is affordable to half the population. Thus, by reducing the per-product profit, a larger portion of the population is able to afford the product.

The expected total profit, which we intend to maximise, is therefore

$$\mathrm{ETP}(p, m) = p \cdot \underbrace{P \cdot \mathrm{sig}(-s_p \cdot (p - p_0)) \cdot \mathrm{sig}(s_m \cdot (m - m_0))}_{\text{expected number of sales}} - m.$$

The stationarity conditions obtained by differentiating this expected objective are a pair of coupled transcendental equations. Thus, even in this simple example, optimisation is naturally performed numerically, for instance by gradient descent.

1.2 Contributions and Outline

The example above illustrates the central semantic issue addressed in this paper: probabilistic programs carry differentiable information not only through their deterministic computations, but also through the effectful structure by which probabilistic computations are generated and handled. The body of the paper addresses this issue as follows.

First, we give a factorisation-based account of the finite atomic distribution (FAD) monad. In **Set**, a FAD is a finitely supported nonnegative weight function; we reconstruct it by factorising the weighted-sum strong monad morphism induced by the barycentre algebra on lists of point/log-weight pairs (Section 2.2). This is the bridge to the differentiated semantics: in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, a distribution w on a space X with cotangent spaces $A(x)$ has fibre

$$\bigoplus_{x \in \mathrm{supp}(w)} (A(x) \oplus \mathbb{R}),$$

whose summands record cotangents for the atom x and for its log-weight. Thus the lifted FAD monad captures movement of atoms and variation of weights (Sections 2.3 and 2.4).

Second, we use this lift to extend reverse-mode CHAD to a first-order probabilistic language with products, coproducts, Euclidean base types $\mathbb{R}^n$, differentiable primitives, the monadic type constructor M, categorical distributions, and expectation-like handler algebras. Its ordinary semantics lives in **Set**, its differentiated semantics in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and the transformation is the induced structure-preserving translation into a target language for derivatives.

Third, we prove correctness by logical relations. We construct a category LR relating the ordinary **Set**-semantics of source programs to the $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$-semantics of their differentiated programs. Its projections recover, respectively, the ordinary source denotation and the CHAD denotation. The resulting correctness theorem (Theorem 4.1) says that, for real-input/real-output terms, the transformed program computes the original value and its reverse derivative.

Fourth, we encode the motivating example in the source language using categorical distributions, monadic composition, and an expectation handler. The transformed program exhibits the intended cotangent flow through deterministic arithmetic, atom locations, and log-weights.

Finally, although the paper focuses on finite discrete probability, the construction also identifies a reusable pattern for discrete-output monadic effects. Section 6 explains this general pattern, and Appendix H spells it out for finite multiset non-determinism, exceptions, and $\mathbb{R}^n$-writer accumulation. As such, this paper makes important steps towards both richer probabilistic-differentiable languages and towards general differentiable programming with algebraic effects and handlers.

2 The Finite Atomic Distribution Monad

The probabilistic effect studied in the main body of the paper is represented by the *finite atomic distribution* (*FAD*) monad. For a set X, its elements are finitely supported, nonnegative weight distributions on X. Operationally, this is the monad of finite probabilistic computations: monadic composition sums the weights of finite execution traces, while handlers, such as expectation, consume the resulting weighted distribution. For reverse-mode CHAD, however, the ordinary set-level monad is not enough. We need a lifted version that also records how cotangents propagate through the probabilistic structure itself, including both the locations of atoms and their weights.

This section develops the FAD monad in the form needed later in the paper. We begin in **Set**, recalling its familiar description as the monad of finitely supported nonnegative weight functions. We then give an equivalent factorisation presentation in **Set**: lists of point/log-weight pairs serve as raw weighted syntax, their weighted-sum semantics gives a strong monad morphism, and factorising this morphism identifies precisely the presentations that determine the same finite atomic distribution. The following subsection packages these data as a FAD factorisation structure. This structure is then instantiated in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, where the associated FAD monad carries cotangent information for atoms and log-weights, and in LR, where the same construction is used in the logical-relations correctness argument.

2.1 FAD Monad on Set

We first recall the direct set-level description of the FAD monad. This is the familiar finite-support distribution monad on **Set**, used here in its unnormalised weighted form; it appears both in the probabilistic-programming literature Erwig and Kollmansberger (2006) and in implementations Erwig and Kollmansberger (2025).

We use standard categorical notation: products and coproducts are written 1, $X_1 \times X_2$, $\prod_{i\in I} X_i$, and 0, $X_1 \amalg X_2$, $\coprod_{i\in I} X_i$, with projections π_i, inclusions ι_i, tuplings $\langle f_i\rangle_{i\in I}$, and cotuplings $[f_i]_{i\in I}$. Exponentials are denoted by $[X, Y]$ or by Y^X.

A finite atomic distribution (FAD) on a set X is a finitely supported nonnegative function $w : X \to \mathbb{R}$. Explicitly, we define a FAD as the following notion of *unnormalised weight distribution*:

$$w(x) \geq 0 \text{ for all } x \in X, \qquad |\operatorname{supp} w| < \infty, \qquad \operatorname{supp} w := \{x \in X \mid w(x) \neq 0\}.$$

The set $\operatorname{supp} w$ is the *support* of w; its elements are the *atoms* of the distribution, and $w(x)$ is the *weight* of the atom x. We write $M(X)$ for the set of finite atomic distributions on X.

The functorial action pushes weights forward. For $f : X \to Y$, define $M(f) : M(X) \to M(Y)$ by

$$M(f)(w)\,(y) := \sum_{\substack{x\in\operatorname{supp} w\\ f(x)=y}} w(x)\,.$$

The unit $(\eta)_X : X \to M(X)$ sends each point to its Dirac distribution, that is, $(\eta)_X(x)\,(x') = 1$ if $x = x'$, and $(\eta)_X(x)\,(x') = 0$ otherwise. The multiplication $(\mu)_X : M^2(X) \to M(X)$, where $M^2 := M \circ M$, flattens a FAD of FADs by summing all contributions to each point:

$$(\mu)_X(\omega)\,(x) := \sum_{\substack{w\in\operatorname{supp}\omega\\ x\in\operatorname{supp} w}} \omega(w)\cdot w(x)\,.$$

The maps $\eta : \mathrm{Id}_{\mathbf{Set}} \to M$ and $\mu : M^2 \to M$ are natural and equip M with a monad structure.

This completes the direct description of the FAD monad on **Set**. We next give a second presentation of the same monad, not by changing the object being defined, but by changing how it is obtained: finite atomic distributions are recovered as the image of a weighted-sum map from ordered point/log-weight presentations. This factorisation presentation is the one that admits the axiomatic formulation used in the rest of the section.

2.2 Recovering FAD by Weighted-Sum Factorisation

The preceding subsection described $M(X)$ directly as the set of finitely supported nonnegative weight functions on X. We now recover the same object as the image of a weighted-sum transformation. The domain of this transformation is the ordered-list representation of weighted finite support: a list of point/log-weight pairs over X acts on a test function $f : X \to \mathbb{R}$ by evaluating f at the listed points and summing with weights $\exp\gamma_j$. Its image factorisation has $M(X)$ as the middle object: the left factor forms the corresponding finite atomic distribution by adding the weights of coincident points, while the right factor embeds such a distribution as its integration functional. This is the calculation in **Set** whose categorical content is isolated in the next subsection.

We begin with the real-valued case, which provides the barycentre algebra used below. For a list of point/log-weight pairs $\langle\langle t_1, \gamma_1\rangle, \cdots, \langle t_n, \gamma_n\rangle\rangle \in \coprod_{n\in\mathbb{N}} (\mathbb{R}\times\mathbb{R})^n$, where $t_j, \gamma_j \in \mathbb{R}$, define its unnormalised barycentre by[1]

$$\mathrm{bary}\Big(\langle\langle t_j, \gamma_j\rangle\rangle_{j=1}^n\Big) := \sum_{j=1}^n t_j \cdot \exp\gamma_j. \tag{1}$$

Here γ_j is a log-weight, and $\exp\gamma_j$ is the corresponding positive weight assigned to t_j. If the list contains *collisions*, meaning that $t_i = t_j =: t$ for distinct $i, j \in \{1, \cdots, n\}$, then the total weight assigned to t is the sum of $\exp\gamma_k$ over all k such that $t_k = t$.

For an arbitrary set X, we write

$$\mathsf{AST}(X) := \coprod_{n\in\mathbb{N}} (X\times\mathbb{R})^n$$

for the set of lists of point/log-weight pairs over X. Given a function $f : X \to \mathbb{R}$, define

$$\mathsf{AST}(f) : \mathsf{AST}(X) \to \mathsf{AST}(\mathbb{R})\,; \qquad \langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n \mapsto \langle\langle f(x_j), \gamma_j\rangle\rangle_{j=1}^n\,. \tag{2}$$

Here AST stands for *abstract syntax tree*, for reasons explained in Section 2.3. Thus $\mathsf{AST}(f)$ acts on a raw weighted presentation by applying f to its points and leaving the log-weights unchanged. Composing with the barycentre gives the weighted-sum map $(\mathrm{ws})_X$:

$$\begin{aligned}(\mathrm{ws})_X &: \mathsf{AST}(X) \to [[X,\mathbb{R}],\mathbb{R}]\\ \langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n &\mapsto \left(f \mapsto \mathrm{bary}\circ\mathsf{AST}(f)\Big(\langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n\Big) = f \mapsto \sum_{j=1}^n f(x_j)\cdot\exp\gamma_j\right).\end{aligned} \tag{3}$$

We factorise $(\mathrm{ws})_X$ through $M(X)$. Define maps $(\mathrm{categorical})_X$ and $(\int)_X$ such that

$$(\mathrm{ws})_X = \left(\int\right)_X \circ (\mathrm{categorical})_X : \mathsf{AST}(X) \to M(X) \to [[X,\mathbb{R}],\mathbb{R}]\,.$$

The left factor $(\mathrm{categorical})_X$ sends a raw presentation to the corresponding finite atomic distribution, accumulating the weights of all occurrences of each point. For any $\langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n \in \mathsf{AST}(X)$, define

$$(\mathrm{categorical})_X\Big(\langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n\Big) : X \to \mathbb{R} \quad \text{s.t.} \quad x \mapsto \sum_{\substack{1\le j\le n\\ x_j = x}} \exp\gamma_j. \tag{4}$$

Its support is exactly $\{x_1, \cdots, x_n\}$, since the displayed sum is zero precisely when it is empty. The name categorical reflects the standard probability theory nomenclature of finitely supported weight distributions. $(\mathrm{categorical})_X$ is surjective, but not injective; it forgets about ordering and collisions.

The right factor $(\int)_X$ sends a finite atomic distribution to the corresponding integration functional on test functions. For $w \in M(X)$, define

$$\left(\int\right)_X(w) : [X,\mathbb{R}] \to \mathbb{R} \quad \text{s.t.} \quad f \mapsto \sum_{x\in\operatorname{supp} w} f(x)\cdot w(x)\,. \tag{5}$$

[1] A normalised barycentre would require division by the total mass. Here we deliberately omit this normalisation.

This map is injective because indicator functions separate finite atomic distributions: for each $x \in X$, $(\int)_X(w)\left(\mathbb{1}_{\{x\}}\right) = w(x)$. Hence, for $w, w' \in M(X)$, $(\int)_X(w) = (\int)_X(w')$ if and only if $w = w'$.

The composite has the required weighted-sum semantics. Indeed, for any list $\langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n \in \mathsf{AST}(X)$ and any function $f : X \to \mathbb{R}$,

$$\left(\int\right)_X \circ (\text{categorical})_X\left(\langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n\right)(f) = \sum_{x\in\{x_1,\cdots,x_n\}} \left(f(x) \cdot \sum_{\substack{1\le j\le n \\ x_j = x}} \exp \gamma_j \right)$$
$$= \sum_{j=1}^n f(x_j) \cdot \exp \gamma_j = (\text{ws})_X\left(\langle\langle x_j, \gamma_j\rangle\rangle_{j=1}^n\right)(f) .$$

Thus $(\text{ws})_X = (\int)_X \circ (\text{categorical})_X$. Since $(\text{categorical})_X$ is surjective and $(\int)_X$ is injective, this is the *image factorisation* of $(\text{ws})_X$ in **Set**.

The calculation has isolated the three ingredients that we will abstract next. The ordered lists form the raw weighted-syntax monad AST; the barycentre algebra induces the weighted-sum morphism into a continuation monad; and the FAD monad on **Set** is recovered as the middle object of the resulting image factorisation. We next package these ingredients as a FAD factorisation structure on a category X and define the associated FAD monad on X.

2.3 FAD Monads by Axiomatic Factorisation

We now abstract the factorisation just described in **Set**. The aim is to specify the structure on a category X that is sufficient to form weighted raw syntax, interpret it by a barycentre algebra, and factorise the resulting strong monad morphism. Once these data are fixed, the FAD monad on X is not additional primitive structure: it is the monad obtained as the middle term of the factorisation

$$\mathsf{AST}^{\mathsf{X}} \xrightarrow{\text{categorical}^{\mathsf{X}}} M^{\mathsf{X}} \xrightarrow{\int^{\mathsf{X}}} \mathsf{K}V^{\mathsf{X}}.$$

For $\mathsf{X} = \mathbf{Set}$, with the choices specified below, this construction recovers the FAD monad on **Set**.

We assume that X is bicartesian closed with countable coproducts. Thus we fix a terminal object 1, an initial object 0, binary products and coproducts, countable coproducts, and, for each object A, a right adjoint $[A, -] : \mathsf{X} \to \mathsf{X}$ to $(-) \times A : \mathsf{X} \to \mathsf{X}$.

We also fix a monoid object R in X, with unit $1_R : 1 \to R$ and multiplication $\cdot_R : R \times R \to R$. This is the *monoid of weights*; in the motivating instance, it is the additive monoid of log-weights. Finally, we fix an R-semimodule V, called the *barycentre semimodule*, with structure maps

$$+_V : V \times V \to V, \qquad 0_V : 1 \to V, \qquad \cdot_R^V : V \times R \to V.$$

Here an R-semimodule means a commutative monoid object V equipped with a right action $V \times R \to V$ of the monoid object R by commutative-monoid endomorphisms.

In **Set**, the relevant choices are $R = \mathbb{R}$ (thought of as log-weights), with

$$\begin{aligned} 1_R : 1 &\to \mathbb{R} \\ * &\mapsto 0 \end{aligned} \quad \text{and} \quad \begin{aligned} \cdot_R : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ \langle a, b\rangle &\mapsto a + b, \end{aligned} \tag{6}$$

and $V = \mathbb{R}$ (thought of as real values), with

$$\begin{aligned} 0_V : 1 &\to \mathbb{R} \\ * &\mapsto 0 \end{aligned} \quad\text{and}\quad \begin{aligned} +_V : \mathbb{R}\times\mathbb{R} &\to \mathbb{R} \\ \langle a, b\rangle &\mapsto a+b \end{aligned} \quad\text{and}\quad \begin{aligned} \cdot^V_R : \mathbb{R}\times\mathbb{R} &\to \mathbb{R} \\ \langle v, r\rangle &\mapsto v\cdot \exp r. \end{aligned} \tag{7}$$

The case $V = \mathbb{R}^p$, for $p \in \mathbb{N}$, is obtained in the same way.

From R and V we form two monads. The raw weighted-syntax monad and the continuation monad on V have underlying endofunctors

$$\mathsf{AST}^{\mathsf{X}} := \coprod_{n\in\mathbb{N}} (-\times R)^n \quad\text{and}\quad \mathsf{K}V^{\mathsf{X}} := [[-, V], V]\,. \tag{8}$$

The monad structure on $\mathsf{AST}^{\mathsf{X}}$ is obtained by combining the ordered-list monad $A \mapsto \coprod_{n\in\mathbb{N}} A^n$ Dubuc (1974) with the R-writer monad $A \mapsto A \times R$, induced by the monoid $\langle R, \cdot_R, 1_R\rangle$; see, for instance, Chapter 7 of Mac Lane (1997). These monads are combined using the canonical distributive law recalled in Section 6.3 of Bacci et al. (2024). Thus $\mathsf{AST}^{\mathsf{X}}$ is an ordered-list presentation of weighted finite support. The identifications imposed by permutations and collisions are not built into $\mathsf{AST}^{\mathsf{X}}$; they are produced by factorisation.

Both $\mathsf{AST}^{\mathsf{X}}$ and $\mathsf{K}V^{\mathsf{X}}$ are strong monads. For $\mathsf{AST}^{\mathsf{X}}$, the strength is induced by distributivity of products over countable coproducts and by the canonical maps $A \times B^n \to A^n \times B^n \xrightarrow{\sim} (A \times B)^n$ obtained from the diagonal morphism and tuplings of projections. Then, by using $B \times R$ in place of B, this gives a natural transformation

$$A \times \mathsf{AST}^{\mathsf{X}}(B) \to \mathsf{AST}^{\mathsf{X}}(A \times B)\,.$$

The continuation monad carries its standard strength in every cartesian closed category.

The semimodule structure on V defines an $\mathsf{AST}^{\mathsf{X}}$-algebra structure on V, the *barycentre algebra*. Explicitly, it is the morphism

$$\mathrm{bary}^{\mathsf{X}} : \mathsf{AST}^{\mathsf{X}}(V) \to V; \quad\text{defined by}\quad \mathrm{bary}^{\mathsf{X}} = \left[\sum_n \circ \left(\cdot^V_R\right)^n\right]_{n\in\mathbb{N}}, \tag{9}$$

where $\sum_n : V^n \to V$ is the n-ary sum determined by $+_V$ and 0_V, and $\left(\cdot^V_R\right)^n : (V \times R)^n \to V^n$ is the product of n copies of $\cdot^V_R$.

Under continuation-passing style, the barycentre algebra corresponds to a strong monad morphism ws^{X} to the continuation monad $[[-, V], V]$ whose A-component sends $a \in \mathsf{AST}^{\mathsf{X}}A$ to $\lambda k.\ \mathrm{bary}^{\mathsf{X}}\left(\mathsf{AST}^{\mathsf{X}}k(a)\right)$.

Lemma 2.1 (Algebra CPS; see Appendix G)**.** *Let $\left\langle T, \mu^T, \eta^T, \mathrm{st}^T\right\rangle$ be a strong monad on a cartesian closed category X, and let V be an object of X. Then there is a 1:1-correspondence between T-algebra structures on V and strong monad morphisms $T \to \mathsf{K}V^{\mathsf{X}}$.*

Applying Lemma 2.1 to the barycentre algebra gives a strong monad morphism

$$\mathrm{ws}^{\mathsf{X}} : \mathsf{AST}^{\mathsf{X}} \to \mathsf{K}V^{\mathsf{X}} \qquad\text{with components}\qquad \left(\mathrm{ws}^{\mathsf{X}}\right)_A : \mathsf{AST}^{\mathsf{X}}(A) \to [[A, V], V]\,.$$

It remains to specify the factorisation. In **Set**, we used the image factorisation. In X, we fix an orthogonal factorisation system $\langle \mathcal{E}, \mathcal{M}\rangle$. Thus every $f : A \to B$ factors as $f = m \circ e$, with $e \in \mathcal{E}$ and $m \in \mathcal{M}$, and $\mathcal{E}$ is orthogonal to $\mathcal{M}$: for every equation $v \circ e = m \circ u$, with $e \in \mathcal{E}$ and $m \in \mathcal{M}$, there is a unique w such that $u = w \circ e$ and $v = m \circ w$. In **Set**, taking $\mathcal{E}$

to be the epis (i.e., surjections) and $\mathcal{M}$ to be the (regular) monos (i.e., injections) recovers the usual image factorisation. We assume that $\mathcal{E}$ is closed under binary products. Since left classes of orthogonal factorisation systems are closed under coproducts, this implies closure of $\mathcal{E}$ under $\mathsf{AST}^{\mathsf{X}}$.

For each object A, factorise the component $\left(\mathrm{ws}^{\mathsf{X}}\right)_A$ as

$$\begin{array}{ccc} \mathsf{AST}^{\mathsf{X}}(A) & \xrightarrow{\left(\mathrm{ws}^{\mathsf{X}}\right)_A} & \mathsf{K}V^{\mathsf{X}}(A) \\ & {\scriptstyle \left(\mathrm{categorical}^{\mathsf{X}}\right)_A} \searrow \quad \nearrow {\scriptstyle \left(\int^{\mathsf{X}}\right)_A} & \\ & M^{\mathsf{X}}(A) & \end{array}$$

with $\left(\mathrm{categorical}^{\mathsf{X}}\right)_A \in \mathcal{E}$ and $\left(\int^{\mathsf{X}}\right)_A \in \mathcal{M}$. At this stage $M^{\mathsf{X}}(A)$ denotes only the intermediate objects of the factorisation. The next theorem ensures that these objects assemble into a strong monad.

Lemma 2.2 (Factorisation of strong monad morphisms, (Kammar and McDermott, 2018, Theorem 2.5))**.** *Let X be a category, $\langle \mathcal{E}, \mathcal{M} \rangle$ a factorisation system, S and T monads on X, and $m : S \to T$ a monad morphism. Denote by $m[S]$ the functor obtained by pointwise factorisation of m, with $m^{\mathbf{e}}$ and $m^{\mathbf{m}}$ denoting the pointwise left- and right-class factors. If $\mathcal{E}$ is closed under S, then $m[S]$ is a monad, and $m^{\mathbf{e}}$ and $m^{\mathbf{m}}$ are monad morphisms. If, moreover, $\mathcal{E}$ is closed under binary products, S and T are strong monads, and m is a strong monad morphism, then $m[S]$ is a strong monad and $m^{\mathbf{e}}$ and $m^{\mathbf{m}}$ are strong monad morphisms.*

We can now name the structure and the monad it determines.

Definition 2.3 (FAD factorisation structure and associated FAD monad)**.** A *FAD factorisation structure* on X consists of the bicartesian closed structure with countable coproducts fixed above, a monoid of weights $\langle R, \cdot_R, 1_R \rangle$, a barycentre semimodule $\left\langle V, +_V, 0_V, \cdot^{V}_{R} \right\rangle$, and an orthogonal factorisation system $\langle \mathcal{E}, \mathcal{M} \rangle$ whose left class is closed under binary products.

Its associated *FAD monad* is the monad M^{X} obtained by the pointwise factorisation

$$\mathsf{AST}^{\mathsf{X}} \xrightarrow{\mathrm{categorical}^{\mathsf{X}}} M^{\mathsf{X}} \xrightarrow{\int^{\mathsf{X}}} \mathsf{K}V^{\mathsf{X}} \qquad \text{of} \qquad \mathrm{ws}^{\mathsf{X}} : \mathsf{AST}^{\mathsf{X}} \to \mathsf{K}V^{\mathsf{X}}.$$

Proposition 2.4. *For every FAD factorisation structure on X, the associated FAD monad M^{X} is strong, and the maps $\mathrm{categorical}^{\mathsf{X}} : \mathsf{AST}^{\mathsf{X}} \to M^{\mathsf{X}}$ and $\int^{\mathsf{X}} : M^{\mathsf{X}} \to \mathsf{K}V^{\mathsf{X}}$ are strong monad morphisms.*

Proof. Apply Lemma 2.2 to $\mathrm{ws}^{\mathsf{X}} : \mathsf{AST}^{\mathsf{X}} \to \mathsf{K}V^{\mathsf{X}}$. The required closure of $\mathcal{E}$ under $\mathsf{AST}^{\mathsf{X}}$ follows from closure under binary products and from closure of left classes under coproducts. □

With $\mathsf{X} = \mathbf{Set}$, $R = \mathbb{R}$, $V = \mathbb{R}$, and the (epi, regular mono) factorisation, the associated FAD monad $M^{\mathbf{Set}}$ is canonically isomorphic to the FAD monad on **Set**. The point of the preceding abstraction is that the same construction can now be instantiated in categories with additional semantic structure. The first such instance is $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, where the associated FAD monad refines the FAD monad on the underlying set by adding cotangent fibres for atom locations and log-weights.

2.4 FAD Monad on $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$

We now exhibit a FAD factorisation structure on $\mathsf{F} := \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and write M^{F} for its associated FAD monad. This is the semantic category for differentiated values in reverse-mode CHAD: an object carries a set of possible values together with a specified cotangent space over each value, and a morphism carries both the underlying function on values and the corresponding contravariant action on cotangents. We think of this action as a transposed derivative.

The category $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ is the free coproduct completion of $\mathbf{Vect}^{\mathrm{op}}$ Adámek and Rosický (2020). We use it as the category of sets equipped with cotangent fibres. An object is a pair $\langle X, A\rangle$, where X is a set and $A : X \to \mathbf{Vect}^{\mathrm{op}}$ is a functor, viewing X as a discrete category. Equivalently, A is a family $\{A(x)\}_{x\in X}$ of vector spaces indexed by X.

A morphism $\langle X, A\rangle \to \langle Y, B\rangle$ is a pair $\langle f, \alpha\rangle$, where $f : X \to Y$ is a function and $\alpha : A \to B \circ f$ is a natural transformation in $\mathbf{Vect}^{\mathrm{op}}$. Equivalently, for each $x \in X$, it determines a linear map

$$(\alpha)_x : B(f(x)) \to A(x)\,.$$

Thus f acts on base points, while $(\alpha)_x$ acts contravariantly on cotangents. In the intended interpretation, $A(x)$ is the cotangent space at x, and $(\alpha)_x$ plays the role of the transposed derivative of f at x. We write $(\langle X, A\rangle)_{\mathrm{set}} = X$, $(\langle X, A\rangle)_{\mathrm{fbr}} = A$, $(\langle f, \alpha\rangle)_{\mathrm{set}} = f$, $(\langle f, \alpha\rangle)_{\mathrm{fbr}} = \alpha$ for the base and fibre parts. We use this notation throughout the construction below.

The category $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ is complete and cocomplete. In particular, for any family $\{\langle X_i, A_i\rangle\}_{i\in I}$ of objects, products and coproducts are given by

$$\prod_{i\in I}\langle X_i, A_i\rangle = \left\langle \left(\prod_{i\in I} X_i\right), \left(\{x_i\}_{i\in I} \mapsto \bigoplus_{i\in I} A_i(x_i)\right)\right\rangle \quad \text{and} \quad \coprod_{i\in I}\langle X_i, A_i\rangle = \left\langle \left(\coprod_{i\in I} X_i\right), [A_i]_{i\in I}\right\rangle. \tag{10}$$

Moreover, $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ has exponentials (Lucatelli Nunes and Vákár, 2023, Proposition 74): $[\langle X, A\rangle, \langle Y, B\rangle] = \left\langle \prod_{x\in X} \sum_{y\in Y} \mathbf{Vect}(B(y), A(x)), f \mapsto \bigoplus_{x\in X} B(\pi_1(f(x)))\right\rangle$. Hence $\mathsf{F} = \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ is bicartesian closed with countable coproducts, as required by the axiomatic construction.

We use the (epi, regular mono) factorisation system on $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. It admits the following explicit description.

Lemma 2.5. *Epis and regular monos form a factorisation system on* $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. *For a morphism* $\langle f, \alpha\rangle : \langle X, A\rangle \to \langle Y, B\rangle$, *membership in the left class* $\mathcal{E}$ *is equivalent to* $f : X \to Y$ *being surjective and* α *being jointly injective, in the sense that, for each* $y \in Y$, *the linear map*

$$\langle (\alpha)_x\rangle_{\substack{x\in X\\ f(x)=y}} : B(y) \to \prod_{\substack{x\in X\\ f(x)=y}} A(x)$$

is injective. Membership in the right class $\mathcal{M}$ *is equivalent to* $f : X \to Y$ *being injective and* α *being pointwise surjective, in the sense that, for each* $x \in X$, *the linear map*

$$(\alpha)_x : B(f(x)) \to A(x)$$

is surjective. Moreover, the left class $\mathcal{E}$ *is closed under binary products.*

Therefore F has the categorical structure and factorisation system required for a FAD factorisation structure. It remains to choose the monoid of weights and the barycentre semimodule. Both are obtained by lifting the corresponding **Set**-structures to the real cotangent bundle

$$T^*\mathbb{R} := \langle \mathbb{R}, \mathrm{const}\mathbb{R} \rangle \,, \qquad \text{where} \qquad \mathrm{const}Y(\ x) \ = \ Y.$$

The monoid of weights is $T^*\mathbb{R}$. Its unit and multiplication are

$$\begin{array}{llll llll}
1_R & : & \langle 1, \mathrm{const}0 \rangle & \to T^*\mathbb{R} & \cdot_R & : & T^*\mathbb{R} \times T^*\mathbb{R} & \to T^*\mathbb{R} \\
(1_R)_{\mathrm{set}} & : & 1 & \to \mathbb{R} & (\cdot_R)_{\mathrm{set}} & : & \mathbb{R} \times \mathbb{R} & \to \mathbb{R} \\
 & & * & \mapsto 0 & & & \langle a, b \rangle & \mapsto a + b \\
((1_R)_{\mathrm{fbr}})_* & : & \mathbb{R} & \to 0 & ((\cdot_R)_{\mathrm{fbr}})_{\langle a,b \rangle} & : & \mathbb{R} & \to \mathbb{R} \oplus \mathbb{R} \\
 & & v & \mapsto 0 & & & v & \mapsto \langle v, v \rangle \,.
\end{array} \tag{11}$$

The base parts agree with Equation (6); the fibre parts are the corresponding transposed derivatives.

We also take the barycentre semimodule to be $T^*\mathbb{R}$.[2] Its zero element and addition are

$$\begin{array}{llll llll}
0_V & : & \langle 1, \mathrm{const}0 \rangle & \to T^*\mathbb{R} & +_V & : & T^*\mathbb{R} \times T^*\mathbb{R} & \to T^*\mathbb{R} \\
(0_V)_{\mathrm{set}} & : & 1 & \to \mathbb{R} & (+_V)_{\mathrm{set}} & : & \mathbb{R} \times \mathbb{R} & \to \mathbb{R} \\
 & & * & \mapsto 0 & & & \langle a, b \rangle & \mapsto a + b \\
((0_V)_{\mathrm{fbr}})_* & : & \mathbb{R} & \to 0 & ((+_V)_{\mathrm{fbr}})_{\langle a,b \rangle} & : & \mathbb{R} & \to \mathbb{R} \oplus \mathbb{R} \\
 & & v & \mapsto 0 & & & v & \mapsto \langle v, v \rangle \,.
\end{array} \tag{12}$$

Again, the base parts are those of Equation (7), and the fibre parts are their transposed derivatives. The action of the monoid of weights on the barycentre semimodule is

$$\begin{array}{llll}
\cdot_R^V & : & T^*\mathbb{R} \times T^*\mathbb{R} & \to T^*\mathbb{R} \\
\left(\cdot_R^V\right)_{\mathrm{set}} & : & \mathbb{R} \times \mathbb{R} & \to \mathbb{R} \\
 & & \langle v, r \rangle & \mapsto v \cdot \exp r \\
\left(\left(\cdot_R^V\right)_{\mathrm{fbr}}\right)_{\langle v,r \rangle} & : & \mathbb{R} & \to \mathbb{R} \oplus \mathbb{R} \\
 & & t & \mapsto \langle t \cdot \exp r, t \cdot v \cdot \exp r \rangle \,.
\end{array} \tag{13}$$

The base part is the action used in **Set**, while the fibre part is its transposed derivative.

These choices make $\mathsf{F} = \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ into a FAD factorisation structure. Let M^{F} denote the associated FAD monad. For an object $\langle X, A \rangle$, the factorisation construction gives

$$M^{\mathsf{F}}(\langle X, A \rangle) := \left\langle M(X) \,, \left(w \mapsto \bigoplus_{x \in \operatorname{supp} w} (A(x) \oplus \mathbb{R}) \right) \right\rangle . \tag{14}$$

Thus the base of $M^{\mathsf{F}}(\langle X, A \rangle)$ is the set of finite atomic distributions on X. The fibre over $w \in M(X)$ has one summand for each atom $x \in \operatorname{supp} w$: the component $A(x)$ carries cotangent information for the position of the atom, while the real component carries cotangent information for $\log w(x)$.

[2] As in **Set**, one may instead take $T^*\mathbb{R}^p$ for some $p \in \mathbb{N}$.

For a morphism $\langle f, \alpha \rangle : \langle X, A \rangle \to \langle Y, B \rangle$, the base part is

$$\Big(M^{\mathsf{F}}(\langle f, \alpha \rangle)\Big)_{\mathrm{set}} = M(f)\,.$$

For $w \in M(X)$, the fibre part is

$$\Big(\Big(M^{\mathsf{F}}(\langle f, \alpha \rangle)\Big)_{\mathrm{fbr}}\Big)_w : \bigoplus_{y \in f(\operatorname{supp} w)} (B(y) \oplus \mathbb{R}) \to \bigoplus_{x \in \operatorname{supp} w} (A(x) \oplus \mathbb{R})$$

$$\sum_{y \in \operatorname{supp} M(f)(w)} (b_y \cdot \mathbf{e}_y + s_y \cdot \varepsilon_y) \mapsto \sum_{x \in \operatorname{supp} w} \left(\frac{w(x) \cdot \Big((\alpha)_x \Big(b_{f(x)}\Big) \cdot \mathbf{e}_x + s_{f(x)} \cdot \varepsilon_x\Big)}{M(f)(w)(f(x))} \right). \tag{15}$$

Here $\mathbf{e}_\bullet$ denotes the atom-location component and $\varepsilon_\bullet$ the log-weight component in the relevant direct sum. The formula says that cotangents are pulled back along $\langle f, \alpha \rangle$ by distributing the cotangent at an image atom among all atoms mapping to it, with the contribution of x scaled by the relative weight

$$\frac{w(x)}{M(f)(w)(f(x))}.$$

The atom component is then transported through $(\alpha)_x$, while the log-weight component is copied to the corresponding ε_x-summand.

Under the base projection, the monoid of weights, the barycentre semimodule, and the monad M^{F} recover the corresponding structures in **Set**. Thus M^{F} is a lift of the FAD monad on **Set** along the base projection, with the additional fibre information required for reverse-mode differentiation. Over a distribution $w \in M(X)$, this fibre consists of one cotangent component for each atom $x \in \operatorname{supp} w$, and one real cotangent component for the log-weight $\log w(x)$.

2.4.1 Expected-value gradients and score contributions

We now spell out how the lifted FAD monad separates the usual contributions to the derivative of an expected value. The point is not to introduce an external analytic argument, but to show that the decomposition is already encoded by the fibre maps in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. In particular, for a function $f(\theta, z)$, with z sampled from a finite distribution $q(\theta)$ depending on the parameter θ, the lifted semantics separates three sources of variation: the explicit dependence of f on θ, the dependence of the atoms z on θ, and the dependence of the log-weights on θ.

To make this comparison readable, we use differential notation for objects and morphisms in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. Thus we write

$$T^*U = \langle U, \{T^*_u U\}_{u \in U} \rangle$$

and write morphisms as

$$\hat{h} = \langle h, \boldsymbol{\nabla} h \rangle : T^*X \to T^*Y, \qquad (\boldsymbol{\nabla} h)_x : T^*_{h(x)}Y \to T^*_x X,$$

where $(\boldsymbol{\nabla} h)_x$ denotes the x-fibre of the fibre part.

Consider the lifted real-valued expectation algebra

$$\hat{\mathrm{E}}^{\mathsf{F}}_{\mathbb{R}} : M^{\mathsf{F}}(T^*\mathbb{R}) \to T^*\mathbb{R}.$$

For $u \in M(\mathbb{R})$, its base and fibre parts are

$$\mathrm{E}^{\mathsf{F}}_{\mathbb{R}}(u) = \sum_{r \in \operatorname{supp} u} u(r) \cdot r, \qquad \left(\boldsymbol{\nabla}\, \mathrm{E}^{\mathsf{F}}_{\mathbb{R}}\right)_u(t) = \sum_{r \in \operatorname{supp} u} \left(u(r) \cdot t \cdot \mathbf{e}_r + u(r) \cdot r \cdot t \cdot \varepsilon_r\right). \tag{16}$$

The two summands in the fibre component have different origins. The $\mathbf{e}_r$-term is the cotangent contribution of the atom value r; the ε_r-term is the cotangent contribution of the log-weight $\log u(r)$.

Let

$$\hat{g} = \langle g, \boldsymbol{\nabla} g\rangle : T^*X \to T^*\mathbb{R}, \qquad w \in M(X), \qquad u := M(g)(w).$$

The Frobenius identity gives

$$\sum_{r \in \operatorname{supp} u} u(r) \cdot r = \sum_{x \in \operatorname{supp} w} w(x) \cdot g(x).$$

On fibres, composing the map induced by $M^{\mathsf{F}}(\hat{g})$ with Equation (16) gives

$$\left(\boldsymbol{\nabla}\left(\hat{\mathrm{E}}^{\mathsf{F}}_{\mathbb{R}} \circ M^{\mathsf{F}}(\hat{g})\right)\right)_w(t) = \sum_{x \in \operatorname{supp} w} \left(w(x) \cdot (\boldsymbol{\nabla} g)_x(t) \cdot \mathbf{e}_x + w(x) \cdot g(x) \cdot t \cdot \varepsilon_x\right). \tag{17}$$

Indeed, the fibre map of $M^{\mathsf{F}}(\hat{g})$ contributes, in the x-summand, the relative factor

$$\frac{w(x)}{M(g)(w)(g(x))},$$

while Equation (16) contributes the corresponding mass

$$M(g)(w)(g(x)).$$

These factors cancel, leaving precisely the expression in Equation (17).

Now let

$$\hat{q} = \langle q, \boldsymbol{\nabla} q\rangle : T^*\Theta \to M^{\mathsf{F}}(T^*Z)$$

be a parameterised FAD. For each $\theta \in \Theta$, choose a local collision-free presentation

$$q(\theta) = \operatorname{categorical}(\langle\langle z_i(\theta), \ell_i(\theta)\rangle\rangle_{i \in I}) \quad \text{and} \quad w_i(\theta) := \exp \ell_i(\theta).$$

Let

$$\underline{\hat{q}} := \left(\mathrm{st}^{\mathsf{F}}\right)_{T^*\Theta, T^*Z} \circ \langle \mathrm{id}_{T^*\Theta}, \hat{q}\rangle.$$

Then

$$\underline{q}(\theta) = \operatorname{categorical}(\langle\langle \theta, z_i(\theta)\rangle, \ell_i(\theta)\rangle_{i \in I}).$$

With respect to this local labelling, the fibre part of $\underline{\hat{q}}$ is

$$\left(\boldsymbol{\nabla}\underline{q}\right)_\theta\left(\sum_{i \in I}\left(\langle d_i, a_i\rangle \cdot \mathbf{e}_{\langle\theta, z_i(\theta)\rangle} + s_i \cdot \varepsilon_{\langle\theta, z_i(\theta)\rangle}\right)\right) = \sum_{i \in I} d_i + \sum_{i \in I} (\boldsymbol{\nabla} z_i)_\theta(a_i) + \sum_{i \in I} (\boldsymbol{\nabla} \ell_i)_\theta(s_i), \tag{18}$$

where

$$d_i \in T^*_\theta\Theta, \qquad a_i \in T^*_{z_i(\theta)}Z, \qquad s_i \in \mathbb{R}.$$

Finally, let

$$\widehat{f} = \langle f, \boldsymbol{\nabla} f\rangle : T^*\Theta \times T^*Z \to T^*\mathbb{R}.$$

We write the two components of its fibre part as

$$(\boldsymbol{\nabla} f)_{\langle\theta,z\rangle}(t) = \left\langle \left((\boldsymbol{\nabla} f)^{\Theta}\right)_{\langle\theta,z\rangle}(t)\,, \left((\boldsymbol{\nabla} f)^{Z}\right)_{\langle\theta,z\rangle}(t) \right\rangle.$$

Apply Equation (17) to the composite

$$T^*\Theta \xrightarrow{\hat{\underline{q}}} M^{\mathsf{F}}(T^*\Theta \times T^*Z) \xrightarrow{M^{\mathsf{F}}(\widehat{f})} M^{\mathsf{F}}(T^*\mathbb{R}) \xrightarrow{\mathrm{E}^{\mathsf{F}}_{\mathbb{R}}} T^*\mathbb{R}$$

and then use Equation (18). This gives

$$\left(\boldsymbol{\nabla}\left(\hat{\mathrm{E}}^{\mathsf{F}}_{\mathbb{R}} \circ M^{\mathsf{F}}\left(\widehat{f}\right) \circ \hat{\underline{q}}\right)\right)_{\theta}(t) = \sum_{i\in I} w_i(\theta) \cdot \left[\left((\boldsymbol{\nabla} f)^{\Theta}\right)_{\langle\theta,z_i(\theta)\rangle}(t) + (\boldsymbol{\nabla} z_i)_{\theta}\left(\left((\boldsymbol{\nabla} f)^{Z}\right)_{\langle\theta,z_i(\theta)\rangle}(t)\right) \right.$$
$$\left. + \; f(\langle\theta, z_i(\theta)\rangle) \cdot (\boldsymbol{\nabla}\ell_i)_{\theta}(t)\right]. \tag{19}$$

The three terms inside the brackets have distinct origins. The first is the explicit parameter contribution, coming from the dependence of $f(\theta, z)$ on θ. The second is the moving-atom contribution, obtained by transporting cotangents through the dependence of $z_i(\theta)$ on θ. The third is the score contribution, coming from the dependence of the log-weight $\ell_i(\theta) = \log w_i(\theta)$ on θ.

This calculation explains the operational role of the lifted FAD monad in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. It separates the reverse-mode contributions carried by a finite probabilistic computation into atom-location terms and log-weight terms, and it does so internally, through the fibre maps of the associated FAD monad. The next subsection lifts the construction once more, from $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ to the logical-relations category used to prove that the syntactic CHAD transformation computes these cotangent contributions correctly.

2.5 FAD Monad on the Logical-Relations Category

The final instance of the FAD factorisation construction is the logical-relations category used in the correctness proof. Whereas $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ records the differentiated semantics of programs, LR relates this differentiated semantics to the ordinary semantics in **Set**. The purpose of this subsection is to equip LR with a FAD factorisation structure, and hence with an associated FAD monad M^{LR}. This is the semantic ingredient needed later to interpret the source language in the logical-relations model and to prove correctness of the reverse-mode CHAD transformation.

We use the comma-category presentation of logical relations from (Lucatelli Nunes and Vákár, 2023, Section 11). Let $G : \mathsf{C} \to \mathsf{D}$ be a functor. We write

$$\mathsf{LR} := (\mathsf{D} \downarrow G)\,.$$

An object of LR is a triple $\langle D, C, f\rangle$, where D is an object of D, C is an object of C, and $f : D \to G(C)$ is a morphism in D. A morphism

$$\langle u, v\rangle : \langle D, C, f\rangle \to \langle D', C', f'\rangle$$

consists of morphisms $u : D \to D'$ in D and $v : C \to C'$ in C such that $G(v) \circ f = f' \circ u$. There is therefore a forgetful functor

$$U : \mathsf{LR} \to \mathsf{D} \times \mathsf{C}, \qquad \text{defined by} \qquad U(\langle D, C, f\rangle) := \langle D, C\rangle\,, \qquad U(\langle u, v\rangle) := \langle u, v\rangle\,.$$

For the correctness argument in this paper, we take

$$\mathsf{D} := \mathbf{Set}, \qquad \mathsf{C} := \mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}}),$$

and define G to be the hom-functor

$$G := \mathsf{C}(\langle \mathbb{E}, T^*\mathbb{E} \rangle, -), \tag{20}$$

where $\mathbb{E}$ is a fixed finite-dimensional real vector space, regarded as a set, and $T^*\mathbb{E} := \langle \mathbb{E}, \mathrm{const}\mathbb{E}^* \rangle$ is its cotangent bundle.

The space $\mathbb{E}$ is used as a probe object. For example, when $\mathbb{E} = \mathbb{R}$, morphisms from $\langle \mathbb{E}, T^*\mathbb{E} \rangle$ into an object of C can be read as parametrised curves equipped with cotangent data. In this way, differentiability of a map can be tested by its action on such probes: postcomposition should send admissible probes in the domain to admissible probes in the codomain.

Thus, in the present instance, an object of LR is a triple $\langle D, C, p \rangle$ where $p : D \to \mathsf{C}(\langle \mathbb{E}, T^*\mathbb{E} \rangle, C)$ is a function. When p is injective, D may be regarded as a specified class of probes $\langle \mathbb{E}, T^*\mathbb{E} \rangle \to C$, namely those satisfying the predicate represented by p. A morphism $\langle u, v \rangle : \langle D, C, p \rangle \to \langle D', C', p' \rangle$ asserts that postcomposition with $v : C \to C'$ sends probes satisfying p to probes satisfying p'.

We write

$$(\langle D, C, p \rangle)_{\mathrm{dom}} := D, \qquad (\langle D, C, p \rangle)_{\mathrm{cod}} := C, \qquad (\langle u, v \rangle)_{\mathrm{dom}} := u, \qquad (\langle u, v \rangle)_{\mathrm{cod}} := v$$

for the domain and codomain parts of objects and morphisms of LR.

In the correctness proof, the predicates represented in LR will express differentiability and correctness of the syntactic reverse derivative. Accordingly, the source language must admit an interpretation in LR. This requires, in particular, that LR carry the categorical structure needed for the language: finite products and coproducts, the FAD monad, and the distinguished objects and operations used to interpret base types and primitives.[3] The remainder of this subsection verifies the FAD part of this structure by applying the FAD factorisation construction of Section 2.3.

The main structural input is the monadic-comonadic analysis of these comma categories from the CHAD logical-relations construction.

Lemma 2.6 ((Lucatelli Nunes and Vákár, 2023, Corollary 99))**.** *Assume that* C *has binary coproducts and* D *has binary products. Then* $U : (\mathsf{D} \downarrow G) \to \mathsf{D} \times \mathsf{C}$ *is comonadic and monadic, provided that* G *has a left adjoint.*

In the present case, $\mathsf{C} = \mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ is complete and cocomplete; hence so is $\mathbf{Set} \times \mathsf{C}$. Since C has copowers, the hom-functor G has a left adjoint, and Lemma 2.6 applies. Thus

$$U : \mathsf{LR} \to \mathbf{Set} \times \mathsf{C}$$

is both monadic and comonadic. The monadic part implies that U creates limits, while the comonadic part implies that U creates colimits. Hence LR is complete and cocomplete, and its limits and colimits are computed by taking the corresponding limits and colimits in the underlying $\mathbf{Set}$- and C-components, with the unique induced logical-relation structure.

[3] For the description of the source language and its interpretation requirements, see Section 3.1.

The same componentwise description gives the factorisation system required by Section 2.3. A morphism $\langle u, v\rangle$ lies in the left class precisely when u and v lie in the corresponding left classes in **Set** and $\mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$; similarly for the right class. Writing $v = \langle v_1, v_2\rangle$, the left-class condition says that u and v_1 are surjective and that v_2 lies in the left class described in Section 2.4. The right-class condition says that u and v_1 are injective and that v_2 lies in the corresponding right class. This is the usual image factorisation, equivalently the (epi, regular mono) factorisation system in LR. Since the left classes in **Set** and $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ are closed under binary products, and U creates products, the left class in LR is closed under binary products as well.

It remains to specify the monoid of weights and the barycentre semimodule in LR. Both are obtained by imposing the differentiability predicate on the corresponding structures in $\mathsf{C} = \mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. The monoid of weights is $\langle \mathrm{Diff}\,(\mathbb{E}, \mathbb{R})\,, \langle \mathbb{R}, T^*\mathbb{R}\rangle\,, \mathrm{dv}_{\mathbb{R}}\rangle$ where $\mathrm{Diff}\,(\mathbb{E}, \mathbb{R})$ denotes the set of differentiable functions $\mathbb{E} \to \mathbb{R}$, and $\mathrm{dv}_{\mathbb{R}}$ is the usual differentiability predicate. More generally, for each $k \in \mathbb{N}$,

$$\mathrm{dv}_{\mathbb{R}^k} : \mathrm{Diff}\left(\mathbb{E}, \mathbb{R}^k\right) \to \mathbf{Set}\left(\mathbb{E}, \mathbb{R}^k\right) \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})\left(T^*\mathbb{E}, T^*\mathbb{R}^k\right) \quad \text{s.t.} \quad f \mapsto \langle f, \langle f, \boldsymbol{\nabla} f\rangle\rangle\,. \tag{21}$$

The barycentre semimodule is given by $\langle \mathrm{Diff}\,(\mathbb{E}, \mathbb{R})\,, \langle \mathbb{R}, T^*\mathbb{R}\rangle\,, \mathrm{dv}_{\mathbb{R}}\rangle$.[4]

The structure maps for the monoid and the semimodule are induced componentwise from their counterparts in **Set** and $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, with the domain part given by postcomposition on differentiable functions. For example, the operation $+_V$ is defined by

$$\begin{aligned} (+_V)_{\mathrm{dom}} &:= +_V^{\mathbf{Set}} \circ - &&: \ \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}^2) &&\to \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}) \\ (+_V)_{\mathrm{cod}} &:= \left\langle +_V^{\mathbf{Set}}, +_V^{\mathsf{F}} \right\rangle &&: \ \langle \mathbb{R}^2, T^*\mathbb{R}^2\rangle &&\to \langle \mathbb{R}, T^*\mathbb{R}\rangle\,. \end{aligned} \tag{22}$$

Here we use the canonical identifications

$$\mathrm{Diff}\,(\mathbb{E}, \mathbb{R}) \times \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}) \simeq \mathrm{Diff}\left(\mathbb{E}, \mathbb{R}^2\right) \quad \text{and} \quad \langle \mathbb{R}, T^*\mathbb{R}\rangle \times \langle \mathbb{R}, T^*\mathbb{R}\rangle \simeq \left\langle \mathbb{R}^2, T^*\mathbb{R}^2 \right\rangle.$$

The remaining structural morphisms are defined analogously. In each case, the codomain part is the corresponding morphism in $\mathsf{C} = \mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, while the domain part restricts it to the differentiable probes.

This componentwise construction is the sense in which the logical-relations structure is fibred over C: objects, structural maps, limits, colimits, and factorisations in LR project to their corresponding constructions in C.

We may therefore apply the FAD factorisation construction of Section 2.3 to LR. The resulting monad M^{LR} is fibred over $\mathsf{C} = \mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. Concretely, for an object $\langle D, C, p\rangle$ and a morphism $\langle u, v\rangle : \langle D, C, p\rangle \to \langle D', C', p'\rangle$ in LR,

$$\left(M^{\mathsf{LR}}(\langle D, C, p\rangle)\right)_{\mathrm{cod}} = M \times M^{\mathsf{F}}(C) \quad \text{and} \quad \left(M^{\mathsf{LR}}(\langle u, v\rangle)\right)_{\mathrm{cod}} = M \times M^{\mathsf{F}}(v)\,. \tag{23}$$

The domain part records the admissible differentiable probes into the FAD object. Explicitly,

$$\left(M^{\mathsf{LR}}(\langle D, C, p\rangle)\right)_{\mathrm{dom}} = \left\{ \omega : D \to \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}) \;\middle|\; \begin{array}{l} |\mathrm{supp}\,\omega| < \infty \text{ and} \\ \forall\,(x \in \mathrm{supp}\,\omega)\,, \forall\,(v \in \mathbb{E})\,, \omega(x)(v) > 0 \end{array} \right\}. \tag{24}$$

[4] For the corresponding p-dimensional barycentre semimodule, one takes $\langle \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}^p)\,, \langle \mathbb{R}^p, T^*\mathbb{R}^p\rangle\,, \mathrm{dv}_{\mathbb{R}^p}\rangle$.

Thus the domain part consists of finitely supported $\mathrm{Diff}\,(\mathbb{E},\mathbb{R})$-valued weight distributions on D, with strictly positive non-zero weights at every probe point. On morphisms, it is given by the usual pushforward of finite support:

$$\Big(M^{\mathsf{LR}}(\langle u,v\rangle)\Big)_{\mathrm{dom}} : \Big(M^{\mathsf{LR}}(\langle D,C,p\rangle)\Big)_{\mathrm{dom}} \to \Big(M^{\mathsf{LR}}(\langle D',C',p'\rangle)\Big)_{\mathrm{dom}} \quad \text{s.t.} \quad \omega \mapsto \left(y \mapsto \sum_{\substack{x\in\mathrm{supp}\,\omega \\ u(x)=y}} \omega(x) \right). \tag{25}$$

Let X denote the **Set**-component of C. When $\mathbb{E}=\mathbb{R}$, the domain set $\Big(M^{\mathsf{LR}}(\langle D,C,p\rangle)\Big)_{\mathrm{dom}}$ can be read as the class of differentiable curves $h:\mathbb{E}\to M(X)$ selected by the logical relation. Locally, such a curve is represented by a finite weighted presentation

$$h(v) = \mathrm{categorical}\Big(\langle\langle f_j(v)\,, g_j(v)\rangle\rangle_{j=1}^{n}\Big)\,,$$

where the curves $f_j:\mathbb{E}\to X$ are represented by elements of D through p, and the functions $g_j:\mathbb{E}\to\mathbb{R}$ are differentiable. The logical relation simultaneously records the corresponding transposed derivatives.

We have therefore constructed the associated FAD monad M^{LR} on the logical-relations category. It is compatible with the codomain projection to $\mathbf{Set}\times\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, while its domain part selects the differentiable probes into finite atomic distributions, including differentiable atom data and differentiable log-weights. This completes the semantic preparation for the syntactic categories: the source language will be interpreted in **Set**, the differentiated target semantics in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and the correctness argument in LR. The next section introduces the syntactic category of the source language and the structure that these semantic interpretations must preserve.

3 Syntactic categories

The CHAD perspective is to formulate automatic differentiation as a semantic construction. Rather than defining a derivative transformation merely as an *ad hoc* transformation, one first identifies the categorical structure freely generated by the language and then defines a functor preserving that structure. In this way, a code transformation is controlled by a categorical transformation. The syntactic category is the object that makes this passage precise.

Given a type theory, its syntactic category has as objects the types generated by the theory[5] and as morphisms the α-equivalence classes of programs. Composition is given by capture-avoiding substitution. The categorical structure of the syntactic category records the corresponding type-forming operations: for instance, products, coproducts, and exponentials interpret product, sum, and function types, respectively. An interpretation of the theory in a category X is then a functor from the syntactic category to X preserving the chosen structure.

In this paper we use two type theories, the source language and the target language. The source language is the language of programs to be differentiated. The target language extends it with the additional structure needed to express reverse-mode derivatives. The CHAD transformation sends source programs to target programs by the corresponding structure-preserving functor between their syntactic categories.

[5]This assumes that the theory has product types. Without product types, one can instead work with the usual construction based on equivalence classes of contexts.

3.1 Source language

In this section, we describe the *source language*. Its syntactic category is the free distributive category generated by basic types real^n, $n \in \mathbb{N}$, and basic built-in operations op, equipped with a chosen strong monad, a categorical operation, and a handler algebra.

Types in this language are given according to the following context-free grammar,

$$\tau, \sigma, \rho ::= 0 \mid 1 \mid \tau \times \sigma \mid \tau \sqcup \sigma \mid \mathsf{real}^k \; (k \in \mathbb{N}) \mid \mathrm{M}\,\tau \tag{26}$$

with real being used as abbreviation for real^1.

For terms, we denote variables by $x, y, z, \ldots$ and contexts as $\Gamma = x_1 : \tau_1, \cdots, x_n : \tau_n$. Term construction is defined by the following grammar

$$\begin{aligned} t, s, r ::= {} & x \mid \langle\,\rangle \mid \langle t, s\rangle \mid \mathrm{pr}_1\, t \mid \mathrm{pr}_2\, t \mid \mathrm{inl}\, t \mid \mathrm{inr}\, t \\ & \mid \mathsf{case}\; t \;\mathsf{of}\; \mathsf{inl}(x) \Rightarrow s \mid \mathsf{inr}(y) \Rightarrow r \mid \mathrm{abort}(t) \\ & \mid \mathsf{let}\; x = t \;\mathsf{in}\; s \mid \mathsf{op}(t_1, \ldots, t_k) \mid \mathrm{return}(t) \\ & \mid \mathrm{bind}(t; x.s) \mid \mathrm{categorical}(\langle\langle t_i, w_i\rangle\rangle_{i=1}^n) \mid \mathrm{E}_\nu(t) \end{aligned} \tag{27}$$

where $\nu = \mathsf{real}^p$, for some $p \in \mathbb{N}$, is a fixed type on which the handler algebra E_ν is defined, and op ranges over a family of chosen built-in operations, each with a signature $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$, representing the basic operations available in the source language. This family is required to include all affine maps and pointwise linear algebra on products and sums, but extra operations may be added depending on the use case: a differential equation solver might require Bessel functions and spherical harmonics, while a signal analyser might require trigonometric functions, for example.

Typing judgements for the 0, 1, product, and coproduct types are standard, and they may be found in Appendix A. Any operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$ defines a typing rule

$$\frac{\Gamma \vdash t_j : \mathsf{real}^{n_j} \quad (j \in \{1, \ldots, k\})}{\Gamma \vdash \mathsf{op}(t_1, \ldots, t_k) : \mathsf{real}^m}\,\textsc{Operations}$$

For types of the form $\mathrm{M}\,\tau$, we add the following extra rules

$$\frac{\Gamma \vdash t : \mathrm{M}\,\nu}{\Gamma \vdash \mathrm{E}_\nu(t) : \nu}\,\mathrm{M\,E} \qquad \frac{\Gamma \vdash t : \mathrm{M}\,\sigma \quad \Gamma, x : \sigma \vdash s : \mathrm{M}\,\tau}{\Gamma \vdash \mathrm{bind}(t; x.s) : \mathrm{M}\,\tau}\,\mathrm{M\,I1}$$
$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathrm{return}(t) : \mathrm{M}\,\tau}\,\mathrm{M\,I2} \qquad \frac{\Gamma \vdash t_i : \tau \quad \Gamma \vdash w_i : \mathsf{real} \quad (i \in \{1, \cdots, n\})}{\Gamma \vdash \mathrm{categorical}(\langle\langle t_i, w_i\rangle\rangle_{i=1}^n) : \mathrm{M}\,\tau}\,\mathrm{M\,I3} \tag{28}$$

Finally, we add equations. In addition to the standard equational theory for the 0, 1, product, and coproduct types, we add equations for the bind, return, and algebra constructions. First, the Kleisli laws for the bind and return constructions are

$$\begin{gathered} \mathrm{bind}(\mathrm{return}(t)\,; x.s) \equiv s[t/x] \qquad \mathrm{bind}(t; x.\,\mathrm{return}(x)) \equiv t \\ \mathrm{bind}(\mathrm{bind}(t; x.s)\,; y.r) \equiv \mathrm{bind}(t; x.\,\mathrm{bind}(s; y.r)) \end{gathered} \tag{29}$$

where we use $s[t/x]$ for the capture-avoiding substitution of the term t for the variable x in the expression s.

There are also equational rules for the handler algebra E_ν, which ensure that it behaves as an Eilenberg-Moore algebra for M over the type $\nu = \mathsf{real}^p$,

$$\mathrm{E}_\nu(\mathrm{return}(v)) \equiv v \qquad \mathrm{E}_\nu(\mathrm{bind}(t; x.u)) \equiv \mathrm{E}_\nu(\mathrm{bind}(t; x.\,\mathrm{return}(\mathrm{E}_\nu(u)))) \tag{30}$$

We define the category Syn as the syntactic category of this source language. In categorical terms, Syn can be seen as the freely generated distributive category on the objects real^n, $n \in \mathbb{N}$, and operations op, equipped with a fixed strong monad, a chosen categorical operation, and a chosen Eilenberg–Moore algebra E_ν over some object $\nu = \mathsf{real}^p$ for some $p \in \mathbb{N}$.

Given any other distributive category X, with a strong monad $M^{\mathsf{X}} : \mathsf{X} \to \mathsf{X}$, chosen categorical constructors, and an Eilenberg–Moore algebra $\mathrm{alg} : M^{\mathsf{X}}(V) \to V$ for some object V, we may define an interpretation functor $[\![\cdot]\!] : \mathsf{Syn} \to \mathsf{X}$ by choosing, for each $n \in \mathbb{N}$, an object X_n in X such that $X_p = V$. The object map of this functor is given by the type interpretations, which are inductively defined on the structure of the types as

$$\begin{array}{ccc} [\![0]\!] = 0_{\mathsf{X}} & [\![1]\!] = 1_{\mathsf{X}} & [\![\mathsf{real}^n]\!] = X_n \\ [\![\tau \times \sigma]\!] = [\![\tau]\!] \times [\![\sigma]\!] & [\![\tau \sqcup \sigma]\!] = [\![\tau]\!] \amalg [\![\sigma]\!] & [\![\mathrm{M}\,\tau]\!] = M^{\mathsf{X}}([\![\tau]\!]) \end{array} \tag{31}$$

where 0_{X} and 1_{X} denote chosen initial and terminal objects in X, and $\times$ and $\amalg$ denote chosen product and coproduct structures, respectively.

Contexts are interpreted as products of the underlying types, that is, $[\![x_1 : \tau_1, \cdots, x_n : \tau_n]\!] = \prod_{i=1}^n [\![\tau_i]\!]$, and a typing judgement $\Gamma \vdash t : \tau$ is then interpreted as a morphism $[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.

Variables, constants, and constructors for the distributive structure are interpreted in the standard way (see Appendix A). A built-in operator $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$ must be associated with a morphism $[\![\mathsf{op}]\!] : \prod_{j=1}^k X_{n_j} \to X_m$, so that

$$[\![\Gamma \vdash \mathsf{op}(t_1, \cdots, t_k) : \mathsf{real}^m]\!] = [\![\mathsf{op}]\!] \circ \langle [\![\Gamma \vdash t_j : \mathsf{real}^{n_j}]\!] \rangle_{j=1}^k : [\![\Gamma]\!] \to X_m \tag{32}$$

and the monadic operations are interpreted in terms of the unit η^{X}, multiplication μ^{X}, and strength st^{X} of the monad M^{X},

$$\begin{aligned} [\![\Gamma \vdash \mathrm{return}(t) : \mathrm{M}\,\tau]\!] &= \left(\eta^{\mathsf{X}}\right)_{[\![\tau]\!]} \circ [\![\Gamma \vdash t : \tau]\!] \\ [\![\Gamma \vdash \mathrm{bind}(t; x.s) : \mathrm{M}\,\tau]\!] &= \mu^{\mathsf{X}}_{[\![\tau]\!]} \circ M^{\mathsf{X}}([\![\Gamma, x : \sigma \vdash s : \mathrm{M}\,\tau]\!]) \circ \mathrm{st}^{\mathsf{X}}_{[\![\Gamma]\!],[\![\sigma]\!]} \circ \left\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \vdash t : \mathrm{M}\,\sigma]\!] \right\rangle \end{aligned}$$

Terms constructed with categorical are interpreted using the chosen morphisms $\mathrm{categorical}^{\mathsf{X}}_{\tau,n}$.

$$[\![\Gamma \vdash \mathrm{categorical}(\langle\langle t_i, w_i \rangle\rangle_{i=1}^n) : \mathrm{M}\,\tau]\!] = \mathrm{categorical}^{\mathsf{X}}_{\tau,n} \circ \langle\langle [\![\Gamma \vdash t_i : \tau]\!], [\![\Gamma \vdash w_i : \mathsf{real}]\!] \rangle\rangle_{i=1}^n \,.$$

For the FAD language treated here, no additional equations for these constructors are imposed.

Finally, the handler algebra E_ν is interpreted as the chosen algebra on X_p,

$$[\![\Gamma \vdash \mathrm{E}_\nu(t) : \nu]\!] = \mathrm{alg} \circ [\![\Gamma \vdash t : \mathrm{M}\,\nu]\!] \tag{33}$$

Following this prescription, one can prove the following *soundness theorem*,

Lemma 3.1 (Soundness of source language interpretations)**.** *If $\Gamma \vdash t \equiv s : \tau$ is derivable in the equational theory above, then $[\![\Gamma \vdash t : \tau]\!] = [\![\Gamma \vdash s : \tau]\!]$ as morphisms $[\![\Gamma]\!] \to [\![\tau]\!]$ in X.*

We may now state the universal property of the syntactic category of the source language.

Lemma 3.2 (Universal property of Syn)**.** *Let X be any distributive category equipped with*

- *A strong monad* $\left\langle M^{\mathsf{X}}, \eta^{\mathsf{X}}, \mu^{\mathsf{X}}, \mathrm{st}^{\mathsf{X}} \right\rangle$*;*
- *For every* $n \in \mathbb{N}$*, chosen objects* X_n*;*
- *An Eilenberg-Moore algebra* $\mathrm{alg} : M^{\mathsf{X}}(X_p) \to X_p$ *over the object* X_p*;*
- *A suitably typed interpretation* $[\![\mathsf{op}]\!]$ *of every built-in operation* op*;*
- *For each* $n \in \mathbb{N}$ *and each object* A *interpreting a type* τ*, an arrow* $\mathrm{categorical}^{\mathsf{X}}_{\tau,n} : (A \times X_1)^n \to M^{\mathsf{X}}(A)$*, used to interpret* categorical *terms built from lists of length* n *of atoms of type* τ *and weights in* X_1*.*

Then there exists a unique strict functor $[\![\cdot]\!] : \mathsf{Syn} \to \mathsf{X}$ *preserving finite products and coproducts, the strong monad structure, the handler algebra, and* categorical *as above, and sending every equation of* Syn *to an identity in* X*.*

The proof of existence and uniqueness follows from the prescribed construction above.

3.2 Interpretation of the source language in Set

Applying the universal property of the source language (Lemma 3.2) to the data below gives a unique interpretation functor $[\![\cdot]\!]_{\mathbf{Set}} : \mathsf{Syn} \to \mathbf{Set}$ interpreting the source language in the category of sets. For every $n \in \mathbb{N}$, this interpretation is characterised by

$$[\![\mathsf{real}^n]\!]_{\mathbf{Set}} := \mathbb{R}^n \tag{34}$$

and, for any built-in operator $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$ in the source language, we interpret op as a differentiable function

$$[\![\mathsf{op}]\!]_{\mathbf{Set}} : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k} \to \mathbb{R}^m. \tag{35}$$

The strong monad $M^{\mathbf{Set}}$ is the FAD monad M, with its unit, multiplication, and strength. The handler algebra for $\nu = \mathsf{real}^p$ is interpreted as the expectation algebra

$$\mathrm{alg}^{\mathbf{Set}}_{\nu} : M(\mathbb{R}^p) \to \mathbb{R}^p, \qquad w \mapsto \sum_{x \in \mathrm{supp}\, w} x \cdot w(x)\,.$$

Finally, categorical is interpreted using the construction from Equation (4): for each type τ, we use

$$\mathrm{categorical}^{\mathbf{Set}}_{\tau} : \coprod_{n \in \mathbb{N}} ([\![\tau]\!]_{\mathbf{Set}} \times \mathbb{R})^n \to M([\![\tau]\!]_{\mathbf{Set}})\,.$$

3.3 Target language

Code written in the source language (Section 3.1) can be automatically differentiated using the CHAD algorithm. The transformed code is written in a separate language, called the *target language*.

The target language is dependently typed, split into a cartesian type theory and, for every context of cartesian types, a linear type theory. Semantically, the cartesian fragment of the target language allows one to reason about base spaces, while the linear type theories describe cotangents.

We denote by CSyn the term model of the cartesian fragment of the target language. Crucially, Syn can be seen as a full subcategory of CSyn. For each context Γ of cartesian types, we denote by $\mathsf{LSyn}[\Gamma]$ the syntactic category of the associated linear type theory. Since any substitution $\Gamma \to \Gamma'$ between cartesian contexts Γ induces a functor $\mathsf{LSyn}[\Gamma'] \to \mathsf{LSyn}[\Gamma]$, we have a strict indexed category $\mathsf{LSyn} : \mathsf{CSyn}^{\mathrm{op}} \to \mathbf{Cat}$. The syntactic category of the target language is then given by the Grothendieck construction Tgt (see Section 2.3 of Lucatelli Nunes and Vákár (2023)). An explicit description of objects and morphisms in this category can be seen in Section 3.4.

As in the source language, we write τ, σ, ρ for the *cartesian types*, and reserve underlined Greek letters $\underline{\tau}, \underline{\sigma}, \underline{\rho}$ for linear types. We then have *cartesian typing judgements* of the form $\Gamma \vdash t : \tau$ and *linear typing judgements* of the form $\Gamma; v : \underline{\sigma} \vdash s : \underline{\rho}$.

We can divide the description of the target language into three layers:

- **Non-monadic core**: deterministic operations. For brevity, we omit the description of this layer here (see Appendix B).
- **Oplax strong-monad core**: rules for a strong monad once a compatible cotangent lift has been supplied.
- **Distribution and handler**: language features that are specific to the FAD monad.

In doing so, we isolate the data needed to apply CHAD to effects.

Cartesian types are described by the following grammar:

$$\tau, \sigma, \rho ::= (\text{As in Equation (26)}) \mid \underline{\sigma} \multimap \underline{\rho} \mid \Sigma\, (x : \sigma)\, .\underline{\tau}' \tag{36}$$

meaning that, in the cartesian fragment, we also have function types between linear types and dependent sums. For linear types, we use the grammar

$$\underline{\tau}, \underline{\sigma}, \underline{\rho} ::= \underline{\mathsf{real}^k}\ \ (k \in \mathbb{N}) \mid \underline{1} \mid \underline{\tau} \times \underline{\sigma} \mid \mathrm{case}\ t\ \mathrm{of}\ \{\mathrm{in}_i\, x_i \mapsto \underline{\sigma}_i\}_{i=1}^{n} \mid \mathrm{M}^{w}_{p:\mathrm{M}\,\tau}\, [x \mapsto \underline{\sigma}] \tag{37}$$

meaning that the base Euclidean-space types have associated linear types (in the sense that the cotangent space of a Euclidean space at any point can be identified with the base space itself), as well as the fibre-wise zero object 1. Moreover, we allow fibre-wise binary biproducts $\times$ and families of linear types indexed by finite disjunctions, and we introduce the cotangent linear type associated with monadic types.

We also distinguish *cartesian* and *linear terms*. The cartesian terms are described by the grammar

$$t, s, r ::= (\text{As in Equation (27)}) \mid \langle a, t \rangle \mid \mathrm{case}\ p\ \mathrm{of}\ \langle x, y \rangle \mapsto s \mid \lambda v\ .\ s \tag{38}$$

where the constructor $\langle a, t \rangle$ and the eliminator $\mathrm{case}\ p\ \mathrm{of}\ \langle x, y \rangle \mapsto s$ are used in the typing rules for the cartesian Σ-type, and $\lambda v\ .\ s$ is used for abstraction over the unique linear variable v.

For linear terms, we use the following grammar:

$$\begin{aligned} t, s, r ::= {} & v \mid \mathsf{let}\ v = t\ \mathsf{in}\ s \mid \langle t, s \rangle \mid \mathrm{pr}_i\, t \mid 0 \mid t + s \mid r\, t \\ & \mid \mathsf{op}(t_1, \cdots, t_k)(v) \mid \mathrm{return}^{w}_{t}(s) \mid \mathrm{bind}^{w}_{s;x\ .\ t}(r; u) \\ & \mid \mathrm{categorical}^{w}_{\langle\langle t_i, w_i \rangle\rangle_{i=1}^{n}}(\langle\langle t'_i, w'_i \rangle\rangle_{i=1}^{n}) \mid \mathrm{E}^{w}_{\nu,t}(s) \end{aligned} \tag{39}$$

The three lines correspond to the three layers: the first line consists of the non-monadic core operations (linear variables, let bindings, pairings and projections, the zero vector, sums, and $r\,t$ for linear application with r a cartesian term of type $\underline{\sigma} \multimap \underline{\rho}$). The second line has the derivatives of the built-in operations op of the source language, as well as the linear fibre components of the monad operations return and bind. Finally, the third line has the linear fibre components of categorical and of the handler algebra E_{ν}.

The typing rules for the non-monadic core are standard, so we omit them here (see Appendix B).

There are two typing rules for the monadic core, corresponding to the return and bind operations. For return, the rule is

$$\frac{\Gamma \vdash t : \tau \quad \Gamma, x : \tau \vdash \underline{\sigma} \text{ type} \quad \Gamma; v : \underline{\sigma}[t/x] \vdash s : \underline{\rho}}{\Gamma; v : \mathrm{M}^{w}_{\mathrm{return}(t):\mathrm{M}(\tau)}\,[x \mapsto \underline{\sigma}] \vdash \mathrm{return}^{w}_{t}(s) : \underline{\rho}} \tag{40}$$

and for bind, the rule is

$$\frac{\begin{array}{c}\Gamma, x : \sigma \vdash \underline{\sigma} \text{ type} \qquad \Gamma, y : \tau \vdash \underline{\tau} \text{ type} \qquad \Gamma \vdash \underline{\rho} \text{ type} \qquad \Gamma \vdash s : \mathrm{M}(\sigma) \\ \Gamma, x : \sigma \vdash t : \mathrm{M}(\tau) \quad \Gamma; v : \mathrm{M}^{w}_{s:\mathrm{M}(\sigma)}\,[x \mapsto \underline{\sigma}] \vdash r : \underline{\rho} \quad \Gamma, x : \sigma; v : \mathrm{M}^{w}_{t:\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash u : \underline{\rho} \times \underline{\sigma}\end{array}}{\Gamma; v : \mathrm{M}^{w}_{\mathrm{bind}(s;x\,.\,t):\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash \mathrm{bind}^{w}_{s;x\,.\,t}(r;u) : \underline{\rho}} \tag{41}$$

These typing rules are the target-language interface for the chosen lifted strong monad. For the FAD monad, the lift M^{F} also supplies transpose rules for categorical and for the handler algebra E_{ν}. For categorical, we have, for all $n \in \mathbb{N}$,

$$\frac{\begin{array}{c}\Gamma, x : \tau \vdash \underline{\tau} \text{ type} \qquad \Gamma \vdash \underline{\rho} \text{ type} \\ \Gamma \vdash w_i : \mathsf{real} \quad \Gamma \vdash t_i : \tau \quad \Gamma; v : \underline{\mathsf{real}} \vdash w'_i : \underline{\rho} \quad \Gamma; v : \underline{\tau}[t_i/x] \vdash t'_i : \underline{\rho} \quad (1 \le i \le n)\end{array}}{\Gamma; v : \mathrm{M}^{w}_{\mathrm{categorical}(\langle\langle t_i, w_i\rangle\rangle_{i=1}^{n}):\mathrm{M}(\tau)}\,[x \mapsto \underline{\tau}] \vdash \mathrm{categorical}^{w}_{\langle\langle t_i, w_i\rangle\rangle_{i=1}^{n}}(\langle\langle t'_i, w'_i\rangle\rangle_{i=1}^{n}) : \underline{\rho}} \tag{42}$$

and, for the handler algebra, recalling that $\nu = \mathsf{real}^{p}$, we define $\underline{\nu} := \underline{\mathsf{real}}^{p}$. Then,

$$\frac{\Gamma \vdash t : \mathrm{M}(\nu) \quad \Gamma; v : \mathrm{M}^{w}_{t:\mathrm{M}(\nu)}\,[\_ \mapsto \underline{\nu}] \vdash s : \underline{\rho}}{\Gamma; v : \underline{\nu} \vdash \mathrm{E}^{w}_{\nu,t}(s) : \underline{\rho}} \tag{43}$$

The equations for the cartesian fragment are standard, whereas those for the linear fragment can be seen as originating from a formal derivative procedure over the cartesian base counterparts (see Appendix B). For instance, the right unit law for return and bind (see Equation (29))

$$\mathrm{bind}(s; x.\,\mathrm{return}(x)) \equiv s.$$

When imposed on the cartesian fragment, this law gives rise to the equality

$$\mathrm{bind}^{w}_{s;x.\,\mathrm{return}(x)}(r; \langle 0, \mathrm{return}^{w}_{x}(v)\rangle) \equiv r. \tag{44}$$

3.4 Interpretation of the target language in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$

In Section 3.2, we provided set-based semantics for the source language via the interpretation functor $[\![\cdot]\!]_{\mathbf{Set}} : \mathsf{Syn} \to \mathbf{Set}$. Here we provide concrete semantics for the target language syntactic category Tgt by interpreting it in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ (see Section 2.4).

An object of Tgt may be seen as a pair $\langle p : \tau, \underline{\sigma} \rangle$ (up to α-equivalence) such that $\underline{\sigma}$ is a linear type in $\mathsf{LSyn}[p : \tau]$. A morphism $\langle t, s \rangle : \left\langle p : \tau, \underline{\rho} \right\rangle \to \left\langle q : \sigma, \underline{\chi} \right\rangle$ is a pair of a cartesian arrow $t : p \to q$ in CSyn, and of a linear arrow $s : \underline{\chi}[t/q] \to \underline{\rho}$ in $\mathsf{LSyn}[p : \tau]$.

We start by providing an interpretation of the cartesian fragment CSyn of the target language. This fragment is interpreted in **Set** by extending the interpretation of Syn, as described in Section 3.2. For that reason, we keep the same notation, writing $[\![0]\!]_{\mathbf{Set}} := \emptyset$, $[\![\mathsf{real}^k]\!]_{\mathbf{Set}} := \mathbb{R}^k$, $[\![\tau \times \sigma]\!]_{\mathbf{Set}} := [\![\tau]\!]_{\mathbf{Set}} \times [\![\sigma]\!]_{\mathbf{Set}}$, and so on. The linear interpretation of a linear type under a cartesian context $p : \tau$ is also described inductively over the structure of the linear types, and is described in Appendix C. In particular, we give $[\![\underline{\mathsf{real}^k}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) := \mathbb{R}^k$ and, for any cartesian judgement $p : \tau \vdash q : \mathrm{M}(\rho)$, $[\![\mathrm{M}^w_{q:\mathrm{M}(\rho)}\,[x \mapsto \underline{\sigma}]]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) = \bigoplus_{x \in \mathrm{supp}[\![q]\!]_{\mathbf{Set}}(\gamma)} \left([\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau,x:\rho}(\gamma, x) \oplus \mathbb{R} \right)$ for any cartesian variable $p : \tau$ and $\gamma \in [\![\tau]\!]_{\mathbf{Set}}$.

So, for any object $\langle p : \tau, \underline{\sigma} \rangle$ in Tgt, we define its interpretation in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ as

$$[\![\langle p : \tau, \underline{\sigma} \rangle]\!]_{\mathsf{F}} := \left\langle [\![\tau]\!]_{\mathbf{Set}}, \left(\gamma \mapsto [\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \right) \right\rangle \tag{45}$$

and morphisms are interpreted, displaying the corresponding ordinary **Vect**-map in the fibre, as

$$[\![\langle t, s \rangle]\!]_{\mathsf{F}} := \left\langle [\![t]\!]_{\mathbf{Set}}, \left(\gamma \mapsto \left([\![s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)^{\mathrm{op}} \in \mathbf{Vect}\left([\![\underline{\chi}]\!]^{\mathrm{lin}}_{q:\sigma}([\![t]\!]_{\mathbf{Set}}(\gamma)) \, , [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \right) \right) \right) \right\rangle \tag{46}$$

3.5 CHAD code transformation

The core idea of CHAD is the translation of the automatic differentiation code transformation into a structure-preserving categorical transformation, i.e. a functor. This is a functor $\mathsf{CHAD} : \mathsf{Syn} \to \mathsf{Tgt}$. For each term in context $\Gamma \vdash t : \tau$ in the source language, we get a term under context

$$\mathsf{CHAD}(\Gamma)_1 \vdash \mathsf{CHAD}[\Gamma](t) : \Sigma\,(p : \mathsf{CHAD}(\tau)_1)\ .\ (\mathsf{CHAD}(\tau)_2 \multimap \mathsf{CHAD}(\Gamma)_2) \tag{47}$$

In other words, the term under context $\Gamma \vdash t : \tau$ is translated as a term of this dependent-sum type: its first projection has type $\mathsf{CHAD}(\tau)_1$, and the corresponding fibres are given by the function types $\mathsf{CHAD}(\tau)_2 \multimap \mathsf{CHAD}(\Gamma)_2$, corresponding to transposed derivatives (backpropagators).

Similar to denotational semantics, type and term transformations are defined inductively over the structure of the types and terms. The definitions for base types, as well as for products and coproducts is straightforward, and can be seen in Appendix D. For the monadic types, we define

$$\mathsf{CHAD}(\mathrm{M}(\tau))_1 := \mathrm{M}(\mathsf{CHAD}(\tau)_1) \quad \text{and} \quad \mathsf{CHAD}(\mathrm{M}(\tau))_2 := \mathrm{M}^w_{p:\mathrm{M}\left(\mathsf{CHAD}(\tau)_1\right)}\,[x \mapsto \mathsf{CHAD}(\tau)_2\,[x/p]] \tag{48}$$

and the terms related to the monadic structure have their derivatives defined as

$$\mathsf{CHAD}[\Gamma](\mathrm{return}(t)) := \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b \rangle \to \left\langle \mathrm{return}(p)\,, \lambda v\ .\ \mathrm{return}^w_p(b(v)) \right\rangle \tag{49}$$

$$\begin{aligned}\mathsf{CHAD}[\Gamma](\mathrm{bind}(t; x\ .\ s)) := \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \langle p, b \rangle \to \mathrm{let}\ \ r = \mathrm{case}\ \ \mathsf{CHAD}[\Gamma, x : \sigma](s)\ \ \mathrm{of}\ \ \langle q, c \rangle \to q\ \ \mathrm{in} \\ \left\langle \mathrm{bind}(p; x\ .\ r)\,, \lambda v\ .\ \mathrm{bind}^w_{p;x\ .\ r}(b(v)\,; \mathrm{case}\ \ \mathsf{CHAD}[\Gamma, x : \sigma](s)\ \ \mathrm{of}\ \ \langle q, c \rangle \to c(v)) \right\rangle\end{aligned} \tag{50}$$

while the term constructors specific to the FAD monad have their code transformations defined as

$$\begin{aligned}
&\mathsf{CHAD}[\Gamma](\mathrm{categorical}(\langle\langle t_i, w_i\rangle\rangle_{i=1}^n)) := \\
&\qquad \mathrm{case}\ \mathsf{CHAD}[\Gamma](t_1)\ \mathrm{of}\ \langle u_1, u_1'\rangle \to \cdots \mathrm{case}\ \mathsf{CHAD}[\Gamma](t_n)\ \mathrm{of}\ \langle u_n, u_n'\rangle \to \\
&\qquad \mathrm{case}\ \mathsf{CHAD}[\Gamma](w_1)\ \mathrm{of}\ \langle a_1, a_1'\rangle \to \cdots \mathrm{case}\ \mathsf{CHAD}[\Gamma](w_n)\ \mathrm{of}\ \langle a_n, a_n'\rangle \to \\
&\qquad\qquad \Big\langle \mathrm{categorical}(\langle\langle u_i, a_i\rangle\rangle_{i=1}^n), \lambda v\ .\ \mathrm{categorical}^w_{\langle\langle u_i,a_i\rangle\rangle_{i=1}^n}(\langle\langle u_i'(v), a_i'(v)\rangle\rangle_{i=1}^n)\Big\rangle
\end{aligned} \tag{51}$$

$$\mathsf{CHAD}[\Gamma](\mathrm{E}_\nu(t)) := \mathrm{case}\ \mathsf{CHAD}[\Gamma](t)\ \mathrm{of}\ \langle p, b\rangle \to \Big\langle \mathrm{E}_\nu(p), \lambda v\ .\ \mathrm{E}^w_{\nu,p}(b(v))\Big\rangle \tag{52}$$

4 Logical relations and correctness

4.1 Interpretation of source language in the category of logical relations

As in **Set**, we can use the universal property of the syntactic category of the source language to construct an interpretation of the source language in the category LR of logical relations. It is enough to specify the interpretations of the elementary real types, the built-in operations, the monadic type constructor, categorical, and the handler algebra.

For the elementary real number types, we define, for all $n \in \mathbb{N}$,

$$[\![\mathsf{real}^n]\!]_{\mathsf{LR}} := \langle \mathrm{Diff}(\mathbb{E}, \mathbb{R}^n), \langle \mathbb{R}^n, T^*\mathbb{R}^n\rangle, \mathrm{dv}_{\mathbb{R}^n}\rangle \tag{53}$$

and, for any built-in operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$ in the source language, its domain part is given by postcomposition with $[\![\mathsf{op}]\!]_{\mathbf{Set}}$, while its codomain part is the base function together with its transposed derivative,

$$\begin{aligned}
([\![\mathsf{op}]\!]_{\mathsf{LR}})_{\mathrm{dom}} &:= ([\![\mathsf{op}]\!]_{\mathbf{Set}} \circ -) : \underbrace{\mathrm{Diff}(\mathbb{E}, \mathbb{R}^{n_1}) \times \cdots \times \mathrm{Diff}(\mathbb{E}, \mathbb{R}^{n_k})}_{\text{identified with } \mathrm{Diff}(\mathbb{E}, \mathbb{R}^{n_1}\times\cdots\times\mathbb{R}^{n_k})} \to \mathrm{Diff}(\mathbb{E}, \mathbb{R}^m) \\
([\![\mathsf{op}]\!]_{\mathsf{LR}})_{\mathrm{cod}} &:= \langle [\![\mathsf{op}]\!]_{\mathbf{Set}}, \langle [\![\mathsf{op}]\!]_{\mathbf{Set}}, \boldsymbol{\nabla}[\![\mathsf{op}]\!]_{\mathbf{Set}}\rangle\rangle : \left\langle \prod_{j=1}^k \mathbb{R}^{n_j}, \prod_{j=1}^k T^*\mathbb{R}^{n_j} \right\rangle \to \langle \mathbb{R}^m, T^*\mathbb{R}^m\rangle
\end{aligned} \tag{54}$$

The strong monad is taken to be the FAD monad M^{LR} on LR (see Section 2.5), with the corresponding interpretations of unit, multiplication, and strength. The interpretation of categorical is given by the factorisation procedure on LR described in Section 2.5. For $\nu = \mathsf{real}^p$, the handler algebra E_ν is interpreted by the corresponding M^{LR}-algebra $\mathrm{alg}^{\mathsf{LR}}_\nu : M^{\mathsf{LR}}([\![\nu]\!]_{\mathsf{LR}}) \to [\![\nu]\!]_{\mathsf{LR}}$, whose codomain part is the pair consisting of the expectation algebra in **Set** and the lifted expectation algebra in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and whose domain part sends a differentiable function of FADs to the differentiable expectation function obtained by applying that algebra pointwise.

It then follows from the universal property of the syntactic category of the source language (Lemma 3.2) that these data define a unique interpretation functor $[\![\cdot]\!]_{\mathsf{LR}} : \mathsf{Syn} \to \mathsf{LR}$.

Crucially, by composing this interpretation functor with $U_{\mathbf{Set}} := \Pi_1 \circ \Pi_2 \circ U$, where $U : \mathsf{LR} \to \mathbf{Set} \times \mathsf{C}$ is the comma-category forgetful functor, $\Pi_2 : \mathbf{Set} \times \mathsf{C} \to \mathsf{C}$ is the second projection into $\mathsf{C} = \mathbf{Set} \times \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and $\Pi_1 : \mathsf{C} \to \mathbf{Set}$ is the first projection, we get another distributive functor. Indeed, U creates limits and colimits, Π_1 and Π_2 preserve them, and the codomain part of M^{LR} is fibred over the FAD monads in **Set** and $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$.

This functor $U_{\mathbf{Set}} \circ [\![\cdot]\!]_{\mathsf{LR}}$ satisfies the same data as the **Set**-interpretation, in particular Equations (34) and (35). Hence, by uniqueness,

$$U_{\mathbf{Set}} \circ [\![\cdot]\!]_{\mathsf{LR}} = [\![\cdot]\!]_{\mathbf{Set}} \tag{55}$$

Analogously, composing $[\![\cdot]\!]_{\mathsf{LR}} : \mathsf{Syn} \to \mathsf{LR}$ with $U_{\mathsf{F}} := \Pi_2 \circ \Pi_2 \circ U : \mathsf{LR} \to \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, we get a distributive functor $\mathsf{Syn} \to \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ that interprets the monad type constructor (and its associated unit and multiplication) as the FAD monad in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. It follows from the universal property of Syn that

$$U_{\mathsf{F}} \circ [\![\cdot]\!]_{\mathsf{LR}} = [\![\cdot]\!]_{\mathsf{F}} \circ \mathsf{CHAD} \tag{56}$$

4.2 Correctness Theorem

In this section, we use the previously defined machinery to prove the following correctness theorem.

Theorem 4.1 (Correctness Theorem)**.** *For any well-typed source term of the form* $x : \mathsf{real}^n \vdash t : \mathsf{real}^m$*, where* $n, m \in \mathbb{N}$*, we have that*

$$[\![t]\!]_{\mathbf{Set}} : \mathbb{R}^n \to \mathbb{R}^m \text{ is differentiable and } [\![\mathsf{CHAD}(t)]\!]_{\mathsf{F}} = \langle [\![t]\!]_{\mathbf{Set}}, \boldsymbol{\nabla} [\![t]\!]_{\mathbf{Set}} \rangle\,. \tag{57}$$

To prove this, fix a well-typed source term $x : \mathsf{real}^n \vdash t : \mathsf{real}^m$. Interpreting this judgement in the logical-relations category via the functor $[\![\cdot]\!]_{\mathsf{LR}} : \mathsf{Syn} \to \mathsf{LR}$, we get a morphism

$$[\![t]\!]_{\mathsf{LR}} : \langle \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}^n)\,, \langle \mathbb{R}^n, T^*\mathbb{R}^n \rangle\,, \mathrm{dv}_{\mathbb{R}^n} \rangle \to \langle \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}^m)\,, \langle \mathbb{R}^m, T^*\mathbb{R}^m \rangle\,, \mathrm{dv}_{\mathbb{R}^m} \rangle$$

in LR. By the defining commutative square for morphisms in a comma category,

$$G(([\![t]\!]_{\mathsf{LR}})_{\mathrm{cod}}) \circ \mathrm{dv}_{\mathbb{R}^n} = \mathrm{dv}_{\mathbb{R}^m} \circ ([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}} : \mathrm{Diff}\,(\mathbb{E}, \mathbb{R}^n) \to \mathsf{C}(\langle \mathbb{E}, T^*\mathbb{E} \rangle\,, \langle \mathbb{R}^m, T^*\mathbb{R}^m \rangle)\,.$$

Since $\mathbb{E}$ was an arbitrary finite-dimensional real vector space, we may take $\mathbb{E} = \mathbb{R}^n$. In this case, we apply both sides of the equality to $\mathrm{id}_{\mathbb{R}^n} \in \mathrm{Diff}\,(\mathbb{R}^n, \mathbb{R}^n)$. Using $\mathrm{dv}_{\mathbb{R}^n}(\mathrm{id}_{\mathbb{R}^n}) = \mathrm{id}_{\langle \mathbb{R}^n, T^*\mathbb{R}^n \rangle}$, the left-hand side is

$$G(([\![t]\!]_{\mathsf{LR}})_{\mathrm{cod}}) \circ \mathrm{dv}_{\mathbb{R}^n}(\mathrm{id}_{\mathbb{R}^n}) = ([\![t]\!]_{\mathsf{LR}})_{\mathrm{cod}} \circ \mathrm{id}_{\langle \mathbb{R}^n, T^*\mathbb{R}^n \rangle} = ([\![t]\!]_{\mathsf{LR}})_{\mathrm{cod}} = \Pi_2 \circ U([\![t]\!]_{\mathsf{LR}})\,,$$

and the right-hand side is

$$\mathrm{dv}_{\mathbb{R}^m} \circ ([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n}) = \langle ([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n})\,, \langle ([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n})\,, \boldsymbol{\nabla}\,(([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n})) \rangle \rangle\,.$$

Applying the first projection $\Pi_1 : \mathsf{C} \to \mathbf{Set}$ to both expressions, it follows from Equation (55) that

$$([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n}) = \Pi_1 \circ \Pi_2 \circ U([\![t]\!]_{\mathsf{LR}}) = U_{\mathbf{Set}}([\![t]\!]_{\mathsf{LR}}) = [\![t]\!]_{\mathbf{Set}}$$

and thus, we can prove Theorem 4.1, by using Equation (56):

$$[\![\mathsf{CHAD}(t)]\!]_{\mathsf{F}} = U_{\mathsf{F}}([\![t]\!]_{\mathsf{LR}}) = \Pi_2(\Pi_2 \circ U([\![t]\!]_{\mathsf{LR}})) = \langle ([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n}), \boldsymbol{\nabla}\,(([\![t]\!]_{\mathsf{LR}})_{\mathrm{dom}}(\mathrm{id}_{\mathbb{R}^n})) \rangle = \langle [\![t]\!]_{\mathbf{Set}}, \boldsymbol{\nabla} [\![t]\!]_{\mathbf{Set}} \rangle\,.$$

5 Returning to the example

In this section, we return to the stochastic optimisation problem discussed in Section 1.1. The goal is to encode it in the source language and then apply the code transformation. We then study how the transformed code relates to the mathematical description of the derivative semantics.

5.1 Encoding the example in the source language

As a first step in encoding the example in the source language, we must set up the family of built-in operations. In addition to the basic operations for representing affine maps and linear algebra, we also add five nullary operations (i.e. constants) $s_m, s_p, m_0, p_0, P : \mathsf{real}$ and two unary operations $\mathrm{sig}, \mathrm{lsig} : \mathsf{real} \to \mathsf{real}$, representing the sigmoid and the logarithm of the sigmoid function, respectively. Then, in the context $\Gamma = p : \mathsf{real}, m : \mathsf{real}$, the program is given by the term

$$\begin{aligned} &\mathrm{let}\ \ s = \mathrm{categorical}\left(\langle\langle 0, \mathrm{lsig}(-s_m \cdot (m - m_0))\rangle\right., \\ &\qquad\qquad\qquad \left.\langle P \cdot \mathrm{sig}(-s_p \cdot (p - p_0))\right., \mathrm{lsig}(s_m \cdot (m - m_0))\rangle\rangle) \\ &\mathrm{in}\ \ \mathrm{let}\ \ t = \mathrm{bind}(s; x.\, \mathrm{return}(p \cdot x - m))\ \ \mathrm{in}\ \ \mathrm{E}_{\mathsf{real}}(t) \end{aligned} \tag{58}$$

where we have chosen $\nu = \mathsf{real}$. First, we introduce a distribution s for the two possible total-sales scenarios. Then, we map the function $x \mapsto p \cdot x - m$ over those scenarios using bind and return, giving the profit distribution t. Finally, we compute the expected total profit using the $\mathrm{E}_{\mathsf{real}}$ handler.

In this simple example, the final result is an expected value. However, the source language allows different expected values to be mixed within the same expression. For instance, suppose that one wants to maximise a risk-adjusted profit given by the ratio of the square of the expected profit to the profit variance. The source-language version of this calculation is

$$\begin{aligned} &\mathrm{let}\ \ s = \mathrm{categorical}\left(\langle\langle 0, \mathrm{lsig}(-s_m \cdot (m - m_0))\rangle\right., && \textit{// Sales distribution} \\ &\qquad\qquad \left.\langle P \cdot \mathrm{sig}(-s_p \cdot (p - p_0))\right., \mathrm{lsig}(s_m \cdot (m - m_0))\rangle\rangle) && \\ &\mathrm{in}\ \ \mathrm{let}\ \ t = \mathrm{bind}(s; x.\, \mathrm{return}(p \cdot x - m)) && \textit{// Profit distribution} \\ &\mathrm{in}\ \ \mathrm{let}\ \ a = \mathrm{E}_{\mathsf{real}}(t) && \textit{// Expected profit} \\ &\mathrm{in}\ \ \mathrm{let}\ \ b = \mathrm{bind}\Big(t; x.\, \mathrm{return}\Big(x^2\Big)\Big) && \textit{// Squared profit} \\ &\mathrm{in}\ \ \mathrm{let}\ \ v = \mathrm{E}_{\mathsf{real}}(b) - a^2 && \textit{// Variance} \\ &\mathrm{in}\ \ a^2/v && \end{aligned}$$

The source language also supports more complex combinations of expected values, including expected values of expressions that themselves contain expected values.

5.2 Code transformation

We now consider the differentiated code from the example in Equation (58). The full code can be seen in Appendix E. Here we highlight some of the main features.

Recall that the differentiation is done with respect to the context $\Gamma = p : \mathsf{real}, m : \mathsf{real}$, and we adopt the following abbreviations:

$$\begin{aligned} a_0 &:= 0 & \gamma_0 &:= \mathrm{lsig}(-s_m \cdot (m - m_0)) \\ a_1 &:= P \cdot \mathrm{sig}(-s_p \cdot (p - p_0)) & \gamma_1 &:= \mathrm{lsig}(s_m \cdot (m - m_0)) \end{aligned}$$

The first lines of the differentiated code are case statements that pattern match on

$$\begin{array}{l}\text{case } \langle a_0, \boldsymbol{\nabla} a_0\rangle \text{ of } \langle \theta_0, \theta_0'\rangle \to \text{case } \langle a_1, \boldsymbol{\nabla} a_1\rangle \text{ of } \langle \theta_1, \theta_1'\rangle \to \\ \text{case } \langle \gamma_0, \boldsymbol{\nabla}\gamma_0\rangle \text{ of } \langle \rho_0, \rho_0'\rangle \to \text{case } \langle \gamma_1, \boldsymbol{\nabla}\gamma_1\rangle \text{ of } \langle \rho_1, \rho_1'\rangle \to \\ \quad \Big\langle \text{categorical}\Big(\langle\langle \theta_j, \rho_j\rangle\rangle_{j=0}^1\Big), \lambda v \,.\, \text{categorical}^w_{\langle\langle \theta_j, \rho_j\rangle\rangle_{j=0}^1}\Big(\Big\langle\Big\langle \theta_j'(v), \rho_j'(v)\Big\rangle\Big\rangle_{j=0}^1\Big)\Big\rangle\end{array}$$

These lines correspond to the derivative of categorical. Since $a_i = \theta_i$ and $\gamma_i = \rho_i$, the first entry of the tuple is simply a recalculation of the same categorical distribution of sales. The second entry uses the categorical^w linear term construction to construct a cotangent value combining the transposed derivative of the atoms (θ_i') and of the log-weights (ρ_i'), as is the case with the denotational semantics in $\mathsf{Fam}(\mathbf{Vect}^{\text{op}})$ (Section 2.4).

Another point to highlight is the block concerning the transposed derivative of the handler:

$$\begin{array}{ll}\text{case} & \text{case}\Big\langle t, \lambda v \,.\, \text{coproj}_{\text{idx}(t;\Gamma,s,t)}\, v\Big\rangle \\ & \text{of } \langle \zeta, \zeta'\rangle \to \Big\langle \mathrm{E}_{\mathsf{real}}(\zeta), \lambda v \,.\, \mathrm{E}^w_{\mathsf{real},\zeta}(\zeta'(v))\Big\rangle \\ \text{of} & \langle \epsilon, \epsilon'\rangle \to \langle \epsilon[\delta/t], \lambda v \,.\, \mathsf{let}\ w = \epsilon'[\delta/t](v)\ \mathsf{in}\ \mathrm{pr}_1\, w + \delta'(\mathrm{pr}_2\, w)\rangle\end{array}$$

Here, the transposed derivative appears as the transposed derivative of the variable t with respect to the extended context $\Gamma, s : \mathrm{M}(\mathsf{real}), t : \mathrm{M}(\mathsf{real})$. We then compute its expected value, and use the $\mathrm{E}^w_{\mathsf{real}}$ term constructor to compute the derivative of the handler algebra. In the last line, t is substituted by the δ term, whose definition is given in a surrounding case construction.

6 Beyond probability: discrete-output algebraic effects

The FAD monad is the main case study, but our construction is not intrinsically probabilistic. The reusable data are a strong monad presented by discrete-output operations, a chosen Eilenberg–Moore handler algebra, and compatible lifts of the monad, the operations, and the handler to $\mathsf{Fam}(\mathbf{Vect}^{\text{op}})$ and to the logical-relations category. With these data, the cotangent rules for algebraic operations, generic return and bind, and handlers from Sections 3.1–3.4 have the same form.

For a typed algebraic signature Σ with generic effects $\omega : I_\omega \to TO_\omega$, the parameter types I_ω may be differentiable: $[\![I_\omega]\!]_{\mathsf{F}}$ may have non-trivial cotangent fibres. The direct fibred-polynomial construction requires the operation output type to be finite and discrete $O_\omega = 1 \sqcup \cdots \sqcup 1$ (so $[\![O_\omega]\!]_{\mathsf{F}}$ has trivial fibre). Before quotienting by equations, the free AST monad has the form

$$\mathsf{AST}_\Sigma X \simeq \mu Y.\, X \sqcup \coprod_{\omega\in\Sigma} I_\omega \times \prod_{o\in O_\omega} Y :$$

we take a finite product of continuation subtrees. The lift to $\mathsf{Fam}(\mathbf{Vect}^{\text{op}})$ adds cotangents for the parameter and branches, while the branch index o has zero fibre. (Outputs with non-trivial fibres can destroy fibredness.) Appendix H gives the corresponding generic CHAD theorem.

Finite non-determinism For finite non-determinism, we replace the FAD monad with the finite multiset monad $\mathsf{MFin}X = \coprod_{k\in\mathbb{N}} \left(X^k/\mathfrak{S}_k\right)$, where multiplicity is retained but order is forgotten. From the signature with no equations, we can take the corresponding AST monad as the list monad (where order is relevant). This is analogous to what happens to FADs: where FADs can be seen as lists of point/log-weight pairs modulo collisions and reordering, multisets can be seen as lists modulo reordering. These additional equations are then imposed by an AST-algebra.

For multisets, a AST-algebra (i.e. a List-algebra) is precisely a monoid structure on the underlying object. By making this monoid commutative, one then imposes order-irrelevance. Thus $\langle \mathbb{R}^n, 0, + \rangle$, for example, induces a List-algebra $\mathrm{sum} : \mathrm{List}(\mathbb{R}^n) \to \mathbb{R}^n$ for multiplicity-sensitive finite non-deterministic computations valued in $\mathbb{R}^n$. The ordinary lifted monad also quotients the fibres: over a multiset represented by $\langle x_1, \ldots, x_k \rangle$, the fibre is the stabiliser-invariant part of $\bigoplus_i A_{x_i}$, equivalently one cotangent slot per support point after collisions. Hence repeated indistinguishable outcomes do not keep independent occurrence cotangents. The finite-choice reverse rule is the FAD categorical rule with the log-weight components removed, followed by this quotienting of cotangents.

Exceptions For exceptions, $M_E(X) = X \amalg E$ may be taken as its own AST. In general, a handler on a carrier A is determined by a recovery map $h : E \to A$, giving the algebra $[\mathrm{id}_A, h] : A \amalg E \to A$. To recover the monad from the CPS factorisation, it is enough to use the free algebra on carrier $A = M_E(2) = 2 \amalg E$ as a canonical algebra: monad multiplication $(\mu)_2 : (2 \amalg E) \amalg E \to 2 \amalg E$. If E has continuous structure, a raised exception carries an exception-label cotangent until it is handled, and the handler uses the transposed derivative of the recovery map. Since here the monad and the AST coincide, exceptions fit the same pattern without requiring a quotient construction.

Accumulation For writer accumulation, let $W_M(X) = X \times M$ be the writer monad associated to a monoid M. For the basic case $M = \mathbb{R}^n$ we have a cancellable monoid, and again it may be taken as its own AST. Its algebras are right M-actions $A \times M \to A$, and so for the AST-algebra we use the carrier $\mathbb{R}^n$ with action $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$, $\langle u, v \rangle \mapsto u + v$. The lifted writer monad attaches to $\langle x, r \rangle$ the fibre $A_x \oplus \mathbb{R}^n$, and the handler rule is the transposed derivative of addition.

Remark (source syntax for the additional first-order effects). For finite multiset non-determinism we add operations choose_k with permutation and commutative-monoid laws, with a fold handler for the chosen commutative monoid; operationally, a choice node can be implemented as a parallel fork and the handler as the corresponding join/reduction. For exceptions one may add $\mathrm{raise}(e) : \mathrm{M}\,\tau$, with handlers determined by maps $E \to \tau$. For the $\mathbb{R}^n$-writer monad one may add $\mathrm{tell}(r) : \mathrm{M}\,1$, together with the basic handler $\mathrm{M}\,\mathsf{real}^n \to \mathsf{real}^n$ induced by addition, which consumes the accumulated output. The target language then adds the corresponding linear terms for the transposed derivatives of operations and handlers; the generic return^w and bind^w rules are unchanged. These fork–join and accumulation patterns are common in high-performance code that we want to differentiate.

7 Related work

This work connects probabilistic programming semantics, stochastic gradient estimation, and semantic accounts of AD. The continuation view of distributions has classical roots in

Schwartz's theory of distributions Schwartz (1950). In probabilistic programming, Vákár, Kammar, and Staton construct a probabilistic powerdomain by factoring integration into a continuation monad Vákár et al. (2019), while Kammar and McDermott develop the factorisation-systems technology for monadic lifting and logical relations Kammar and McDermott (2018).

Stochastic gradient estimation is central in machine learning; Mohamed et al. (2020) surveys score, pathwise, and measure-valued estimators. Stochastic computation graphs showed how these compose by AD-like backpropagation Schulman et al. (2015), and Storchastic allows different estimators at different stochastic nodes van Krieken et al. (2021). ADEV is especially close, extending forward-mode AD to expectations of probabilistic programs with discrete and continuous distributions Lew et al. (2023). We contribute the corresponding reverse-mode account: reverse mode exposes cotangent-flow and fibredness issues that do not arise in forward mode. In particular, continuous probability is not merely the discrete construction with sums replaced by integrals; the relevant continuation/image construction need not be fibred over primal outcomes, so we leave it to future work. Compared to ADEV, we also use image factorisation to carve out distributions and their reverse-derivative structure from the continuation monad, and formulate expectation as an algebraic-effect handler Plotkin and Power (2003); Plotkin and Pretnar (2013).

There is also growing work on deterministic reverse AD with algebraic effects and handlers. De Vilhena and Pottier verify a define-by-run reverse-mode AD library based on effect handlers de Vilhena and Pottier (2023), while Sigal studies effect-handler implementations of AD in Frank, OCaml, and related settings Sigal (2021, 2024). Our focus is different: leaving implementation strategy open, with Efficient CHAD as a natural starting point Smeding and Vákár (2024), we give a denotationally correct source transformation for languages whose source terms themselves have algebraic effects. This builds on the CHAD paradigm for pure source languages Vákár (2021); Vákár and Smeding (2022); Lucatelli Nunes and Vákár (2023) and its extension to iteration and partiality Lucatelli Nunes et al. (2025), showing that the technique also applies to impure, discrete-output algebraic effects and handlers.

Funding

This research was supported by the ERC project FoRECAST. The second author was also supported by the Centre for Mathematics of the University of Coimbra (CMUC, doi:10.54499/UID/00324/2025), funded by the Fundação para a Ciência e a Tecnologia (FCT) through grants UID/00324/2025 and UID/PRR/00324/2025.

References

Jiří Adámek, and Jiří Rosický. (2020). "How Nice Are Free Completions of Categories?." *Topology and its Applications* 273 : 106972. doi: `https://doi.org/10.1016/j.topol.2019.106972`.

Giorgio Bacci, Radu Mardare, Prakash Panangaden, and Gordon Plotkin. (2024). "Sum and Tensor of Quantitative Effects." *Logical Methods in Computer Science* Volume 20, Issue 4(9) . doi: `https://doi.org/10.46298/lmcs-20(4:9)2024`.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. (2018). "Automatic Differentiation in Machine Learning: A Survey." *Journal of Machine Learning Research* 18(153) : 1–43. http://jmlr.org/papers/v18/17-468.html.

Paulo Emílio de Vilhena, and François Pottier. (2023). "Verifying an Effect-Handler-Based Define-by-Run Reverse-Mode AD Library." *Logical Methods in Computer Science* 19(4) : 5:1–5:51. arXiv: 2112.07292. doi: https://doi.org/10.46298/LMCS-19(4:5)2023.

Eduardo J Dubuc. (1974). "Free Monoids." *Journal of Algebra* 29(2) : 208–228. doi: https://doi.org/10.1016/0021-8693(74)90095-7.

Martin Erwig, and Steve Kollmansberger. (2006). "Probabilistic Functional Programming in Haskell." *Journal of Functional Programming* 16(1) : 21–34. https://web.engr.oregonstate.edu/~erwig/papers/PFP_JFP06.pdf.

Martin Erwig, and Steve Kollmansberger. (2025). "Probabilistic Functional Programming." Software version 0.2.8. https://hackage.haskell.org/package/probability.

Martin Hyland, and John Power. (2007). "The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads." *Electronic Notes in Theoretical Computer Science* 172 : 437–458. doi: https://doi.org/10.1016/j.entcs.2007.02.019.

Martin Hyland, Paul Blain Levy, Gordon Plotkin, and John Power. (2007). "Combining Algebraic Effects with Continuations." *Theoretical Computer Science* 375(1) : 20–40. doi: https://doi.org/10.1016/j.tcs.2006.12.026.

Ohad Kammar, and Dylan McDermott. (2018). "Factorisation Systems for Logical Relations and Monadic Lifting in Type-and-Effect System Semantics." In *MFPS*. pp. 14.

Joaquim Lambek, and P. J. Scott. (1986). "Introduction to Higher Order Categorical Logic." *Cambridge Studies in Advanced Mathematics*. Cambridge University Press.

Alexander K. Lew, Mathieu Huot, Sam Staton, and Vikash K. Mansinghka. (2023). "ADEV: Sound Automatic Differentiation of Expected Values of Probabilistic Programs." *Proceedings of the ACM on Programming Languages* 7(POPL) : 5:121–5:153. doi: https://doi.org/10.1145/3571198.

Fernando Lucatelli Nunes, and Matthijs Vákár. (2023). "CHAD for Expressive Total Languages." *Mathematical Structures in Computer Science* 33(4–5) : 311–426. doi: https://doi.org/10.1017/S096012952300018X.

Fernando Lucatelli Nunes, Gordon Plotkin, and Matthijs Vákár. (2025). "Unraveling the Iterative CHAD." arXiv: 2505.15002. https://arxiv.org/abs/2505.15002.

Saunders Mac Lane. (1997). "Categories for the Working Mathematician." *Graduate Texts in Mathematics*. Springer.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. (2020). "Monte Carlo Gradient Estimation in Machine Learning." *Journal of Machine Learning Research* 21(132) : 1–62. https://jmlr.org/papers/v21/19-346.html.

Gordon Plotkin, and John Power. (2003). “Algebraic Operations and Generic Effects." *Applied Categorical Structures* 11(1) : 69–94. doi: https://doi.org/10.1023/A:1023064908962.

Gordon D. Plotkin, and Matija Pretnar. (2013). “Handling Algebraic Effects." *Logical Methods in Computer Science* 9(4) : 23:1–23:36. doi: https://doi.org/10.2168/LMCS-9(4:23)2013.

John Schulman, Nicolas Heess, Théophane Weber, and Pieter Abbeel. (2015). “Gradient Estimation Using Stochastic Computation Graphs." In *Advances in Neural Information Processing Systems*. pp. 3528–3536. arXiv: 1506.05254.

Laurent Schwartz. (1950). “Théorie des distributions." *Actualités scientifiques et industrielles*. Hermann.

Jesse Sigal. (2021). “Automatic Differentiation via Effects and Handlers: An Implementation in Frank." arXiv: 2101.08095. doi: https://doi.org/10.48550/arXiv.2101.08095.

Jesse Aaron Sigal. (2024). “Automatic Differentiation via Effects and Handlers." PhD thesis. University of Edinburgh. hdl: 1842/41919. doi: https://doi.org/10.7488/era/4642.

Tom J. Smeding, and Matthijs I. L. Vákár. (2024). “Efficient CHAD." *Proceedings of the ACM on Programming Languages* 8(POPL) : 1060–1088. doi: https://doi.org/10.1145/3632878.

Emile van Krieken, Jakub M. Tomczak, and Annette ten Teije. (2021). “Storchastic: A Framework for General Stochastic Automatic Differentiation." In *Advances in Neural Information Processing Systems 34*. Neural Information Processing Systems Foundation. pp. 7574–7587. arXiv: 2104.00428. https://proceedings.neurips.cc/paper/2021/hash/3dfe2f633108d604df160cd1b01710db-Abstract.html.

Matthijs Vákár. (2021). “Reverse AD at Higher Types: Pure, Principled and Denotationally Correct." In *Programming Languages and Systems: 30th European Symposium on Programming, ESOP 2021*. Springer. pp. 607–634. doi: https://doi.org/10.1007/978-3-030-72019-3_22.

Matthijs Vákár, and Tom Smeding. (2022). “CHAD: Combinatory Homomorphic Automatic Differentiation." *ACM Transactions on Programming Languages and Systems* 44(3) : 20:1–20:49. doi: https://doi.org/10.1145/3527634.

Matthijs Vákár, Ohad Kammar, and Sam Staton. (2019). “A Domain Theory for Statistical Probabilistic Programming." *Proceedings of the ACM on Programming Languages* 3(POPL) : 36:1–36:29. doi: https://doi.org/10.1145/3290349.

A Source language for CHAD

In this section, we present the complete type theory of the source language for CHAD, whose abbreviated introduction can be found in Section 3.1.

This is a first-order language whose term model can be understood as a free distributive category generated by the basic types real^k, $k \in \mathbb{N}$, and by a family of chosen operations op, and equipped with a strong monad, a categorical constructor, and a specific Eilenberg–Moore algebra.

A.1 Type and term grammar

The type grammar for the source language is the one displayed in Equation 59

$$\tau, \sigma, \rho ::= 0 \mid 1 \mid \tau \times \sigma \mid \tau \sqcup \sigma \mid \mathsf{real}^k \; (k \in \mathbb{N}) \mid \mathrm{M}\,\tau \tag{59}$$

with real being used as an abbreviation for real^1.

Similarly, the term construction grammar is the one in Equation 60, denoting variables by $x, y, z, \ldots$ and contexts as $\Gamma = x_1 : \tau_1, \cdots, x_n : \tau_n$

$$\begin{aligned} t, s, r ::= {} & x \mid \langle\, \rangle \mid \langle t, s \rangle \mid \mathrm{pr}_1\, t \mid \mathrm{pr}_2\, t \mid \mathrm{inl}\, t \mid \mathrm{inr}\, t \\ & \mid \mathsf{case}\; t \;\mathsf{of}\; \mathsf{inl}(x) \Rightarrow s \mid \mathsf{inr}(y) \Rightarrow r \;\mid \mathrm{abort}(t) \\ & \mid \mathsf{let}\; x = t \;\mathsf{in}\; s \;\mid \mathsf{op}(t_1, \ldots, t_k) \;\mid \mathrm{return}(t) \\ & \mid \mathrm{bind}(t; x.s) \mid \mathrm{categorical}(\langle\langle t_i, w_i \rangle\rangle_{i=1}^n) \;\mid \mathrm{E}_\nu(t) \end{aligned} \tag{60}$$

where $\nu = \mathsf{real}^p$, for some $p \in \mathbb{N}$, is the fixed type on which the handler algebra E_ν is defined, and op ranges over a family of chosen built-in operations, each with a signature $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$, representing the basic operations available in the source language. These operations should be thought of as *deterministic operations*, and are chosen according to the intended usage of the language. In the categorical constructor, the second components w_i are real-valued log-weights; the corresponding nonnegative weights are produced by the interpretation of categorical.

A.2 Typing rules

The typing rules for the language are as follows.

A.2.1 Nullary product and coproduct

The elimination rule for nullary sum can be seen as a type-theoretic instance of the explosion principle, whereas the introduction rule for the nullary product states the existence of an element of 1 in any context,

$$\frac{\Gamma \vdash t : 0}{\Gamma \vdash \mathrm{abort}(t) : \rho}\,0\mathrm{E} \qquad \frac{}{\Gamma \vdash \langle\, \rangle : 1}\,1\mathrm{I}$$

A.2.2 Binary products

The introduction and elimination rules of binary products behave similarly to their set-based semantics, cartesian products.

$$\frac{\Gamma \vdash u : \tau \times \sigma}{\Gamma \vdash \mathrm{pr}_1\, u : \tau} \times\mathrm{E1} \qquad \frac{\Gamma \vdash u : \tau \times \sigma}{\Gamma \vdash \mathrm{pr}_2\, u : \sigma} \times\mathrm{E2} \qquad \frac{\Gamma \vdash t : \tau \quad \Gamma \vdash s : \sigma}{\Gamma \vdash \langle t, s \rangle : \tau \times \sigma} \times\mathrm{I}$$

A.2.3 Binary sums

As with binary products, binary sums typing rules are based on their intended set-based semantics, the disjoint union. The elimination rule is

$$\frac{\Gamma \vdash t : \tau \sqcup \sigma \quad \Gamma, x : \tau \vdash u : \rho \quad \Gamma, y : \sigma \vdash v : \rho}{\Gamma \vdash \mathsf{case}\ t\ \mathsf{of}\ \mathsf{inl}(x) \Rightarrow u \mid \mathsf{inr}(y) \Rightarrow v : \rho} \sqcup\mathrm{E}$$

and the introduction rules are

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathrm{inl}\, t : \tau \sqcup \sigma} \sqcup\mathrm{I1} \qquad \frac{\Gamma \vdash s : \sigma}{\Gamma \vdash \mathrm{inr}\, s : \tau \sqcup \sigma} \sqcup\mathrm{I2}$$

A.2.4 Operations

Any built-in operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$ defines a typing rule

$$\frac{\Gamma \vdash t_j : \mathsf{real}^{n_j} \quad (j \in \{1, \ldots, k\})}{\Gamma \vdash \mathsf{op}(t_1, \ldots, t_k) : \mathsf{real}^m} \textsc{Operations}$$

A.2.5 Structural rules

We add a structural rule for the let-binding,

$$\frac{\Gamma \vdash t : \sigma \quad \Gamma, x : \sigma \vdash s : \rho}{\Gamma \vdash \mathsf{let}\ x = t\ \mathsf{in}\ s : \rho} \textsc{Let introduction}$$

and another one for assumptions, meaning that variables already in the context may be taken as terms,

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \textsc{Assumption}$$

A.2.6 Syntactic sugar and pattern-matching

The n-ary product and coproduct notation used below is syntactic sugar for the binary grammar. We take both products and coproducts to be right-associated:

$$\tau_1 \times \cdots \times \tau_n := \begin{cases} 1, & n = 0, \\ \tau_1, & n = 1, \\ \tau_1 \times (\tau_2 \times \cdots \times \tau_n), & n \geq 2, \end{cases}$$

and similarly

$$\tau_1 \sqcup \cdots \sqcup \tau_n := \begin{cases} 0, & n = 0, \\ \tau_1, & n = 1, \\ \tau_1 \sqcup (\tau_2 \sqcup \cdots \sqcup \tau_n), & n \geq 2. \end{cases}$$

The corresponding term notation is defined recursively by

$$\langle t_1, \ldots, t_n \rangle := \begin{cases} \langle\,\rangle\,, & n = 0, \\ t_1, & n = 1, \\ \langle t_1, \langle t_2, \ldots, t_n \rangle \rangle\,, & n \geq 2, \end{cases}$$

and by

$$\mathrm{in}_1^n\, t := \mathrm{inl}\, t \quad (n \geq 2), \qquad \mathrm{in}_i^n\, t := \mathrm{inr}(\mathrm{in}_{i-1}^{n-1}\, t) \quad (2 \leq i \leq n),$$

with $\mathrm{in}_1^1\, t := t$. We write $\mathrm{in}_i\, t$ when n is clear.

The n-ary coproduct case expression is defined by nested binary cases:

$$\mathrm{case}_i^1\,(t; x_i.s_i) := \mathsf{let}\ x_1 = t\ \mathsf{in}\ s_1,$$

and, for $n \geq 2$,

$$\mathrm{case}_i^n\,(t; x_i.s_i) := \mathsf{case}\ t\ \mathsf{of}\ \mathsf{inl}(x_1) \Rightarrow s_1 \mid \mathsf{inr}(y) \Rightarrow \mathrm{case}_j^{n-1}\,(y; x_{j+1}.s_{j+1})\,,$$

where y is fresh.

Similarly, n-ary product pattern matching is defined by nested projections:

$$\mathrm{caseprod}\,(t; x_1.s) := \mathsf{let}\ x_1 = t\ \mathsf{in}\ s,$$

and, for $n \geq 2$,

$$\mathrm{caseprod}\,(t; x_1, \ldots, x_n.s) := \mathsf{let}\ z = t\ \mathsf{in}\ \mathsf{let}\ x_1 = \mathrm{pr}_1\, z\ \mathsf{in}\ \ \mathrm{caseprod}\,(\mathrm{pr}_2\, z; x_2, \ldots, x_n.s)\,,$$

where z is fresh. With these definitions, the displayed n-ary product and coproduct typing rules are admissible.

A.2.7 Monadic types

For monadic types, we add typing rules for the return and bind term constructions. Together with their equational rules, this ensures that they behave as the unit and Kleisli extension of a strong monad.

$$\frac{\Gamma \vdash t : \mathrm{M}\,\sigma \quad \Gamma, x : \sigma \vdash s : \mathrm{M}\,\tau}{\Gamma \vdash \mathrm{bind}(t; x.s) : \mathrm{M}\,\tau}\,\mathrm{M\,E} \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathrm{return}(t) : \mathrm{M}\,\tau}\,\mathrm{M\,I}$$

A.2.8 Measure-specific rules

All the rules above would remain the same regardless of which monadic effect one wishes to implement in the source language. The ones that follow are specific to the FAD monad and describe the construction of terms using categorical with real-valued log-weights and the handler algebra E_ν,

$$\frac{\Gamma \vdash t_i : \tau \quad (i \in \{1, \ldots, n\}) \quad \Gamma \vdash w_i : \mathsf{real} \quad (i \in \{1, \ldots, n\})}{\Gamma \vdash \mathrm{categorical}(\langle \langle t_i, w_i \rangle \rangle_{i=1}^n) : \mathrm{M}\,\tau}\,\textsc{Categ.} \qquad \frac{\Gamma \vdash t : \mathrm{M}\,\nu}{\Gamma \vdash \mathrm{E}_\nu(t) : \nu}\,\textsc{Handler}$$

Here the terms w_i are log-weights rather than weights themselves; in the **Set**-interpretation below, categorical exponentiates them, so no positivity premise is needed at the source type level.

A.3 Equational theory

The pure fragment has the usual β- and η-laws for 1, $\times$, and $\sqcup$, which are omitted (see Section II.2 of Lambek and Scott (1986)). For the monadic and the handler fragments, we add the following rules. No additional definitional equations are imposed on the categorical constructors in the FAD source language treated here.

A.3.1 Kleisli laws

We consider the usual equational theory of strong monads. In what follows, $\equiv$ denotes definitional equality,

$$\begin{aligned} \mathrm{bind}(\mathrm{return}(t)\,;x\,.\,s) \equiv s[t/x] \qquad \mathrm{bind}(t;x\,.\,\mathrm{return}(x)) \equiv t \\ \mathrm{bind}(\mathrm{bind}(t;x\,.\,s)\,;y\,.\,r) \equiv \mathrm{bind}(t;x\,.\,\mathrm{bind}(s;y\,.\,r)) \end{aligned} \tag{61}$$

where we use $s[t/x]$ for the capture-avoiding substitution of the term t for the variable x in the expression s.

A.3.2 Eilenberg–Moore algebra laws

We also add laws ensuring that the handler algebra E_ν acts as an Eilenberg–Moore algebra.

$$\mathrm{E}_\nu(\mathrm{return}(v)) \equiv v \qquad \mathrm{E}_\nu(\mathrm{bind}(t;x\,.\,u)) \equiv \mathrm{E}_\nu(\mathrm{bind}(t;x\,.\,\mathrm{return}(\mathrm{E}_\nu(u)))) \tag{62}$$

A.4 Denotational semantics

We denote by Syn the term model of the source language, i.e., the *syntactic category of the source language*. Universally, it is the freely generated distributive category on the objects real^k, $k \in \mathbb{N}$, with operations op ranging in the family of built-in operations, equipped with a strong monad, a specific algebra on real^p for some $p \in \mathbb{N}$, and a categorical operation. This means that any category with this structure admits a structure-preserving functor from Syn, as is spelled out in Lemma A.2.

In particular, we may interpret Syn in **Set**, thus providing set-based semantics to the source language with a functor $[\![\cdot]\!]_{\mathbf{Set}} : \mathsf{Syn} \to \mathbf{Set}$. This semantics is defined recursively on the term and type structures, and is spelled out as follows.

A.4.1 Types

$$\begin{aligned} [\![0]\!]_{\mathbf{Set}} &= \emptyset & [\![\tau \times \sigma]\!]_{\mathbf{Set}} &= [\![\tau]\!]_{\mathbf{Set}} \times [\![\sigma]\!]_{\mathbf{Set}} \\ [\![1]\!]_{\mathbf{Set}} &= \{*\} & [\![\tau \sqcup \sigma]\!]_{\mathbf{Set}} &= [\![\tau]\!]_{\mathbf{Set}} \amalg [\![\sigma]\!]_{\mathbf{Set}} \\ [\![\mathsf{real}^n]\!]_{\mathbf{Set}} &= \mathbb{R}^n & [\![\mathrm{M}\,\tau]\!]_{\mathbf{Set}} &= M([\![\tau]\!]_{\mathbf{Set}}) \end{aligned}$$

A.4.2 Contexts and typing judgements

Using the interpretations of types, we may define interpretations of contexts and of typing judgements: a context $\Gamma = x_1 : \tau_1, \cdots, x_n : \tau_n$ is interpreted as $[\![\Gamma]\!]_{\mathbf{Set}} = \prod_{i=1}^n [\![\tau_i]\!]_{\mathbf{Set}}$, and a typing judgement $\Gamma \vdash t : \tau$ as a function $[\![\Gamma \vdash t : \tau]\!]_{\mathbf{Set}} : [\![\Gamma]\!]_{\mathbf{Set}} \to [\![\tau]\!]_{\mathbf{Set}}$, whose definition is provided inductively on the term structure.

A.4.3 Deterministic fragment

Variables, constants, and constructors of the deterministic fragment are interpreted as follows, for all $\gamma = \langle \gamma_1, \ldots, \gamma_n \rangle \in [\![\Gamma]\!]_{\mathbf{Set}}$,

$$
\begin{aligned}
[\![\Gamma \vdash x_j : \tau_j]\!]_{\mathbf{Set}}(\langle \gamma_1, \ldots, \gamma_n \rangle) &= \gamma_j && (j \in \{1, \ldots, n\}) \\
[\![\Gamma \vdash \langle\,\rangle : 1]\!]_{\mathbf{Set}}(\gamma) &= * \\
[\![\Gamma \vdash \mathrm{pr}_i\, u : \tau_i]\!]_{\mathbf{Set}}(\gamma) &= \pi_i([\![\Gamma \vdash u : \tau_1 \times \tau_2]\!]_{\mathbf{Set}}(\gamma)) && (i \in \{1, 2\}) \\
[\![\Gamma \vdash \langle t, s \rangle : \tau \times \sigma]\!]_{\mathbf{Set}}(\gamma) &= \langle [\![\Gamma \vdash t : \tau]\!]_{\mathbf{Set}}(\gamma)\, , [\![\Gamma \vdash s : \sigma]\!]_{\mathbf{Set}}(\gamma) \rangle \\
[\![\Gamma \vdash \mathrm{inl}\, u : \tau \sqcup \sigma]\!]_{\mathbf{Set}}(\gamma) &= \iota_1([\![\Gamma \vdash u : \tau]\!]_{\mathbf{Set}}(\gamma)) \\
[\![\Gamma \vdash \mathrm{inr}\, u : \tau \sqcup \sigma]\!]_{\mathbf{Set}}(\gamma) &= \iota_2([\![\Gamma \vdash u : \sigma]\!]_{\mathbf{Set}}(\gamma)) \\
[\![\Gamma \vdash \mathrm{abort}\, t : \rho]\!]_{\mathbf{Set}}(\gamma) &= \mathrm{abs}_{[\![\rho]\!]_{\mathbf{Set}}}([\![\Gamma \vdash t : 0]\!]_{\mathbf{Set}}(\gamma)) \\
[\![\Gamma \vdash \mathsf{let}\ x = t\ \mathsf{in}\ s : \rho]\!]_{\mathbf{Set}}(\gamma) &= [\![\Gamma, x : \sigma \vdash s : \rho]\!]_{\mathbf{Set}}(\gamma, [\![\Gamma \vdash t : \sigma]\!]_{\mathbf{Set}}(\gamma))
\end{aligned}
$$

where, for any set S, $\mathrm{abs}_S : \emptyset \to S$ is the unique function. Case statements are interpreted as branching functions,

$$
\begin{aligned}
&[\![\Gamma \vdash \mathsf{case}\ p\ \mathsf{of}\ \mathsf{inl}(x) \Rightarrow u \mid \mathsf{inr}(y) \Rightarrow v : \rho]\!]_{\mathbf{Set}}(\gamma) \\
&= \begin{cases} [\![\Gamma, x : \tau \vdash u : \rho]\!]_{\mathbf{Set}}(\gamma, a) & \text{if } [\![\Gamma \vdash p : \tau \sqcup \sigma]\!]_{\mathbf{Set}}(\gamma) = \iota_1(a) \\ [\![\Gamma, y : \sigma \vdash v : \rho]\!]_{\mathbf{Set}}(\gamma, b) & \text{if } [\![\Gamma \vdash p : \tau \sqcup \sigma]\!]_{\mathbf{Set}}(\gamma) = \iota_2(b) \end{cases}
\end{aligned}
$$

and the built-in operations are interpreted as their representing counterparts, that is, for any operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$,

$$
[\![\Gamma \vdash \mathsf{op}(t_1, \cdots, t_k) : \mathsf{real}^m]\!]_{\mathbf{Set}}(\gamma) = [\![\mathsf{op}]\!]_{\mathbf{Set}}([\![\Gamma \vdash t_1 : \mathsf{real}^{n_1}]\!]_{\mathbf{Set}}(\gamma)\, , \cdots , [\![\Gamma \vdash t_k : \mathsf{real}^{n_k}]\!]_{\mathbf{Set}}(\gamma))
$$

where $[\![\mathsf{op}]\!]_{\mathbf{Set}}$ is required to be a differentiable function, as we are interested in computing derivatives.

A.4.4 Monadic fragment

For the interpretation of monadic types and their associated terms, we assume that the category on which the semantics is to be interpreted has a strong monad $\langle M, \eta, \mu, \mathrm{st} \rangle$, with strength components $\mathrm{st}_{A,B} : A \times MB \to M(A \times B)$. In the case of **Set**, all monads are canonically strong. We interpret return and bind as

$$
\begin{aligned}
[\![\Gamma \vdash \mathrm{return}(t) : \mathrm{M}\,\tau]\!]_{\mathbf{Set}} &= (\eta)_{[\![\tau]\!]_{\mathbf{Set}}} \circ [\![\Gamma \vdash t : \tau]\!]_{\mathbf{Set}} \\
[\![\Gamma \vdash \mathrm{bind}(t; x\ .\ s) : \mathrm{M}\,\tau]\!]_{\mathbf{Set}} &= (\mu)_{[\![\tau]\!]_{\mathbf{Set}}} \circ M([\![\Gamma, x : \sigma \vdash s : \mathrm{M}\,\tau]\!]_{\mathbf{Set}}) \circ (\mathrm{st})_{[\![\Gamma]\!]_{\mathbf{Set}}, [\![\sigma]\!]_{\mathbf{Set}}} \circ \Big\langle \mathrm{id}_{[\![\Gamma]\!]_{\mathbf{Set}}}, [\![\Gamma \vdash t : \mathrm{M}\,\sigma]\!]_{\mathbf{S}}
\end{aligned}
$$

Notice that these formulas are also applicable for interpretations on other distributive categories with strong monads.

A.4.5 Measure-specific terms

Finally, we present the interpretation of terms constructed with categorical or with the handler algebra E_ν. For categorical, the second components are interpreted as log-weights:

$$
\begin{aligned}
&[\![\Gamma \vdash \mathrm{categorical}(\langle\langle t_i, w_i \rangle\rangle_{i=1}^n) : \mathrm{M}\,\tau]\!]_{\mathbf{Set}}(\gamma) \\
&\quad = (\mathrm{categorical})_{[\![\tau]\!]_{\mathbf{Set}}}(\langle\langle [\![\Gamma \vdash t_i : \tau]\!]_{\mathbf{Set}}(\gamma)\, , [\![\Gamma \vdash w_i : \mathsf{real}]\!]_{\mathbf{Set}}(\gamma) \rangle\rangle_{i=1}^n)
\end{aligned}
$$

and

$$\begin{aligned}\llbracket \Gamma \vdash \mathrm{E}_\nu(t) : \nu \rrbracket_{\mathbf{Set}}(\gamma) &= \mathrm{alg}^{\mathbf{Set}}_\nu(\llbracket \Gamma \vdash t : \mathrm{M}\,\nu \rrbracket_{\mathbf{Set}}(\gamma)) \\ &= \sum_{v \in \mathrm{supp}(\llbracket \Gamma \vdash t : \mathrm{M}\,\nu \rrbracket_{\mathbf{Set}}(\gamma))} \llbracket \Gamma \vdash t : \mathrm{M}\,\nu \rrbracket_{\mathbf{Set}}(\gamma)(v) \cdot v\end{aligned}$$

A.5 Soundness and initiality

As claimed, Syn can be seen as a free distributive category with a chosen strong monad on the objects real^k, $k \in \mathbb{N}$, and the built-in family of operations op, as well as categorical. As a freely generated category, it has a universal property, described in Lemma A.2 (restatement of Lemma 3.2), which describes the existence of unique interpretation functors. These interpretations are also *sound*, meaning that any equation from the equational theory holds semantically, as stated in Lemma A.1 (restatement of Lemma 3.1).

Lemma A.1 (Soundness of source language interpretations)**.** *For any interpretation* $\llbracket\cdot\rrbracket : \mathsf{Syn} \to \mathsf{X}$*, if* $\Gamma \vdash t \equiv s : \tau$ *is derivable in the equational theory above, then* $\llbracket \Gamma \vdash t : \tau \rrbracket = \llbracket \Gamma \vdash s : \tau \rrbracket$ *as morphisms* $\llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$ *in* X*.*

The proof follows by routine verification on the ways to obtain a definitional equality $t \equiv s$ in the source language. The soundness of the (omitted) pure fragment laws follows from the distributive category laws; there are no additional categorical-equations to check.

On the other hand, the soundness of term equalities with return and bind follows from the interpretation of bind with the strength st of the monad M, together with the Kleisli equations and coherence laws of strong monads, while the definitional equalities for the handler algebra follow from the Eilenberg–Moore algebra axioms.

The universal property of the syntactic category can be given explicitly as follows.

Lemma A.2 (Universal property of Syn)**.** *Let* X *be any distributive category equipped with*

- *A strong monad* $\left\langle M^{\mathsf{X}}, \eta^{\mathsf{X}}, \mu^{\mathsf{X}}, \mathrm{st}^{\mathsf{X}} \right\rangle$*, with* $\mathrm{st}^{\mathsf{X}}_{A,B} : A \times M^{\mathsf{X}}(B) \to M^{\mathsf{X}}(A \times B)$*;*
- *For every* $n \in \mathbb{N}$*, chosen objects* X_n*;*
- *An Eilenberg–Moore algebra* $\mathrm{alg} : M^{\mathsf{X}}(X_p) \to X_p$ *over the object* X_p*;*
- *For every built-in operation* $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$*, a chosen morphism* $\llbracket \mathsf{op} \rrbracket : X_{n_1} \times \cdots \times X_{n_k} \to X_m$*;*
- *For each* $n \in \mathbb{N}$ *and each object* A *interpreting a type* τ*, an arrow* $\mathrm{categorical}^{\mathsf{X}}_{\tau,n} : (A \times X_1)^n \to M^{\mathsf{X}}(A)$*, used to interpret* categorical *terms built from lists of length* n *of atoms of type* τ *and log-weights in* X_1*.*

Then there exists a unique strict functor $\llbracket\cdot\rrbracket : \mathsf{Syn} \to \mathsf{X}$ *preserving finite products and coproducts, the strong monad structure, the handler algebra, and* categorical *as above, and sending every equation of* Syn *to an identity in* X*.*

The proof follows by standard algebraic presentation arguments, using the formulas for interpretation above, with $\llbracket \mathsf{real}^n \rrbracket = X_n$ for all $n \in \mathbb{N}$, and induction over the structure of the morphisms in Syn. Uniqueness follows from the fact that the equations in Syn are precisely those required to enforce its freely generated structure.

B Target language for CHAD

In this section we present the complete type-theoretic description of the target language for CHAD.

As explained in the abridged version in Section 3.3, the core idea of CHAD is to encode a code transformation as a categorical transformation between syntactic categories. As such code written in the source language (see Section A) is systematically converted by induction on its structural components, giving their differentiated versions. The target language is the language in which these transformed codes are written.

The target language is a dependently typed functional programming language, and it is split into a cartesian type theory and, for every context of cartesian types (i.e. a *cartesian context*), an associated linear type theory.

Denoting by CSyn the term model of the cartesian fragment, each cartesian context Γ (viewed as an object of CSyn, up to variable renaming) determines a syntactic category $\mathsf{LSyn}[\Gamma]$ for the associated linear type theory. Any substitution $\Gamma \to \Gamma'$ induces a reindexing functor $\mathsf{LSyn}[\Gamma'] \to \mathsf{LSyn}[\Gamma]$ between these linear term models. Thus LSyn is a strict indexed category $\mathsf{CSyn}^{\mathrm{op}} \to \mathbf{Cat}$.

The syntactic category of the target language is then given by the Grothendieck construction Tgt (see Section 2.3 of Lucatelli Nunes and Vákár (2023)).

As in the source language, we write τ, σ, ρ for the *cartesian types*, and reserve underlined Greek letters $\underline{\tau}, \underline{\sigma}, \underline{\rho}$ for linear types. We then have *cartesian typing judgements* of the form $\Gamma \vdash t : \tau$ and *linear typing judgements* of the form $\Gamma; \mathsf{v} : \underline{\sigma} \vdash s : \underline{\rho}$, with exactly one distinguished linear variable, whose name may be α-renamed.

We can divide the description of the target language into three layers:

- **Non-monadic core**: deterministic operations. The description of this part is independent of any monad-related components.
- **Oplax strong-monad core**: liftings possible for any strong monad in the source category.
- **Distribution and handler**: language features that are specific to the FAD monad.

By breaking the target language into these layers, we get a generic procedure for introducing (strong) monadic effects into the automatic differentiation framework of CHAD.

B.1 Type construction grammar

Cartesian types are described by the following grammar:

$$\tau, \sigma, \rho ::= (\text{As in Equation (26)}) \mid \underline{\sigma} \multimap \underline{\rho} \mid \Sigma\,(x : \sigma)\,.\underline{\tau}' \tag{63}$$

meaning that, in the cartesian fragment, we also have function types between linear types and dependent sums. For linear types, we use the grammar

$$\underline{\tau}, \underline{\sigma}, \underline{\rho} ::= \underline{\mathsf{real}^k} \;\; (k \in \mathbb{N}) \mid \underline{1} \mid \underline{\tau} \times \underline{\sigma} \mid \mathrm{case}\;\; t\;\; \mathrm{of}\;\; \{\mathrm{in}_i\, x_i \mapsto \underline{\sigma}_i\}_{i=1}^{n} \mid \mathrm{M}^{w}_{p:\mathrm{M}(\tau)}\,[x \mapsto \underline{\sigma}] \tag{64}$$

meaning that the base Euclidean-space types have associated linear types (in the sense that the cotangent space of a Euclidean space at any point can be identified with the base space itself), as well as the fibre-wise zero object 1. Moreover, we allow fibre-wise binary biproducts $\times$ and families of linear types indexed by finite disjunctions, and we introduce the cotangent linear type associated with monadic types.

The goal of highlighting the different layers is to make explicit which constructions are general for the implementation of algebraic monadic effects and which ones are specific to the FAD monad.

B.2 Term construction grammar

As with types, we also distinguish between *cartesian* and *linear terms*. The cartesian terms are described by the grammar

$$t, s, r ::= (\text{As in Equation (60)}) \mid \langle a, t\rangle \mid \mathrm{case}\ p\ \mathrm{of}\ \langle x, y\rangle \mapsto s \mid \lambda v\ .\ s \tag{65}$$

where the constructor $\langle a, t\rangle$ and the eliminator $\mathrm{case}\ p\ \mathrm{of}\ \langle x, y\rangle \mapsto s$ are used in the typing rules for the cartesian Σ-type, and $\lambda v\ .\ s$ is used for abstraction over the unique linear variable v.

Since all term constructions from the source language (Equation (60)) are present in the cartesian fragment of the target language, with the same typing rules and equational theory, the source syntactic category Syn can be seen as a full subcategory of CSyn.

For linear terms, we use the following grammar:

$$\begin{aligned} t, s, r ::= {} & v \mid \mathsf{let}\ v = t\ \mathsf{in}\ s \mid \langle t, s\rangle \mid \mathrm{pr}_i\, t \mid 0 \mid t + s \mid r\, t \\ & \mid \mathsf{op}(t_1, \cdots, t_k)(v) \mid \mathrm{return}^w_t(s) \mid \mathrm{bind}^w_{s;x\ .\ t}(r; u) \\ & \mid \mathrm{categorical}^w_{\langle\langle t_i, w_i\rangle\rangle_{i=1}^n}(\langle\langle t'_i, w'_i\rangle\rangle_{i=1}^n) \mid \mathrm{E}^w_{\nu,t}(s) \end{aligned} \tag{66}$$

The three lines correspond to the three layers: the first line consists of the non-monadic core operations (linear variables, let bindings, pairings and projections, the zero vector, sums, and $r\, t$ for linear application with r a cartesian term of type $\underline{\sigma} \multimap \underline{\rho}$). The second line has the derivatives of the built-in operations op of the source language, as well as the linear fibre components of the monad operations return and bind. Finally, the third line has the linear fibre components of categorical and of the handler algebra E_ν.

B.3 Typing rules

B.3.1 Non-monadic core

The cartesian typing rules are the same as the ones of the source language (see Section A.2) for the common term constructors. For the dependent sums, we have the rules

$$\frac{\Gamma \vdash a : \sigma \quad \Gamma \vdash t : \tau[a/x]}{\Gamma \vdash \langle a, t\rangle : \Sigma\,(x : \sigma)\,.\tau} \qquad \frac{\Gamma \vdash \rho\ \mathrm{type} \quad \Gamma \vdash p : \Sigma\,(x : \sigma)\,.\tau \quad \Gamma, x : \sigma, y : \tau \vdash s : \rho}{\Gamma \vdash \mathrm{case}\ p\ \mathrm{of}\ \langle x, y\rangle \to s : \rho}$$

and for the linear function types, we introduce the typing rules

$$\frac{\Gamma; v : \underline{\sigma} \vdash s : \underline{\rho}}{\Gamma \vdash \lambda v\ .\ s : \underline{\sigma} \multimap \underline{\rho}} \qquad \frac{\Gamma \vdash r : \underline{\sigma} \multimap \underline{\rho} \quad \Gamma; v : \underline{\tau} \vdash t : \underline{\sigma}}{\Gamma; v : \underline{\tau} \vdash r\, t : \underline{\rho}}$$

There are also non-monadic-layer typing rules for linear terms.

Linear variable

$$\frac{}{\Gamma; v : \underline{\sigma} \vdash v : \underline{\sigma}} \qquad \textit{(one distinguished linear variable)}$$

Linear let

$$\frac{\Gamma; v : \underline{\tau} \vdash t : \underline{\sigma} \quad \Gamma; v : \underline{\sigma} \vdash s : \underline{\rho}}{\Gamma; v : \underline{\tau} \vdash \mathsf{let}\ v = t\ \mathsf{in}\ s : \underline{\rho}}$$

Linear biproduct

$$\frac{}{\Gamma; v : \underline{\rho} \vdash 0 : \underline{\sigma}} \qquad \frac{\Gamma; v : \underline{\rho} \vdash s : \underline{\sigma} \quad \Gamma; v : \underline{\rho} \vdash t : \underline{\sigma}}{\Gamma; v : \underline{\rho} \vdash s + t : \underline{\sigma}}$$

$$\frac{\Gamma; v : \underline{\rho} \vdash s : \underline{\tau} \quad \Gamma; v : \underline{\rho} \vdash t : \underline{\sigma}}{\Gamma; v : \underline{\rho} \vdash \langle s, t \rangle : \underline{\tau} \times \underline{\sigma}} \qquad \frac{\Gamma; v : \underline{\rho} \vdash u : \underline{\tau}_1 \times \underline{\tau}_2}{\Gamma; v : \underline{\rho} \vdash \mathrm{pr}_i\, u : \underline{\tau}_i}$$

where $i \in \{1, 2\}$.

Linear case over cartesian sums

$$\frac{\Gamma \vdash t : \tau_1 \sqcup \cdots \sqcup \tau_n \quad \left\{\Gamma, x_i : \tau_i; v : \underline{\sigma}_i \vdash r_i : \underline{\rho}_i\right\}_{i=1}^{n}}{\Gamma; v : \mathrm{case}_i^n\, (t; x_i.\underline{\sigma}_i) \vdash \mathrm{case}_i^n\, (t; x_i.r_i) : \mathrm{case}_i^n \left(t; x_i.\underline{\rho}_i\right)}$$

Built-in operations For any built-in operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$ in the source language, we associate a term constructor Dop, whose intended interpretation is as the transposed Jacobian of op. It has the following associated typing rule.

$$\frac{\{\Gamma \vdash t_j : \mathsf{real}^{n_j}\}_{j=1}^{k}}{\Gamma; v : \underline{\mathsf{real}^m} \vdash \mathrm{Dop}(t_1, \cdots, t_k)(v) : \underline{\mathsf{real}^{n_1}} \times \cdots \times \underline{\mathsf{real}^{n_k}}}$$

B.3.2 Oplax strong-monad core

The following typing rules concern a generic strong monad, together with its chosen cotangent lift. In every rule below, all displayed cartesian contexts are assumed to be well formed. A judgement such as $\Gamma, x : \tau \vdash \underline{\sigma}$ type means that $\underline{\sigma}$ is a linear type in the fibre $\mathsf{LSyn}[\Gamma, x : \tau]$; a judgement $\Gamma \vdash \underline{\rho}$ type means that $\underline{\rho}$ is a linear type in $\mathsf{LSyn}[\Gamma]$. The linear variables occurring in the linear arguments of return^w, bind^w, $\mathrm{categorical}^w$, and E_ν^w are local binding occurrences; following the single-linear-variable convention, we write each of them as v, with scope determined by the surrounding judgement and with the usual capture-avoiding substitution convention.

Type formation

$$\frac{\Gamma \vdash p : \mathrm{M}(\tau) \quad \Gamma, x : \tau \vdash \underline{\sigma}\ \text{type}}{\Gamma \vdash \mathrm{M}^w_{p:\mathrm{M}(\tau)}\, [x \mapsto \underline{\sigma}]\ \text{type}}$$

Here x is the returned value on which the continuation cotangent type may depend.

w-return

$$\frac{\Gamma \vdash t : \tau \quad \Gamma, x : \tau \vdash \underline{\sigma} \text{ type} \quad \Gamma \vdash \underline{\rho} \text{ type} \quad \Gamma; v : \underline{\sigma}[t/x] \vdash s : \underline{\rho}}{\Gamma; v : \mathrm{M}^{w}_{\mathrm{return}(t):\mathrm{M}(\tau)}\,[x \mapsto \underline{\sigma}] \vdash \mathrm{return}^{w}_{t}(s) : \underline{\rho}}$$

Thus the argument s is typed in the returned-value fibre $\underline{\sigma}[t/x]$, while the result consumes a cotangent of return(t) whose continuation fibre is $x \mapsto \underline{\sigma}$.

w-bind

$$\frac{\begin{array}{c}\Gamma \vdash s : \mathrm{M}(\sigma) \qquad \Gamma, x : \sigma \vdash t : \mathrm{M}(\tau) \\ \Gamma, x : \sigma \vdash \underline{\sigma} \text{ type} \qquad \Gamma, y : \tau \vdash \underline{\tau} \text{ type} \qquad \Gamma \vdash \underline{\rho} \text{ type} \\ \Gamma; v : \mathrm{M}^{w}_{s:\mathrm{M}(\sigma)}\,[x \mapsto \underline{\sigma}] \vdash r : \underline{\rho} \quad \Gamma, x : \sigma; v : \mathrm{M}^{w}_{t:\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash u : \underline{\rho} \times \underline{\sigma}\end{array}}{\Gamma; v : \mathrm{M}^{w}_{\mathrm{bind}(s;x\,.\,t):\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash \mathrm{bind}^{w}_{s;x\,.\,t}(r;u) : \underline{\rho}}$$

Here $\underline{\sigma}$ is the continuation-cotangent type for the first computation, depending on its returned value $x : \sigma$, and $\underline{\tau}$ is the continuation-cotangent type for the composite computation, depending on its final returned value $y : \tau$. The second linear argument u returns both a contribution to the external cotangent $\underline{\rho}$ and a cotangent in the fibre $\underline{\sigma}$ of the bound value x.

B.3.3 Measure-specific layer

These typing rules relate specifically to the FAD monad, providing rules for the transposed derivative of categorical and for the handler algebra E_{ν}.

w-categorical For all $n \in \mathbb{N}$, the finite atomic distribution constructor has the following cotangent rule.

$$\frac{\begin{array}{c}\Gamma, x : \tau \vdash \underline{\sigma} \text{ type} \qquad \Gamma \vdash \underline{\rho} \text{ type} \\ \{\Gamma \vdash t_i : \tau\}_{i=1}^{n} \quad \{\Gamma \vdash w_i : \mathsf{real}\}_{i=1}^{n} \quad \left\{\Gamma; v : \underline{\sigma}[t_i/x] \vdash t'_i : \underline{\rho}\right\}_{i=1}^{n} \quad \left\{\Gamma; v : \underline{\mathsf{real}} \vdash w'_i : \underline{\rho}\right\}_{i=1}^{n}\end{array}}{\Gamma; v : \mathrm{M}^{w}_{\mathrm{categorical}(\langle\langle t_i, w_i\rangle\rangle_{i=1}^{n}):\mathrm{M}(\tau)}\,[x \mapsto \underline{\sigma}] \vdash \mathrm{categorical}^{w}_{\langle\langle t_i, w_i\rangle\rangle_{i=1}^{n}}(\langle\langle t'_i, w'_i\rangle\rangle_{i=1}^{n}) : \underline{\rho}}$$

Thus $\underline{\sigma}$ is the cotangent family for the sampled value, while $\underline{\rho}$ is the ambient cotangent type to which each atom and weight backpropagator contributes.

Transposed derivative of the handler Since $\nu = \mathsf{real}^{p}$ for some $p \in \mathbb{N}$, write $\underline{\tau}_{\nu} := \underline{\mathsf{real}^{p}}$ for its associated linear type. The constant family $y : \nu \mapsto \underline{\tau}_{\nu}$ is a well-formed linear type in $\mathsf{LSyn}[\Gamma, y : \nu]$. We impose the following typing rule.

$$\frac{\Gamma \vdash t : \mathrm{M}(\nu) \quad \Gamma \vdash \underline{\rho} \text{ type} \quad \Gamma; v : \mathrm{M}^{w}_{t:\mathrm{M}(\nu)}\,[y \mapsto \underline{\tau}_{\nu}] \vdash s : \underline{\rho}}{\Gamma; v : \underline{\tau}_{\nu} \vdash \mathrm{E}^{w}_{\nu,t}(s) : \underline{\rho}}$$

B.4 Equational theory

In addition to the equations from the equational theory of the source language (Section A.3), which also apply for the cartesian fragment, the following equations describe the equational theory of the target language. They are stated up to the usual capture-avoiding substitution on the bound variables.

Cartesian layer (Σ, products, and sums)

$$
\begin{aligned}
&(\Sigma\beta) && \mathrm{caseprod}\,(\langle a, t\rangle\,; x, y.s) &&\equiv s\,[x := a, y := t]\\
&(\Sigma\eta) && \mathrm{caseprod}\,(p; x, y.\,\langle x, y\rangle) &&\equiv p
\end{aligned}
$$

Linear biproduct (fibrewise) and additivity

$$
\begin{aligned}
&(\oplus\beta) && \mathrm{pr}_1\,\langle s, t\rangle &&\equiv s\\
& && \mathrm{pr}_2\,\langle s, t\rangle &&\equiv t\\
&(\oplus\eta) && \langle \mathrm{pr}_1\,t, \mathrm{pr}_2\,t\rangle &&\equiv t\\
&(\text{Add-unit}) && s + 0 &&\equiv s\\
& && 0 + s &&\equiv s\\
&(\text{Add-assoc/comm}) && (r + s) + t &&\equiv r + (s + t)\\
& && s + t &&\equiv t + s\\
&(\text{Add-lin}) && r[0\,/v] &&\equiv 0\\
& && r[s + t/v] &&\equiv r[s/v] + r[t/v]
\end{aligned}
$$

Linear structural

$$
(\text{Lin. struct.}) \qquad \mathsf{let}\ v = t\ \mathsf{in}\ s \equiv s[t/v]
$$

Linear function space

$$
\begin{aligned}
&(\multimap \beta) && \lambda v\ .\ s(t) &&\equiv s[t/v]\\
&(\multimap \eta) && \lambda v\ .\ r(v) &&\equiv r \quad \text{when} \quad (v \notin \mathrm{FV}(r))
\end{aligned}
$$

Built-in operations For every built-in operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^{m}$, its associated transposed Jacobian term constructor Dop is additive in its linear argument and respects cartesian substitution.

Oplax fibred strong-monad and algebra equations We impose the following equations whenever all displayed terms are well-typed. They are stated modulo the Kleisli equations and Eilenberg–Moore algebra equations in the cartesian fragment, so that the linear types on the two sides are identified by the corresponding cartesian equality. All occurrences of the distinguished linear variable in arguments of

$$
\mathrm{return}^{w}_{t}(r)\,, \qquad \mathrm{bind}^{w}_{s;x.t}(r; u)\,, \qquad \mathrm{E}^{w}_{\nu,t}(r)
$$

are binding occurrences and are implicitly α-renamed as necessary to avoid capture. The equations we impose are precisely those of an oplax indexed strong monad and an oplax indexed Eilenberg–Moore algebra. They are also the equations that are required to ensure that the interpretation of the source language in the target language is well-defined, i.e. the transposed derivatives of the Kleisli and Eilenberg–Moore algebra equations for the source language.

The w-left-unit law is the transpose of

$$
\mathrm{bind}(\mathrm{return}(a)\,; x.t) \equiv t[a/x].
$$

Suppose

$$
\Gamma \vdash a : \sigma, \qquad \Gamma, x : \sigma \vdash t : \mathrm{M}(\tau)\,,
$$

$$\Gamma, x : \sigma \vdash \underline{\sigma} \text{ type}, \qquad \Gamma, y : \tau \vdash \underline{\tau} \text{ type}, \qquad \Gamma \vdash \underline{\rho} \text{ type},$$

and

$$\Gamma; v : \underline{\sigma}[a/x] \vdash r : \underline{\rho}, \qquad \Gamma, x : \sigma; v : \mathrm{M}^w_{t:\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash u : \underline{\rho} \times \underline{\sigma}.$$

Then

$$\begin{aligned} &\mathrm{bind}^w_{\mathrm{return}(a);x.t}(\mathrm{return}^w_a(r)\,; u) \\ &\qquad \equiv \mathsf{let}\ v = u[a/x]\ \mathsf{in}\ \ \mathrm{pr}_1\, v + r[\mathrm{pr}_2\, v/v]. \end{aligned} \tag{67}$$

The w-right-unit law is the transpose of

$$\mathrm{bind}(s; x.\,\mathrm{return}(x)) \equiv s.$$

Suppose

$$\Gamma \vdash s : \mathrm{M}(\sigma)\,, \qquad \Gamma, x : \sigma \vdash \underline{\sigma} \text{ type}, \qquad \Gamma \vdash \underline{\rho} \text{ type},$$

and

$$\Gamma; v : \mathrm{M}^w_{s:\mathrm{M}(\sigma)}\,[x \mapsto \underline{\sigma}] \vdash r : \underline{\rho}.$$

Then

$$\mathrm{bind}^w_{s;x.\,\mathrm{return}(x)}(r; \langle 0, \mathrm{return}^w_x(v)\rangle) \equiv r. \tag{68}$$

In the second argument, the term

$$\mathrm{return}^w_x(v)$$

is typed using the returned-value fibre

$$y : \sigma \mapsto \underline{\sigma}[y/x].$$

Thus it produces a term of type $\underline{\sigma}$ in the context $\Gamma, x : \sigma$.

The w-associativity law is the transpose of

$$\mathrm{bind}(\mathrm{bind}(s; x.t)\,; y.q) \equiv \mathrm{bind}(s; x.\,\mathrm{bind}(t; y.q))\,.$$

Suppose

$$\Gamma \vdash s : \mathrm{M}(\sigma)\,, \qquad \Gamma, x : \sigma \vdash t : \mathrm{M}(\tau)\,, \qquad \Gamma, y : \tau \vdash q : \mathrm{M}(\upsilon)\,,$$

$$\Gamma, x : \sigma \vdash \underline{\sigma} \text{ type}, \qquad \Gamma, y : \tau \vdash \underline{\tau} \text{ type}, \qquad \Gamma, z : \upsilon \vdash \underline{\upsilon} \text{ type}, \qquad \Gamma \vdash \underline{\rho} \text{ type},$$

and

$$\begin{gathered} \Gamma; v : \mathrm{M}^w_{s:\mathrm{M}(\sigma)}\,[x \mapsto \underline{\sigma}] \vdash r : \underline{\rho}, \\ \Gamma, x : \sigma; v : \mathrm{M}^w_{t:\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash u : \underline{\rho} \times \underline{\sigma}, \\ \Gamma, y : \tau; v : \mathrm{M}^w_{q:\mathrm{M}(\upsilon)}\,[z \mapsto \underline{\upsilon}] \vdash e : \underline{\rho} \times \underline{\tau}. \end{gathered}$$

Then

$$\begin{aligned} &\mathrm{bind}^w_{\mathrm{bind}(s;x.t);y.q}\Big(\mathrm{bind}^w_{s;x.t}(r; u)\,; e\Big) \\ &\qquad \equiv \mathrm{bind}^w_{s;x.\,\mathrm{bind}(t;y.q)}\Big(r; \mathrm{bind}^w_{t;y.q}(u; \mathsf{let}\ v = e\ \mathsf{in}\ \ \langle\langle \mathrm{pr}_1\, v, 0\rangle\,, \mathrm{pr}_2\, v\rangle)\Big)\,. \end{aligned} \tag{69}$$

On the right-hand side, q and e are weakened from $\Gamma, y : \tau$ to $\Gamma, x : \sigma, y : \tau$, and the inserted zero has type $\underline{\sigma}$.

Side note: the oplax comparison term for cartesian reindexing is definable. It is defined by the pure-continuation special case of bind^w. If

$$\Gamma \vdash p : \mathrm{M}(\sigma)\,, \qquad \Gamma, x : \sigma \vdash h : \tau, \qquad \Gamma, y : \tau \vdash \underline{\tau} \text{ type},$$

then

$$\begin{aligned}&\Gamma; v : \mathrm{M}^{w}_{\mathrm{bind}(p;x.\,\mathrm{return}(h)):\mathrm{M}(\tau)}\,[y \mapsto \underline{\tau}] \vdash \alpha_{p,x.h,\underline{\tau}}(v) : \mathrm{M}^{w}_{p:\mathrm{M}(\sigma)}\,[x \mapsto \underline{\tau}[h/y]]\,,\\ &\alpha_{p,x.h,\underline{\tau}}(v) := \mathrm{bind}^{w}_{p;x.\,\mathrm{return}(h)}(v; \langle 0, \mathrm{return}^{w}_{h}(v)\rangle)\,.\end{aligned} \tag{70}$$

The identity and composition equations for these comparison maps follow from (68) and (69), respectively. Thus the displayed equations present an oplax fibred monad over the source monad. Since they are schematic in an arbitrary cartesian context, the usual contextual strength is inherited from the given strength of the source monad.

The handler w-unit law is the transpose of

$$\mathrm{E}_{\nu}(\mathrm{return}(a)) \equiv a.$$

Suppose

$$\Gamma \vdash a : \nu, \qquad \Gamma \vdash \underline{\rho}\ \text{type}, \qquad \Gamma; v : \underline{\tau}_{\nu} \vdash r : \underline{\rho}.$$

Then

$$\mathrm{E}^{w}_{\nu,\mathrm{return}(a)}(\mathrm{return}^{w}_{a}(r)) \equiv r. \tag{71}$$

The handler w-multiplication law is the transpose of

$$\mathrm{E}_{\nu}(\mathrm{bind}(t; x.u)) \equiv \mathrm{E}_{\nu}(\mathrm{bind}(t; x.\,\mathrm{return}(\mathrm{E}_{\nu}(u))))\,.$$

Suppose

$$\Gamma \vdash t : \mathrm{M}(\sigma)\,, \qquad \Gamma, x : \sigma \vdash u : \mathrm{M}(\nu)\,,$$
$$\Gamma, x : \sigma \vdash \underline{\sigma}\ \text{type}, \qquad \Gamma \vdash \underline{\rho}\ \text{type},$$

and

$$\Gamma; v : \mathrm{M}^{w}_{t:\mathrm{M}(\sigma)}\,[x \mapsto \underline{\sigma}] \vdash r : \underline{\rho},$$
$$\Gamma, x : \sigma; v : \mathrm{M}^{w}_{u:\mathrm{M}(\nu)}\,[y \mapsto \underline{\tau}_{\nu}] \vdash s : \underline{\rho} \times \underline{\sigma}.$$

Then

$$\begin{aligned}&\mathrm{E}^{w}_{\nu,\mathrm{bind}(t;x.u)}\Big(\mathrm{bind}^{w}_{t;x.u}(r; s)\Big)\\ &\quad\equiv \mathrm{E}^{w}_{\nu,\mathrm{bind}(t;x.\,\mathrm{return}(\mathrm{E}_{\nu}(u)))}\Big(\mathrm{bind}^{w}_{t;x.\,\mathrm{return}(\mathrm{E}_{\nu}(u))}\Big(r; \mathrm{return}^{w}_{\mathrm{E}_{\nu}(u)}\Big(\mathrm{E}^{w}_{\nu,u}(s)\Big)\Big)\Big)\,.\end{aligned} \tag{72}$$

B.5 Syntactic category of the target language

With the proper type theory spelled out, we may now provide the proper definition of $\mathsf{LSyn}[\tau]$ for each cartesian type τ:

- *Objects*: linear types $\underline{\sigma}$ well-formed in the context $p : \tau$.
- *Morphisms*: a morphism $\underline{\rho} \to \underline{\sigma}$ is a linear term $p : \tau; v : \underline{\rho} \vdash s : \underline{\sigma}$, considered modulo α-equivalence. *Composition* is performed by linear let bindings, and *identities* are given by the linear variable v itself (see the *Linear variable* typing rule).

Given a term $p : \tau \vdash t : \sigma$ in CSyn, write $q : \sigma$ for the distinguished variable in the fibre over σ. Its reindexing $t^{*} : \mathsf{LSyn}[\sigma] \to \mathsf{LSyn}[\tau]$ acts by substitution on objects and terms:

$$\begin{aligned}&\text{(Objects)} & t^{*}(\underline{\sigma}) &:= \underline{\sigma}[t/q]\\ &\text{(Morphisms)} & t^{*}\Big(\Big(q : \sigma; v : \underline{\rho} \vdash s : \underline{\sigma}\Big)\Big) &:= p : \tau; v : \underline{\rho}[t/q] \vdash s[t/q] : \underline{\sigma}[t/q]\end{aligned}$$

Theorem B.1 (Split fibration structure)**.** *The projection* $\pi : \mathsf{Tgt} \to \mathsf{CSyn}$ *is a split fibration with chosen cartesian lifts given by reindexing* t^{*} *on the nose. Identities and composition for* LSyn *are strictly preserved by reindexing.*

C $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ denotational semantics

In Section 3.4, we discuss briefly the functor $[\![\cdot]\!]_{\mathsf{F}} : \mathsf{Tgt} \to \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, which interprets the target language in the category $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. The goal of this Appendix is to describe this functor in further detail.

Thus cartesian types are interpreted as sets (the base part of the objects in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$), whereas linear judgements over cartesian contexts are interpreted in $\mathbf{Vect}^{\mathrm{op}}$. For convenience, given a morphism $f : U \to V$ in $\mathbf{Vect}^{\mathrm{op}}$, we denote by $f^{\mathrm{op}} : V \to U$ the corresponding linear map in $\mathbf{Vect}$. This lets us display fibre maps in the ordinary $\mathbf{Vect}$ orientation.

C.1 Type semantics

The semantics of the CSyn fragment establishes the base cartesian semantics, interpreting cartesian contexts as the base sets of objects of $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. The term constructors shared with the source language have the same interpretation as in the source-language semantics in $\mathbf{Set}$ (Sections 3.2 and A.4),

$$\begin{aligned} [\![0]\!]_{\mathbf{Set}} &= \emptyset & \qquad [\![\tau \times \sigma]\!]_{\mathbf{Set}} &= [\![\tau]\!]_{\mathbf{Set}} \times [\![\sigma]\!]_{\mathbf{Set}} \\ [\![1]\!]_{\mathbf{Set}} &= \{*\} & \qquad [\![\tau \sqcup \sigma]\!]_{\mathbf{Set}} &= [\![\tau]\!]_{\mathbf{Set}} \amalg [\![\sigma]\!]_{\mathbf{Set}} \\ [\![\mathsf{real}^n]\!]_{\mathbf{Set}} &= \mathbb{R}^n & \qquad [\![\mathrm{M}\,\tau]\!]_{\mathbf{Set}} &= M([\![\tau]\!]_{\mathbf{Set}}) \end{aligned}$$

with deterministic primitives interpreted as their chosen total functions (which we assume to be differentiable), and $\langle \mathrm{M}, \mathrm{return}, \mathrm{bind}\rangle$ as $\langle M, \eta, \mu\rangle$. For a cartesian judgement under the context $p : \tau$, dependent sums are interpreted as follows: for every $\gamma \in [\![\tau]\!]_{\mathbf{Set}}$,

$$[\![\Sigma x : \sigma \,.\, \tau']\!]_{\mathbf{Set}}(\gamma) := \coprod_{a \in [\![\sigma]\!]_{\mathbf{Set}}(\gamma)} [\![\tau']\!]_{\mathbf{Set}}(\gamma, a) \tag{73}$$

and the linear function types are interpreted as the set of linear morphisms between the linear interpretations of the respective linear types,

$$[\![\underline{\sigma} \multimap \underline{\rho}]\!]_{\mathbf{Set}}(\gamma) := \mathbf{Vect}\Big([\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)\,, [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)\Big) \tag{74}$$

where the linear interpretation $[\![\cdot]\!]^{\mathrm{lin}}_{p:\tau}$ of linear types dependent on a cartesian variable $p : \tau$ provides a real vector space interpretation for those linear types, making them into (non-topological) vector bundles over the base set $[\![\tau]\!]_{\mathbf{Set}}$. As such, for every linear type $\underline{\sigma}$ well-formed in the context $p : \tau$, we get a functor (i.e. a family)

$$[\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau} : [\![\tau]\!]_{\mathbf{Set}} \to \mathbf{Vect}^{\mathrm{op}}$$

The linear interpretation is also described inductively over the structure of linear types, for every $\gamma \in [\![\tau]\!]_{\mathbf{Set}}$,

$$\begin{aligned} [\![\underline{\mathsf{real}}^k]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= \mathbb{R}^k \\ [\![\underline{1}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= 0 \\ [\![\underline{\sigma} \times \underline{\rho}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= [\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \oplus [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \\ [\![\mathrm{case}^n_i\,(t; x_i.\underline{\sigma}_i)]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= \Big\{ [\![\underline{\sigma}_i]\!]^{\mathrm{lin}}_{p:\tau, x_i:\tau_i}(\gamma, a) \quad \text{if } [\![t]\!]_{\mathbf{Set}}(\gamma) = \mathrm{in}_i\, a \\ [\![\mathrm{M}^w_{q:\mathrm{M}(\rho)}\,[x \mapsto \underline{\sigma}]]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= \bigoplus_{x \in \mathrm{supp}[\![q]\!]_{\mathbf{Set}}(\gamma)} \Big([\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau, x:\rho}(\gamma, x) \oplus \mathbb{R}\Big) \end{aligned} \tag{75}$$

Compare the definition of the monadic cotangent type semantics to the definition of the fibre part of the monad on objects in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ given in Equation 14.

With these definitions, we get a functor $[\![\cdot]\!]_{\mathsf{F}} : \mathsf{Tgt} \to \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ defined on objects, which, as described in Section B.5, are given by a pair $\langle p : \tau, \underline{\sigma} \rangle$ where $p : \tau$ is a cartesian context and $\underline{\sigma}$ a well-formed linear type under this context, as

$$[\![\langle p : \tau, \underline{\sigma} \rangle]\!]_{\mathsf{F}} := \left\langle [\![\tau]\!]_{\mathbf{Set}}, [\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau} \right\rangle \tag{76}$$

and on any morphism $\langle t, s \rangle : \left\langle p : \tau, \underline{\rho} \right\rangle \to \left\langle q : \sigma, \underline{\chi} \right\rangle$, which consists of a cartesian arrow $t : p \to q$ and a linear arrow in the fibre $s : \underline{\chi}[t/q] \to \underline{\rho}$ in $\mathsf{LSyn}[p : \tau]$, as

$$[\![\langle t, s \rangle]\!]_{\mathsf{F}} := \left\langle [\![t]\!]_{\mathbf{Set}}, \left(\gamma \mapsto \left([\![s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)^{\mathrm{op}} \in \mathbf{Vect}\left([\![\underline{\chi}]\!]^{\mathrm{lin}}_{q:\sigma}([\![t]\!]_{\mathbf{Set}}(\gamma)), [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \right) \right) \right) \right\rangle \tag{77}$$

C.2 Term semantics

Next, we provide interpretations for the terms themselves, which will be mapped to morphisms in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. Again, this is done inductively over the structure of the terms, and the cartesian part of the language is interpreted in **Set** as the base part of the morphisms.

Cartesian fragment For the term constructors in common with the source language, the interpretation is the same as the one described in Section A.4. For the other cartesian term constructors, in a cartesian context $\Gamma = p : \tau$, we define, for all $\gamma \in [\![\tau]\!]_{\mathbf{Set}}$,

$$\begin{aligned} [\![\Gamma \vdash \langle a, t \rangle : \Sigma\,(x : \sigma)\,.\tau']\!]_{\mathbf{Set}}(\gamma) &= \langle [\![a]\!]_{\mathbf{Set}}(\gamma), [\![t]\!]_{\mathbf{Set}}(\gamma) \rangle \\ [\![\Gamma \vdash \mathsf{case}\ p\ \mathsf{of}\ \langle x, y \rangle \to s : \rho]\!]_{\mathbf{Set}}(\gamma) &= [\![s]\!]_{\mathbf{Set}}(\gamma, a, b) \quad \text{where} \quad \langle a, b \rangle = [\![p]\!]_{\mathbf{Set}}(\gamma) \\ [\![\Gamma \vdash \lambda v\,.\,s : \underline{\sigma} \multimap \underline{\rho}]\!]_{\mathbf{Set}}(\gamma) &= [\![s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)^{\mathrm{op}} \end{aligned} \tag{78}$$

Linear application is a linear term constructor; its semantics is given below.

Linear core Linear terms are interpreted fibrewise in $\mathbf{Vect}^{\mathrm{op}}$, or, equivalently, as linear maps in **Vect**, but with domain and codomain reversed. The first display below is written in the $\mathbf{Vect}^{\mathrm{op}}$-orientation; when we need the ordinary **Vect**-orientation, we apply $-^{\mathrm{op}}$. In what follows, we consider the linear terms under a cartesian context $\Gamma = p : \tau$,

$$\begin{aligned} [\![\Gamma; v : \underline{\sigma} \vdash v : \underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= \mathrm{id}_{[\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)} \\ [\![0]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= 0 \\ [\![s + t]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= [\![s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) + [\![t]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \\ [\![\langle s, t \rangle]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= \left[[\![s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma), [\![t]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \right] \\ [\![\mathrm{pr}_i\, u]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= [\![u]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \circ \iota_i && (i \in \{1, 2\}) \\ [\![\mathsf{let}\ v = t\ \mathsf{in}\ s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= [\![t]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) \circ [\![s]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) && (\text{Composition in } \mathbf{Vect}^{\mathrm{op}}) \\ [\![\mathrm{case}^n_i\ (t; x_i.r_i)]\!]^{\mathrm{lin}}_{p:\tau}(\gamma) &= [\![r_i]\!]^{\mathrm{lin}}_{p:\tau, x_i:\tau_i}(\gamma, a) && \text{if } [\![t]\!]_{\mathbf{Set}}(\gamma) = \mathrm{in}_i(a) \end{aligned}$$

For linear application, where $\Gamma \vdash r : \underline{\sigma} \multimap \underline{\rho}$ and $\Gamma; v : \underline{\alpha} \vdash t : \underline{\sigma}$, we use the ordinary **Vect**-orientation

$$[\![r\, t]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)^{\mathrm{op}} = [\![r]\!]_{\mathbf{Set}}(\gamma) \circ [\![t]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)^{\mathrm{op}}\,.$$

Finally, for any built-in operation op : $\mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$,

$$[\![\mathrm{Dop}(t_1, \cdots, t_k)(v)]\!]^{\mathrm{lin}}_{p:\tau}(\gamma)^{\mathrm{op}} = \boldsymbol{\nabla}[\![\mathrm{op}]\!]_{\mathbf{Set}}([\![t_1]\!]_{\mathbf{Set}}(\gamma), \cdots, [\![t_k]\!]_{\mathbf{Set}}(\gamma)) : \mathbb{R}^m \to \mathbb{R}^{n_1} \oplus \cdots \oplus \mathbb{R}^{n_k}$$

where $[\![\mathrm{op}]\!]_{\mathbf{Set}}$ is the intended set-based interpretation of the operation op, assumed to be differentiable, so that taking its transpose derivative $\boldsymbol{\nabla}[\![\mathrm{op}]\!]_{\mathbf{Set}}$ is well-defined.

Transposed derivative of return Let the cartesian context be $\Gamma = p : \chi$. Suppose $p : \chi \vdash t : \tau$, $p : \chi, x : \tau \vdash \underline{\sigma}$ type, and $p : \chi; v : \underline{\sigma}[t/x] \vdash s : \underline{\rho}$. For $\gamma \in [\![\chi]\!]_{\mathbf{Set}}$, put

$$a := [\![t]\!]_{\mathbf{Set}}(\gamma), \qquad A_a := [\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\chi,x:\tau}(\gamma, a), \qquad R_\gamma := [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\chi}(\gamma).$$

Then the ordinary **Vect**-orientation of the fibre map is

$$[\![\mathrm{return}^w_t(s)]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} = [\![s]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} \circ \pi_{A_a} : A_a \oplus \mathbb{R} \to R_\gamma. \tag{79}$$

Here $\pi_{A_a} : A_a \oplus \mathbb{R} \to A_a$ is the projection onto the returned-value cotangent. The log-weight component is discarded, as expected for the derivative of a unit distribution with constant log-weight.

Transposed derivative of bind Let the cartesian context again be $\Gamma = p : \chi$, and consider a well-typed term $\mathrm{bind}^w_{s;x\,.\,t}(r; u)$ with $p : \chi \vdash s : \mathrm{M}(\sigma)$ and $p : \chi, x : \sigma \vdash t : \mathrm{M}(\tau)$. Fix $\gamma \in [\![\chi]\!]_{\mathbf{Set}}$, and define

$$\begin{aligned} w &:= [\![s]\!]_{\mathbf{Set}}(\gamma) && \in M([\![\sigma]\!]_{\mathbf{Set}}), \\ \mu_x &:= [\![t]\!]_{\mathbf{Set}}(\gamma, x) && \in M([\![\tau]\!]_{\mathbf{Set}}), \\ f_\gamma(x) &:= \mu_x && : [\![\sigma]\!]_{\mathbf{Set}} \to M([\![\tau]\!]_{\mathbf{Set}}), \\ \widehat{w} &:= M(f_\gamma)(w) && \in M(M([\![\tau]\!]_{\mathbf{Set}})), \\ \bar{w} &:= (\mu)_{[\![\tau]\!]_{\mathbf{Set}}}(\widehat{w}) && \in M([\![\tau]\!]_{\mathbf{Set}}). \end{aligned}$$

Thus $\bar{w}$ is the set-level denotation of $\mathrm{bind}(s; x.t)$ at γ. Put

$$A_x := [\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\chi,x:\sigma}(\gamma, x), \qquad B_y := [\![\underline{\tau}]\!]^{\mathrm{lin}}_{p:\chi,y:\tau}(\gamma, y), \qquad R_\gamma := [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\chi}(\gamma).$$

The two linear arguments give ordinary-orientation maps

$$\begin{aligned} r^\sharp_\gamma &:= [\![r]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} : \bigoplus_{x \in \operatorname{supp} w} (A_x \oplus \mathbb{R}) \to R_\gamma, \\ u^\sharp_{\gamma,x} &:= [\![u]\!]^{\mathrm{lin}}_{p:\chi,x:\sigma}(\gamma, x)^{\mathrm{op}} : \bigoplus_{y \in \operatorname{supp} \mu_x} (B_y \oplus \mathbb{R}) \to R_\gamma \oplus A_x. \end{aligned}$$

The family $\{u^\sharp_{\gamma,x}\}_{x \in [\![\sigma]\!]_{\mathbf{Set}}}$ is the fibre part of a morphism in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$

$$\widehat{f_\gamma} : \langle [\![\sigma]\!]_{\mathbf{Set}}, x \mapsto R_\gamma \oplus A_x \rangle \to M^{\mathsf{F}}(\langle [\![\tau]\!]_{\mathbf{Set}}, y \mapsto B_y \rangle),$$

whose base map is f_γ. Applying the lifted FAD monad to this morphism and evaluating the fibre map at w gives

$$\Theta^u_w := \left(\left(M^{\mathsf{F}}\left(\widehat{f_\gamma}\right)\right)_{\mathrm{fbr}}\right)_w (:) \bigoplus_{\xi \in \operatorname{supp} \widehat{w}} \left(\left(\bigoplus_{y \in \operatorname{supp} \xi} (B_y \oplus \mathbb{R})\right) \oplus \mathbb{R}\right) \to \bigoplus_{x \in \operatorname{supp} w} ((R_\gamma \oplus A_x) \oplus \mathbb{R}).$$

Similarly, let

$$\varpi^B_{\widehat{w}} := \left(\left(\left(\mu^{\mathsf{F}}\right)_{\langle [\![\tau]\!]_{\mathbf{Set}}, y \mapsto B_y \rangle}\right)_{\mathrm{fbr}}\right)_{\widehat{w}}(:) \bigoplus_{y \in \operatorname{supp} \bar{w}} (B_y \oplus \mathbb{R}) \to \bigoplus_{\xi \in \operatorname{supp} \widehat{w}} \left(\left(\bigoplus_{y \in \operatorname{supp} \xi} (B_y \oplus \mathbb{R})\right) \oplus \mathbb{R}\right)$$

be the ordinary-orientation fibre map of the lifted monad multiplication at $\widehat{w}$. For $C_x := R_\gamma \oplus A_x$, define

$$\Omega_\gamma : \bigoplus_{x \in \operatorname{supp} w} (C_x \oplus \mathbb{R}) \to R_\gamma,$$
$$\Omega_\gamma := \left(\nabla_{R_\gamma} \circ \bigoplus_{x \in \operatorname{supp} w} (\pi_{R_\gamma} \circ \pi_1)\right) + \left(r^\sharp_\gamma \circ \bigoplus_{x \in \operatorname{supp} w} \langle \pi_{A_x} \circ \pi_1, \pi_2 \rangle\right).$$

Here $\pi_1 : C_x \oplus \mathbb{R} \to C_x$, $\pi_2 : C_x \oplus \mathbb{R} \to \mathbb{R}$, and π_{R_γ}, π_{A_x} are the projections from $C_x = R_\gamma \oplus A_x$. The first summand of Ω_γ adds the direct ambient cotangents produced by the continuation backpropagators; the second feeds the cotangent of the bound value and the outer log-weight cotangent through r.

The semantics of bind^w is then

$$[\![\mathrm{bind}^w_{s;x\,.\,t}(r;u)]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} = \Omega_\gamma \circ \Theta^u_w \circ \varpi^B_{\widehat{w}} : \bigoplus_{y \in \operatorname{supp} \bar{w}} (B_y \oplus \mathbb{R}) \to R_\gamma. \tag{80}$$

Because Θ^u_w and $\varpi^B_{\widehat{w}}$ are the actual fibre maps of the lifted FAD functor and multiplication, this definition also covers collisions of atoms and collisions among the intermediate FADs μ_x.

Transposed derivative of categorical Again work in a cartesian context $p : \chi$. Consider a well-typed term

$$\mathrm{categorical}^w_{\langle\langle t_i, w_i \rangle\rangle_{i=1}^n}(\langle\langle t'_i, w'_i \rangle\rangle_{i=1}^n)$$

generated by the rule in the target language. Fix $\gamma \in [\![\chi]\!]_{\mathbf{Set}}$, and put

$$a_i := [\![t_i]\!]_{\mathbf{Set}}(\gamma), \qquad \ell_i := [\![w_i]\!]_{\mathbf{Set}}(\gamma), \qquad q := [\![\mathrm{categorical}(\langle\langle t_i, w_i \rangle\rangle_{i=1}^n)]\!]_{\mathbf{Set}}(\gamma).$$

Let

$$A_x := [\![\underline{\sigma}]\!]^{\mathrm{lin}}_{p:\chi, x:\tau}(\gamma, x), \qquad R_\gamma := [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\chi}(\gamma).$$

The lifted categorical operation $\mathrm{categorical}^{\mathsf{F}}$ has an ordinary-orientation fibre map at the list $\langle\langle a_i, \ell_i \rangle\rangle_{i=1}^n$, which we write as

$$\kappa_\gamma : \bigoplus_{x \in \operatorname{supp} q} (A_x \oplus \mathbb{R}) \to \bigoplus_{i=1}^n (A_{a_i} \oplus \mathbb{R}).$$

This is the fibre part of the left factor in the FAD factorisation; in particular, it performs the correct splitting when several displayed atoms denote the same value. Define

$$t^\sharp_{i,\gamma} := [\![t'_i]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} : A_{a_i} \to R_\gamma,$$
$$w^\sharp_{i,\gamma} := [\![w'_i]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} : \mathbb{R} \to R_\gamma.$$

Then

$$[\![\mathrm{categorical}^w_{\langle\langle t_i, w_i \rangle\rangle_{i=1}^n}(\langle\langle t'_i, w'_i \rangle\rangle_{i=1}^n)]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} = \nabla_{R_\gamma} \circ \left(\bigoplus_{i=1}^n \left(t^\sharp_{i,\gamma} \circ \pi_{A_{a_i}} + w^\sharp_{i,\gamma} \circ \pi_{\mathbb{R}}\right)\right) \circ \kappa_\gamma. \tag{81}$$

Transposed derivative of the handler algebra Let $\nu = \mathsf{real}^d$, and write $V_\nu := \mathbb{R}^d$ for the fibre of $\underline{\tau}_\nu$. Suppose $p : \chi \vdash t : \mathrm{M}(\nu)$, and fix $\gamma \in [\![\chi]\!]_{\mathbf{Set}}$. Put $m := [\![t]\!]_{\mathbf{Set}}(\gamma)$. The lifted handler algebra $\mathrm{E}^{\mathsf{F}}_\nu : M^{\mathsf{F}}\left(T^*\mathbb{R}^d\right) \to T^*\mathbb{R}^d$ has ordinary-orientation fibre map

$$\epsilon_m : V_\nu \to \bigoplus_{y \in \operatorname{supp} m} (V_\nu \oplus \mathbb{R}) ,$$

which, for the expectation handler, is explicitly

$$\epsilon_m(c) = \sum_{y \in \operatorname{supp} m} (m(y)\ c \cdot \mathbf{e}_y + m(y)\ \langle c, y \rangle \cdot \varepsilon_y) , \qquad c \in V_\nu .$$

Here $\mathbf{e}_y$ denotes the atom-cotangent summand and ε_y denotes the log-weight summand. Therefore, for $p : \chi; v : \mathrm{M}^w_t\,[y \mapsto \underline{\tau}_\nu] \vdash s : \underline{\rho}$,

$$[\![\mathrm{E}^w_{\nu,t}(s)]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} = [\![s]\!]^{\mathrm{lin}}_{p:\chi}(\gamma)^{\mathrm{op}} \circ \epsilon_m : V_\nu \to [\![\underline{\rho}]\!]^{\mathrm{lin}}_{p:\chi}(\gamma) . \tag{82}$$

D Code transformation rules

This section gives the full reverse-mode CHAD translation used in Equation (47). For every source judgement $\Gamma \vdash t : \tau$, the translation is a target-language term

$$\mathsf{CHAD}(\Gamma)_1 \vdash \mathsf{CHAD}[\Gamma](t) : \Sigma\,(p : \mathsf{CHAD}(\tau)_1)\,.\,(\mathsf{CHAD}(\tau)_2\,[p/p_\tau] \multimap \mathsf{CHAD}(\Gamma)_2) . \tag{83}$$

The first component is the primal computation and the second component is the backpropagator. To avoid capture in dependent cotangent types, we write $p_\tau : \mathsf{CHAD}(\tau)_1$ for the distinguished primal variable of the type τ. Thus $\mathsf{CHAD}(\tau)_2$ is read in the cartesian context $p_\tau : \mathsf{CHAD}(\tau)_1$, and every substitution into a cotangent type below is understood after this α-renaming.

Convention for the distinguished linear variable. The target language has one distinguished linear variable. In clauses involving return^w, bind^w, $\mathrm{categorical}^w$, or E^w, the variable bound by the outer expression $\lambda v\,.\,(-)$ has the conclusion type of the corresponding w-typing rule. The occurrences of v inside the arguments of the w-constructor are checked at the premise types of that rule. This is exactly the typing convention used in the rules of Section B.

D.1 Type and context translation

Base types, products, and sums. The primal translation $\mathsf{CHAD}(-)_1$ is cartesian, while $\mathsf{CHAD}(-)_2$ gives a linear cotangent type indexed by the corresponding distinguished primal variable:

$$\begin{aligned}
\mathsf{CHAD}(1)_1 &:= 1 \\
\mathsf{CHAD}(0)_1 &:= 0 \\
\mathsf{CHAD}\left(\mathsf{real}^k\right)_1 &:= \mathsf{real}^k \\
\mathsf{CHAD}(\tau \times \sigma)_1 &:= \mathsf{CHAD}(\tau)_1 \times \mathsf{CHAD}(\sigma)_1 \\
\mathsf{CHAD}(\tau \sqcup \sigma)_1 &:= \mathsf{CHAD}(\tau)_1 \sqcup \mathsf{CHAD}(\sigma)_1
\end{aligned} \tag{84}$$

and

$$\begin{array}{rcl} \mathsf{CHAD}(1)_2 & := & \underline{1} \\ \mathsf{CHAD}(0)_2 & := & \underline{1} \\ \mathsf{CHAD}\left(\mathsf{real}^k\right)_2 & := & \underline{\mathsf{real}^k} \\ \mathsf{CHAD}(\tau \times \sigma)_2 & := & \mathsf{CHAD}(\tau)_2\,[\mathrm{pr}_1\, p_{\tau\times\sigma}/p_\tau] \times \mathsf{CHAD}(\sigma)_2\,[\mathrm{pr}_2\, p_{\tau\times\sigma}/p_\sigma]. \\ \mathsf{CHAD}(\tau \sqcup \sigma)_2 & := & \text{case } p_{\tau\sqcup\sigma} \text{ of} \left\{ \begin{array}{l} \mathrm{inl}\ x \mapsto \mathsf{CHAD}(\tau)_2\,[x/p_\tau] \\ \mathrm{inr}\ y \mapsto \mathsf{CHAD}(\sigma)_2\,[y/p_\sigma] \end{array} \right\}. \end{array} \tag{85}$$

For the linear case-type former we use the usual dependent β-rules. For example,

$$\mathsf{CHAD}(\tau \sqcup \sigma)_2\,[\mathrm{inl}\, a/p_{\tau\sqcup\sigma}] \equiv \mathsf{CHAD}(\tau)_2\,[a/p_\tau], \quad \mathsf{CHAD}(\tau \sqcup \sigma)_2\,[\mathrm{inr}\, b/p_{\tau\sqcup\sigma}] \equiv \mathsf{CHAD}(\sigma)_2\,[b/p_\sigma].$$

The same convention applies to the n-ary coproduct notation introduced in Section A.

The monadic type. For the FAD monad, the primal remains a monadic computation and the cotangent is the lifted monadic cotangent type of Section B:

$$\begin{array}{l} \mathsf{CHAD}(\mathrm{M}(\tau))_1 := \mathrm{M}(\mathsf{CHAD}(\tau)_1)\,, \\ \mathsf{CHAD}(\mathrm{M}(\tau))_2 := \mathrm{M}^w_{p_{\mathrm{M}\,\tau}:\mathrm{M}\left(\mathsf{CHAD}(\tau)_1\right)}\,[x \mapsto \mathsf{CHAD}(\tau)_2\,[x/p_\tau]]\,. \end{array} \tag{86}$$

Here $p_{\mathrm{M}\,\tau}$ is the primal variable for the source type $\mathrm{M}(\tau)$, while p_τ is the primal variable for the returned value type τ. This distinction is necessary because the continuation cotangent family is indexed by returned values, not by the whole monadic computation.

Contexts. Contexts are translated componentwise on primals and by fibrewise biproducts on cotangents:

$$\begin{array}{ll} \mathsf{CHAD}(\cdot)_1 := \cdot, & \mathsf{CHAD}(\Gamma, x : \tau)_1 := \mathsf{CHAD}(\Gamma)_1\,,\ x : \mathsf{CHAD}(\tau)_1\,, \\ \mathsf{CHAD}(\cdot)_2 := \underline{1}, & \mathsf{CHAD}(\Gamma, x : \tau)_2 := \mathsf{CHAD}(\Gamma)_2 \times \mathsf{CHAD}(\tau)_2\,[x/p_\tau]. \end{array} \tag{87}$$

The first factor $\mathsf{CHAD}(\Gamma)_2$ is implicitly reindexed along the cartesian weakening $\mathsf{CHAD}(\Gamma)_1\,, x : \mathsf{CHAD}(\tau)_1 \to \mathsf{CHAD}(\Gamma)_1$. Thus pr_1 selects the cotangent tuple for the previous context and pr_2 selects the cotangent for the newest variable.

For $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, define the derived injection of a single variable cotangent into the context cotangent by recursion. Writing $\mathrm{idx}(x_j; \Gamma) = j$, if $\Gamma = \Gamma', x_n : \tau_n$, then

$$\begin{array}{ll} \mathrm{coproj}^\Gamma_n(v) := \left\langle 0_{\mathsf{CHAD}(\Gamma')_2}, v \right\rangle, & \\ \mathrm{coproj}^\Gamma_j(v) := \left\langle \mathrm{coproj}^{\Gamma'}_j(v), 0_{\mathsf{CHAD}(\tau_n)_2[x_n/p_{\tau_n}]} \right\rangle & \quad (1 \le j < n). \end{array} \tag{88}$$

We write $\mathrm{coproj}_{\mathrm{idx}(x;\Gamma)}\, v$ when Γ is clear from context.

D.2 Term translation

All zero terms below are the typed zeroes of the relevant fibrewise biproduct. The term translation is by structural recursion on the source typing derivation.

Nullary terms and variables.

$$\mathsf{CHAD}[\Gamma](\langle\,\rangle) := \Big\langle \langle\,\rangle\,, \lambda v\ .\ 0_{\mathsf{CHAD}(\Gamma)_2} \Big\rangle\,, \tag{89}$$

$$\mathsf{CHAD}[\Gamma](\mathrm{abort}(t)) := \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b\rangle \to \mathrm{abort}(p)\,, \tag{90}$$

$$\mathsf{CHAD}[\Gamma](x) := \Big\langle x, \lambda v\ .\ \mathrm{coproj}_{\mathrm{idx}(x;\Gamma)}\, v \Big\rangle\,. \tag{91}$$

Products.

$$\begin{aligned}\mathsf{CHAD}[\Gamma](\langle t, s\rangle) &:= \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b\rangle \to \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](s)\ \ \mathrm{of}\ \ \langle q, c\rangle \to \\ &\qquad \langle \langle p, q\rangle\,, \lambda v\ .\ b(\mathrm{pr}_1\, v) + c(\mathrm{pr}_2\, v)\rangle\,, \qquad (92)\\ \mathsf{CHAD}[\Gamma](\mathrm{pr}_1\, t) &:= \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b\rangle \to \\ &\qquad \Big\langle \mathrm{pr}_1\, p, \lambda v\ .\ b\Big(\Big\langle v, 0_{\mathsf{CHAD}(\sigma)_2[\mathrm{pr}_2\, p/p_\sigma]}\Big\rangle\Big)\Big\rangle \quad (t : \tau \times \sigma), \qquad (93)\\ \mathsf{CHAD}[\Gamma](\mathrm{pr}_2\, t) &:= \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b\rangle \to \\ &\qquad \Big\langle \mathrm{pr}_2\, p, \lambda v\ .\ b\Big(\Big\langle 0_{\mathsf{CHAD}(\tau)_2[\mathrm{pr}_1\, p/p_\tau]}, v\Big\rangle\Big)\Big\rangle \quad (t : \tau \times \sigma). \qquad (94)\end{aligned}$$

Coproducts. For n-ary sums, the following introduction rule specializes to inl and inr in the binary grammar:

$$\mathsf{CHAD}[\Gamma](\mathrm{in}_i\, t) := \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle x, x'\rangle \to \langle \mathrm{in}_i\, x, x'\rangle\,. \tag{95}$$

For case analysis, suppose $\Gamma \vdash t : \tau_1 \sqcup \cdots \sqcup \tau_n$ and $\Gamma, x_i : \tau_i \vdash s_i : \rho$. Then

$$\begin{aligned}&\mathsf{CHAD}[\Gamma](\mathrm{case}\ \ t\ \ \mathrm{of}\ \ \{\mathrm{in}_i\, x_i \to s_i\}_{i=1}^n) := \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle y, y'\rangle \to \\ &\mathrm{case}\ \ y\ \ \mathrm{of} \left\{ \begin{array}{c} \mathrm{in}_i\, x_i \mapsto \mathrm{case}\ \ \mathsf{CHAD}[\Gamma, x_i : \tau_i](s_i)\ \ \mathrm{of}\ \ \langle z_i, z_i'\rangle \to \\ \langle z_i, \lambda v\ .\ \mathsf{let}\ v = z_i'(v)\ \ \mathsf{in}\ \ \mathrm{pr}_1\, v + y'(\mathrm{pr}_2\, v)\rangle \end{array} \right\}_{i=1}^n .\end{aligned} \tag{96}$$

The application of y' in the branch uses the case-type β-rule above, so $\mathrm{pr}_2\, v$ has precisely the cotangent type of the active summand.

Let bindings and deterministic operations. If $\Gamma \vdash t : \sigma$ and $\Gamma, x : \sigma \vdash s : \rho$, then

$$\begin{aligned}\mathsf{CHAD}[\Gamma](\mathsf{let}\ x = t\ \mathsf{in}\ s) &:= \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b\rangle \to \mathrm{case}\ \ \mathsf{CHAD}[\Gamma, x : \sigma](s)\,[p/x]\ \ \mathrm{of}\ \ \langle q, c\rangle \to \\ &\qquad \langle q, \lambda v\ .\ \mathsf{let}\ v = c(v)\ \ \mathsf{in}\ \ \mathrm{pr}_1\, v + b(\mathrm{pr}_2\, v)\rangle\,. \end{aligned} \tag{97}$$

For a built-in operation $\mathsf{op} : \mathsf{real}^{n_1} \times \cdots \times \mathsf{real}^{n_k} \to \mathsf{real}^m$, write op for the chosen transposed Jacobian term constructor. Its CHAD clause is

$$\begin{aligned}&\mathsf{CHAD}[\Gamma](\mathsf{op}(t_1, \ldots, t_k)) := \\ &\quad \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t_1)\ \ \mathrm{of}\ \ \langle x_1, x_1'\rangle \to \cdots \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t_k)\ \ \mathrm{of}\ \ \langle x_k, x_k'\rangle \to \\ &\qquad \Bigg\langle \mathsf{op}(x_1, \ldots, x_k)\,, \lambda v\ .\ \mathsf{let}\ v = \mathsf{op}(x_1, \ldots, x_k)(v)\ \ \mathsf{in}\ \sum_{j=1}^{k} x_j'\Big(\mathrm{pr}_j\, v\Big) \Bigg\rangle .\end{aligned} \tag{98}$$

Here pr_j denotes the derived projection from the cotangent product returned by op.

Monad unit and multiplication. The unit clause is the one displayed in Equation (49):

$$\mathsf{CHAD}[\Gamma](\mathrm{return}(t)) := \mathrm{case}\ \mathsf{CHAD}[\Gamma](t)\ \mathrm{of}\ \langle p, b\rangle \to \left\langle \mathrm{return}(p)\,, \lambda v\ .\ \mathrm{return}^w_p(b(v))\right\rangle . \tag{99}$$

For bind, let $\Gamma \vdash t : \mathrm{M}(\sigma)$ and $\Gamma, x : \sigma \vdash s : \mathrm{M}(\tau)$. In the scope of x, put

$$r_x := \mathrm{case}\ \mathsf{CHAD}[\Gamma, x : \sigma](s)\ \mathrm{of}\ \langle q, c\rangle \to q, \qquad c_x := \mathrm{case}\ \mathsf{CHAD}[\Gamma, x : \sigma](s)\ \mathrm{of}\ \langle q, c\rangle \to c.$$

Then the bind clause is

$$\mathsf{CHAD}[\Gamma](\mathrm{bind}(t; x\ .\ s)) := \mathrm{case}\ \mathsf{CHAD}[\Gamma](t)\ \mathrm{of}\ \langle p, b\rangle \to \left\langle \mathrm{bind}(p; x\ .\ r_x)\,, \lambda v\ .\ \mathrm{bind}^w_{p;x\ .\ r_x}(b(v)\,; c_x(v))\right\rangle . \tag{100}$$

The first argument of bind^w accumulates the cotangent contribution through the sampled computation t, while the second argument accumulates the cotangent contribution through the continuation s, including the cotangent of the bound variable x.

FAD constructors and handler. For the finite atomic distribution constructor, assume $\Gamma \vdash t_i : \tau$ and $\Gamma \vdash w_i : \mathsf{real}$ for $1 \leq i \leq n$. Then

$$\begin{aligned}
&\mathsf{CHAD}[\Gamma](\mathrm{categorical}(\langle\langle t_i, w_i\rangle\rangle_{i=1}^n)) := \\
&\quad \mathrm{case}\ \mathsf{CHAD}[\Gamma](t_1)\ \mathrm{of}\ \langle u_1, u_1'\rangle \to \cdots \mathrm{case}\ \mathsf{CHAD}[\Gamma](t_n)\ \mathrm{of}\ \langle u_n, u_n'\rangle \to \\
&\quad \mathrm{case}\ \mathsf{CHAD}[\Gamma](w_1)\ \mathrm{of}\ \langle a_1, a_1'\rangle \to \cdots \mathrm{case}\ \mathsf{CHAD}[\Gamma](w_n)\ \mathrm{of}\ \langle a_n, a_n'\rangle \to \\
&\qquad \left\langle \mathrm{categorical}(\langle\langle u_i, a_i\rangle\rangle_{i=1}^n)\,, \lambda v\ .\ \mathrm{categorical}^w_{\langle\langle u_i, a_i\rangle\rangle_{i=1}^n}(\langle\langle u_i'(v)\,, a_i'(v)\rangle\rangle_{i=1}^n)\right\rangle .
\end{aligned} \tag{101}$$

Finally, for the handler algebra E_ν, where $\nu = \mathsf{real}^p$, we define

$$\mathsf{CHAD}[\Gamma](\mathrm{E}_\nu(t)) := \mathrm{case}\ \mathsf{CHAD}[\Gamma](t)\ \mathrm{of}\ \langle p, b\rangle \to \left\langle \mathrm{E}_\nu(p)\,, \lambda v\ .\ \mathrm{E}^w_{\nu,p}(b(v))\right\rangle . \tag{102}$$

D.3 Typing and universal characterization

Theorem D.1 (Well-typedness and uniqueness of the CHAD translation)**.** *For every well-typed source judgement $\Gamma \vdash t : \tau$, the term $\mathsf{CHAD}[\Gamma](t)$ defined above is well typed as in Equation (83). Moreover, these clauses determine the unique structure-preserving functor $\mathsf{CHAD} : \mathsf{Syn} \to \mathsf{Tgt}$ that preserves finite products and coproducts, sends each source monadic type $\mathrm{M}(\tau)$ to the displayed pair consisting of the primal* M*-type and its cotangent lift M^w, interprets each* op *together with its transposed Jacobian* op*, and interprets the FAD constructors and the handler algebra by the displayed pairs* $(\mathrm{categorical}, \mathrm{categorical}^w)$ *and* $(\mathrm{E}_\nu, \mathrm{E}^w_\nu)$*, respectively.*

The typing statement follows by induction on the source typing derivation. The only non-structural cases are the monadic and FAD constructors, where the typing rules for return^w, bind^w, $\mathrm{categorical}^w$, and E^w are exactly the target rules used in the displayed clauses. The uniqueness statement is the universal property of the freely generated source syntactic category (Lemma A.2) applied to the corresponding structure in Tgt. The target equations of Section B ensure that the source Kleisli, distributive, and Eilenberg–Moore equations are preserved.

E Example of differentiated code

This section gives the complete differentiated example code from Equation (58), following the inductive rules of code differentiation outlined in Appendix D. As in Section 5, we adopt the following abbreviations:

$$
\begin{aligned}
a_0 &:= 0 & \gamma_0 &:= \mathrm{lsig}(-s_m \cdot (m - m_0)) \\
a_1 &:= P \cdot \mathrm{sig}(-s_p \cdot (p - p_0)) & \gamma_1 &:= \mathrm{lsig}(s_m \cdot (m - m_0))
\end{aligned}
$$

The full code transformation is then given by

$$
\begin{array}{l}
\text{case} \\
\quad \text{case } \langle a_0, \boldsymbol{\nabla} a_0 \rangle \text{ of } \langle \theta_0, \theta_0' \rangle \to \\
\quad \text{case } \langle a_1, \boldsymbol{\nabla} a_1 \rangle \text{ of } \langle \theta_1, \theta_1' \rangle \to \\
\quad \text{case } \langle \gamma_0, \boldsymbol{\nabla} \gamma_0 \rangle \text{ of } \langle \rho_0, \rho_0' \rangle \to \\
\quad \text{case } \langle \gamma_1, \boldsymbol{\nabla} \gamma_1 \rangle \text{ of } \langle \rho_1, \rho_1' \rangle \to \\
\qquad \Big\langle \text{categorical}\Big(\langle \langle \theta_j, \rho_j \rangle \rangle_{j=0}^{1} \Big), \lambda v \,.\, \text{categorical}^{w}_{\langle\langle \theta_j, \rho_j \rangle\rangle_{j=0}^{1}} \Big(\Big\langle \Big\langle \theta_j'(v), \rho_j'(v) \Big\rangle \Big\rangle_{j=0}^{1} \Big) \Big\rangle \\
\text{of } \langle \alpha, \alpha' \rangle \to \\
\quad \text{case} \\
\qquad \text{case} \\
\qquad\quad \text{case} \Big\langle s, \lambda v \,.\, \text{coproj}_{\mathrm{idx}(s;\Gamma,s)}\, v \Big\rangle \\
\qquad\quad \text{of } \langle \eta, \eta' \rangle \to \\
\qquad\qquad \text{let}\, \xi = \\
\qquad\qquad\quad \text{case} \\
\qquad\qquad\qquad \text{case} \, \langle (p \cdot x - m), \boldsymbol{\nabla}_{p,m,x} (p \cdot x - m) \rangle \\
\qquad\qquad\qquad \text{of } \langle \mu, \mu' \rangle \to \Big\langle \text{return}(\mu), \lambda v \,.\, \text{return}^{w}_{\mu}(\mu'(v)) \Big\rangle \\
\qquad\qquad\quad \text{of } \langle \kappa, \kappa' \rangle \to \kappa \\
\qquad\qquad \text{in} \\
\qquad\qquad\quad \Big\langle \text{bind}\,(\eta; x \,.\, \xi), \lambda v \,.\, \text{bind}^{w}_{\eta; x \,.\, \xi}\,(\eta'(v); \\
\qquad\qquad\qquad (\text{case} \\
\qquad\qquad\qquad\quad \text{case} \, \langle (p \cdot x - m), \boldsymbol{\nabla}_{p,m,x} (p \cdot x - m) \rangle \\
\qquad\qquad\qquad\quad \text{of } \langle \mu, \mu' \rangle \to \Big\langle \text{return}(\mu), \lambda v \,.\, \text{return}^{w}_{\mu}(\mu'(v)) \Big\rangle \\
\qquad\qquad\qquad \text{of } \langle \kappa, \kappa' \rangle \to \kappa')\,(v)) \rangle \\
\qquad \text{of } \langle \delta, \delta' \rangle \to \\
\qquad\quad \text{case} \\
\qquad\qquad \text{case} \Big\langle t, \lambda v \,.\, \text{coproj}_{\mathrm{idx}(t;\Gamma,s,t)}\, v \Big\rangle \\
\qquad\qquad \text{of } \langle \zeta, \zeta' \rangle \to \Big\langle \mathrm{E}_{\mathsf{real}}(\zeta), \lambda v \,.\, \mathrm{E}^{w}_{\mathsf{real},\zeta}(\zeta'(v)) \Big\rangle \\
\qquad\quad \text{of } \langle \epsilon, \epsilon' \rangle \to \langle \epsilon[\delta/t], \lambda v \,.\, \mathsf{let}\ w = \epsilon'[\delta/t](v)\ \ \mathsf{in}\ \ \mathrm{pr}_1\, w + \delta'(\mathrm{pr}_2\, w) \rangle \\
\quad \text{of } \langle \beta, \beta' \rangle \to \langle \beta[\alpha/s], \lambda v \,.\, \mathsf{let}\ w = \beta'[\alpha/s](v)\ \ \mathsf{in}\ \ \mathrm{pr}_1\, w + \alpha'(\mathrm{pr}_2\, w) \rangle
\end{array}
$$

F Deriving concrete gradient estimation formulas

We now explain how the usual gradient rule for weighted particles is already contained in the lifted FAD monad and in the lifted canonical algebra.

In this section, we write the second component of a morphism in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ using the symbol ∇. Thus a morphism $\hat{h} : T^*X \to T^*Y$ is written as

$$\hat{h} = (h, \nabla h),$$

where $h : X \to Y$ is its base part and

$$(\nabla h)_x : T^*_{h(x)}Y \to T^*_x X$$

is the x-fibre of its fibre part. This notation is suggestive: ∇h denotes the chosen cotangent map in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, whether or not X and Y carry genuine smooth structure.

Write

$$\mathrm{E}^{\mathsf{F}}_{\mathbb{R}} : M^{\mathsf{F}}T^*\mathbb{R} \to T^*\mathbb{R}$$

for the lifted canonical algebra on real values. Its base part is the unnormalised expectation map

$$u \longmapsto \sum_{r \in \operatorname{supp} u} u(r)r, \qquad u \in M(\mathbb{R}),$$

and its fibre part is

$$\left(\nabla \mathrm{E}^{\mathsf{F}}_{\mathbb{R}}\right)_u (t) = \sum_{r \in \operatorname{supp} u} \left(u(r)t \cdot \mathbf{e}_r + u(r)rt \cdot \varepsilon_r\right). \tag{103}$$

Here the $\mathbf{e}_r$-component is the cotangent of the real value r, while the ε_r-component is the cotangent of the log-weight attached to r.

Let

$$T^*X = \langle X, \{T^*_x X\}_{x \in X}\rangle$$

be an object of $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and let

$$\hat{g} = (g, \nabla g) : T^*X \to T^*\mathbb{R}$$

be a real-valued test morphism.

For $w \in M(X)$, set

$$u := M(g)(w).$$

The Frobenius, or pushforward, identity says on bases that

$$\int_X g(x)\, dw = \int_{\mathbb{R}} r\, dM(g)(w),$$

or explicitly

$$\sum_{x \in \operatorname{supp} w} w(x)g(x) = \sum_{r \in \operatorname{supp} u} u(r)r.$$

In categorical form, this is the base part of the composite

$$M^{\mathsf{F}}T^*X \xrightarrow{M^{\mathsf{F}}\hat{g}} M^{\mathsf{F}}T^*\mathbb{R} \xrightarrow{\mathrm{E}^{\mathsf{F}}_{\mathbb{R}}} T^*\mathbb{R}.$$

The fibre part of this composite is obtained by composing Equation (103) with the fibre part of $M^{\mathsf{F}}\hat{g}$. For $x \in \operatorname{supp} w$, the denominator is nonzero, and the corresponding summand is

$$\frac{w(x)}{u(g(x))}\left((\nabla g)_x\left(u(g(x))t\right)\cdot \mathbf{e}_x + u(g(x))g(x)t\cdot \varepsilon_x\right).$$

Since $(\nabla g)_x$ is linear, the factor $u(g(x))$ in the atom-cotangent part cancels with the denominator; the log-weight part cancels directly. Hence

$$\left(\nabla\left(\mathrm{E}^{\mathsf{F}}_{\mathbb{R}} \circ M^{\mathsf{F}}\hat{g}\right)\right)_w(t) = \sum_{x\in\operatorname{supp} w}\left(w(x)(\nabla g)_x(t)\cdot \mathbf{e}_x + w(x)g(x)t\cdot\varepsilon_x\right). \tag{104}$$

Thus the lifted algebra and the lifted FAD monad already produce the two basic cotangent contributions: a value cotangent weighted by $w(x)$, and a log-weight cotangent weighted by $w(x)g(x)$.

Now apply this to a parameterised family of particles. Let

$$T^*\Theta = \langle \Theta, \{T^*_\theta\Theta\}_{\theta\in\Theta}\rangle, \qquad T^*Z = \langle Z, \{T^*_z Z\}_{z\in Z}\rangle,$$

and let

$$\hat{q} = (q, \nabla q) : T^*\Theta \to M^{\mathsf{F}}T^*Z$$

be a parameterised FAD, equipped with its chosen cotangent map. Locally, away from collisions of particles, choose a labelling of the support and write the base part of $\hat{q}$ as

$$q(\theta) = \sum_{i\in I} w_i(\theta)\delta_{z_i(\theta)}.$$

The corresponding strengthened FAD over $T^*\Theta \times T^*Z$ is

$$\underline{\hat{q}} = \left\langle \underline{q}, \nabla\underline{q}\right\rangle := \left(\mathrm{st}^{\mathsf{F}}\right)_{T^*\Theta, T^*Z} \circ \langle \mathrm{id}_{T^*\Theta}, \hat{q}\rangle : T^*\Theta \to M^{\mathsf{F}}(T^*\Theta \times T^*Z),$$

whose base part is

$$\underline{q}(\theta) = \sum_{i\in I} w_i(\theta)\delta_{(\theta, z_i(\theta))}.$$

In the chosen local labelling, the fibre of $M^{\mathsf{F}}(T^*\Theta \times T^*Z)$ at $\underline{q}(\theta)$ identifies with

$$\bigoplus_{i\in I}\left(\left(T^*_\theta\Theta \oplus T^*_{z_i(\theta)}Z\right)\oplus \mathbb{R}\right).$$

We write elements of this fibre as finite sums

$$\sum_{i\in I}\left(\langle d_i, a_i\rangle \cdot \mathbf{e}_{(\theta, z_i(\theta))} + s_i \cdot \varepsilon_{(\theta, z_i(\theta))}\right),$$

where

$$d_i \in T^*_\theta\Theta, \qquad a_i \in T^*_{z_i(\theta)}Z, \qquad s_i \in \mathbb{R}.$$

Let

$$\ell_i : \Theta \to \mathbb{R}, \qquad \ell_i(\theta) = \log w_i(\theta).$$

In these coordinates, the fibre part of $\underline{\hat{q}}$ is

$$\begin{aligned}(\nabla\underline{q})_\theta&\left(\sum_{i\in I}\left(\langle d_i, a_i\rangle \cdot \mathbf{e}_{(\theta, z_i(\theta))} + s_i\cdot\varepsilon_{(\theta, z_i(\theta))}\right)\right)\\ &= \sum_{i\in I} d_i + \sum_{i\in I}(\nabla z_i)_\theta(a_i) + \sum_{i\in I}(\nabla \ell_i)_\theta(s_i).\end{aligned} \tag{105}$$

Here

$$(\nabla z_i)_\theta : T^*_{z_i(\theta)}Z \to T^*_\theta\Theta$$

is the θ-fibre of the fibre part of the chosen morphism $\hat{z}_i = (z_i, \nabla z_i) : T^*\Theta \to T^*Z$ representing the parameterised particle, and

$$(\nabla \ell_i)_\theta : \mathbb{R} \to T^*_\theta\Theta$$

is the θ-fibre of the fibre part of the chosen morphism $\hat{\ell}_i = (\ell_i, \nabla\ell_i) : T^*\Theta \to T^*\mathbb{R}$ representing the parameterised log-weight. Thus an atom cotangent is sent backwards through the moving particle z_i, while a log-weight cotangent is sent backwards through $\ell_i = \log w_i$.

Finally, let

$$\widehat{f} = (f, \nabla f) : T^*\Theta \times T^*Z \to T^*\mathbb{R}$$

be a parameter-dependent test morphism, with base part

$$(\theta, z) \longmapsto f(\theta, z).$$

We write the (θ, z)-fibre of its fibre part as

$$(\nabla f)_{(\theta,z)}(t) = \left\langle (\nabla f)^{\Theta}_{(\theta,z)}(t), (\nabla f)^{Z}_{(\theta,z)}(t) \right\rangle,$$

where

$$(\nabla f)^{\Theta}_{(\theta,z)}(t) \in T^*_\theta\Theta, \qquad (\nabla f)^{Z}_{(\theta,z)}(t) \in T^*_z Z.$$

Consider the composite

$$T^*\Theta \xrightarrow{\hat{q}} M^{\mathsf{F}}(T^*\Theta \times T^*Z) \xrightarrow{M^{\mathsf{F}}\widehat{f}} M^{\mathsf{F}}T^*\mathbb{R} \xrightarrow{\mathrm{E}^{\mathsf{F}}_{\mathbb{R}}} T^*\mathbb{R}.$$

By the Frobenius identity, its base part is

$$\theta \longmapsto \sum_{i \in I} w_i(\theta) f(\theta, z_i(\theta)).$$

Applying Equation (104) to $\widehat{f} : T^*\Theta \times T^*Z \to T^*\mathbb{R}$, an output cotangent $t \in \mathbb{R}$ is first sent to the following cotangent of the FAD over $\Theta \times Z$:

$$\sum_{i\in I} \Big(w_i(\theta) \left\langle (\nabla f)^{\Theta}_{(\theta,z_i(\theta))}(t), (\nabla f)^{Z}_{(\theta,z_i(\theta))}(t) \right\rangle \cdot \mathbf{e}_{(\theta,z_i(\theta))} + w_i(\theta) f(\theta, z_i(\theta)) t \cdot \varepsilon_{(\theta,z_i(\theta))} \Big).$$

Composing with Equation (105) therefore gives

$$\begin{aligned} &\Big(\nabla\Big(\mathrm{E}^{\mathsf{F}}_{\mathbb{R}} \circ M^{\mathsf{F}}\widehat{f} \circ \underline{\hat{q}}\Big)\Big)_\theta (t) \\ &= \sum_{i\in I} w_i(\theta)(\nabla f)^{\Theta}_{(\theta,z_i(\theta))}(t) + \sum_{i\in I} w_i(\theta)(\nabla z_i)_\theta \left((\nabla f)^{Z}_{(\theta,z_i(\theta))}(t)\right) \\ &\quad + \sum_{i\in I} (\nabla\ell_i)_\theta \left(w_i(\theta) f(\theta, z_i(\theta)) t\right). \end{aligned} \tag{106}$$

The three summands in Equation (106) correspond to three different cotangent routes. The first term comes from the $T^*_\theta\Theta$-component of the test morphism $\widehat{f} : T^*\Theta \times T^*Z \to T^*\mathbb{R}$, and represents the explicit dependence of f on the parameter. The second term comes from the

$$T^*_{z_i(\theta)}Z$$

part of the atom component inside the fibre of $M^{\mathsf{F}}(T^*\Theta \times T^*Z)$, and represents how moving the particle $z_i(\theta)$ changes the value of the test function. The third term comes from the real component

$$\mathbb{R}$$

inside the same fibre, and represents how changing the log-weight $\ell_i(\theta) = \log w_i(\theta)$ changes the weighted sum. This third contribution is the usual score, or likelihood-ratio, term.

If the weights are already normalised, $w_i(\theta) = p_i(\theta)$, and we write

$$\ell_i(\theta) = \log p_i(\theta),$$

then Equation (106) gives

$$\begin{aligned}&(\nabla(\theta \mapsto \mathbb{E}_{i\sim p_\theta}[f(\theta, z_i(\theta))]))_\theta\,(t)\\ &\quad= \mathbb{E}_{i\sim p_\theta}\left[(\nabla f)^{\Theta}_{(\theta,z_i(\theta))}(t) + (\nabla z_i)_\theta\left((\nabla f)^{Z}_{(\theta,z_i(\theta))}(t)\right) + f(\theta, z_i(\theta))(\nabla \ell_i)_\theta(t)\right].\end{aligned}$$

If normalised probabilities are instead obtained from unnormalised weights by a separate normalisation map, let

$$p_i(\theta) = \frac{w_i(\theta)}{\sum_{j\in I} w_j(\theta)}, \qquad \ell_i(\theta) = \log w_i(\theta).$$

Then the lifted fibre map of that normalisation gives

$$\begin{aligned}&(\nabla(\theta \mapsto \mathbb{E}_{i\sim p_\theta}[f(\theta, z_i(\theta))]))_\theta\,(t)\\ &\quad= \mathbb{E}_{i\sim p_\theta}\Big[(\nabla f)^{\Theta}_{(\theta,z_i(\theta))}(t) + (\nabla z_i)_\theta\left((\nabla f)^{Z}_{(\theta,z_i(\theta))}(t)\right)\\ &\qquad\quad + (f(\theta, z_i(\theta)) - \mathbb{E}_{j\sim p_\theta}[f(\theta, z_j(\theta))])\,(\nabla \ell_i)_\theta(t)\Big].\end{aligned}$$

G Algebras and strong morphisms into the continuation monad

We give a proof of Lemma 2.1. Throughout this appendix, let

$$K := \mathsf{K}V^{\mathsf{X}} = [[-, V], V].$$

We use the internal language of the cartesian closed category X. Thus the elementwise formulas below denote the corresponding morphism equalities obtained by currying and evaluation.

Recall first the continuation monad structure. For a morphism $f : A \to B$, the functorial action of K is contravariant precomposition twice:

$$K(f)\,(c)(\ell) = c(\ell \circ f), \qquad c : K(A), \quad \ell : [B, V].$$

Its unit and multiplication are

$$\left(\eta^K\right)_A(x)(k) = k(x), \qquad x : A, \quad k : [A, V],$$

and

$$\left(\mu^K\right)_A(\Phi)(k) = \Phi(c \mapsto c(k)), \qquad \Phi : K(K(A)), \quad k : [A, V].$$

The canonical strength of K is

$$\left(\mathrm{st}^K\right)_{A,B}(x,c)(h) = c(b \mapsto h(x,b)),$$

where $x : A$, $c : K(B)$, and $h : [A \times B, V]$.

A T-algebra structure on V is a morphism

$$a : T(V) \to V$$

satisfying

$$a \circ \left(\eta^T\right)_V = \mathrm{id}_V \qquad \text{and} \qquad a \circ \left(\mu^T\right)_V = a \circ T(a)\,.$$

Given such an algebra a, define a natural transformation

$$\theta^a : T \to K$$

as follows. Its component at A is the transpose of the composite

$$T(A) \times [A,V] \xrightarrow{\cong} [A,V] \times T(A) \xrightarrow{(\mathrm{st}^T)_{[A,V],A}} T([A,V] \times A) \xrightarrow{T(\mathrm{ev}_{A,V})} T(V) \xrightarrow{a} V,$$

where $\mathrm{ev}_{A,V} : [A,V] \times A \to V$ is evaluation. Equivalently,

$$(\theta^a)_A\,(t)(k) = a(T(k)\,(t)), \qquad t : T(A)\,, \quad k : [A,V]\,.$$

Naturality in A follows immediately from functoriality of T: for $f : A \to B$ and $\ell : [B,V]$,

$$(\theta^a)_B\,(T(f)\,(t))(\ell) = a(T(\ell \circ f)\,(t)) = K(f)\,((\theta^a)_A\,(t))(\ell).$$

The unit law for a monad morphism is

$$\begin{aligned}
(\theta^a)_A\,(\left(\eta^T\right)_A(x))(k) &= a(\left(\eta^T\right)_V(k(x))) \\
&= k(x) \\
&= \left(\eta^K\right)_A(x)(k).
\end{aligned}$$

For multiplication, let $u : T(T(A))$. Then

$$\begin{aligned}
(\theta^a)_A\,(\left(\mu^T\right)_A(u))(k) &= a(T(k)\,(\left(\mu^T\right)_A(u))) \\
&= a(\left(\mu^T\right)_V(T(T(k))\,(u))) \\
&= a(T(a)\,(T(T(k))\,(u))) \\
&= a(T((a \circ T(k)))\,(u)) \\
&= (\theta^a)_{T(A)}\,(u)(s \mapsto (\theta^a)_A\,(s)(k)) \\
&= (\left(\mu^K\right)_A \circ K((\theta^a)_A) \circ (\theta^a)_{T(A)})(u)(k).
\end{aligned}$$

Thus θ^a is a monad morphism.

It remains to check that it is strong. Let $x : A$, $t : T(B)$, and $h : [A \times B, V]$. Using the coherence of the strength of T, applying h after adding the pure parameter x to the computation t is the same as applying the function $b \mapsto h(x,b)$ to the computation t. Hence

$$\begin{aligned}
(\theta^a)_{A\times B}\,(\left(\mathrm{st}^T\right)_{A,B}(x,t))(h) &= a(T(h)\,(\left(\mathrm{st}^T\right)_{A,B}(x,t))) \\
&= a(T((b \mapsto h(x,b)))\,(t)) \\
&= (\theta^a)_B\,(t)(b \mapsto h(x,b)) \\
&= \left(\mathrm{st}^K\right)_{A,B}(x, (\theta^a)_B\,(t))(h).
\end{aligned}$$

Therefore θ^a preserves strength.

Conversely, suppose that

$$\theta : T \to K$$

is a strong monad morphism. Define

$$a^\theta : T(V) \to V$$

by evaluating $(\theta)_V$ at the identity continuation:

$$a^\theta := \mathrm{ev}_{\mathrm{id}_V} \circ (\theta)_V \,.$$

In internal notation,

$$a^\theta(t) = (\theta)_V\,(t)(\mathrm{id}_V).$$

Here $\mathrm{ev}_{\mathrm{id}_V}$ denotes evaluation at the point $\mathrm{id}_V : 1 \to [V, V]$ corresponding to the identity morphism on V.

The algebra unit law follows from the unit law for the monad morphism θ:

$$\begin{aligned} a^\theta(\left(\eta^T\right)_V(v)) &= (\theta)_V\,(\left(\eta^T\right)_V(v))(\mathrm{id}_V) \\ &= \left(\eta^K\right)_V(v)(\mathrm{id}_V) \\ &= v. \end{aligned}$$

For multiplication, let $u : T(T(V))$. Then

$$\begin{aligned} a^\theta(\left(\mu^T\right)_V(u)) &= (\theta)_V\,(\left(\mu^T\right)_V(u))(\mathrm{id}_V) \\ &= \left(\mu^K\right)_V(K((\theta)_V)\,((\theta)_{T(V)}\,(u)))(\mathrm{id}_V) \\ &= (\theta)_{T(V)}\,(u)(s \mapsto (\theta)_V\,(s)(\mathrm{id}_V)) \\ &= (\theta)_{T(V)}\,(u)(a^\theta) \\ &= (\theta)_V\,(T\left(a^\theta\right)(u))(\mathrm{id}_V) \\ &= a^\theta(T\left(a^\theta\right)(u)). \end{aligned}$$

Thus a^θ is a T-algebra structure on V.

Finally, the two constructions are inverse. Starting from an algebra a, we have

$$a^{\theta^a}(t) = (\theta^a)_V\,(t)(\mathrm{id}_V) = a(T(\mathrm{id}_V)\,(t)) = a(t).$$

Starting from a strong monad morphism θ, let $t : T(A)$ and $k : [A, V]$. By the definition of θ^{a^θ}, by naturality of θ, and then by preservation of strength, we obtain

$$\begin{aligned} \left(\theta^{a^\theta}\right)_A(t)(k) &= a^\theta\left(T(\mathrm{ev}_{A,V})\,(\left(\mathrm{st}^T\right)_{[A,V],A}(k, t))\right) \\ &= (\theta)_{[A,V]\times A}\,(\left(\mathrm{st}^T\right)_{[A,V],A}(k, t))(\mathrm{ev}_{A,V}) \\ &= \left(\mathrm{st}^K\right)_{[A,V],A}(k, (\theta)_A\,(t))(\mathrm{ev}_{A,V}) \\ &= (\theta)_A\,(t)(k). \end{aligned}$$

Hence $\theta^{a^\theta} = \theta$. Therefore T-algebra structures on V are in one-to-one correspondence with strong monad morphisms $T \to \mathsf{K}V^{\mathsf{X}}$, as claimed.

H Other algebraic effects

This appendix gives the generic CHAD technique for discrete-output effects including the ones mentioned in Section 6. The main body develops the case of the FAD monad in detail. Here we identify the reusable part: if an algebraic effect only has values returned to continuations through discrete output types, then the raw AST monad has a fibred polynomial lift and the CHAD proof extends uniformly as soon as we give a compatible lifted generic effects and a compatible lifted handler algebra. We assume some standard background about (many-sorted) Lawvere theories, algebraic effects, and continuation monads Hyland and Power (2007); Plotkin and Power (2003); Hyland et al. (2007).

The presentation is organised as follows. We first explain what we mean by discrete-output effects and we specify what data need to be supplied to apply CHAD to them. Then we state the source and target language extensions and give a generic CHAD code transformation and correctness theorem. We next specialise to the three concrete instances from Section 6. Finally, we demonstrate why we impose the limitation to non-discrete outputs.

H.1 Discrete-output setup and admissible data

H.1.1 Polynomial ASTs with discrete outputs

Let Σ be an algebraic signature with a small set of sorts. A generic effect symbol is written

$$\omega : I_\omega \to TO_\omega.$$

Here, we call I_ω the effect parameter type or input type and O_ω the effect output type.

Equivalently, with a **Set** interpretation, it determines by Yoneda algebraic-operation families

$$[\![I_\omega]\!]_{\mathbf{Set}} \times X^{[\![O_\omega]\!]_{\mathbf{Set}}} \to TX,$$

and

$$[\![I_\omega]\!]_{\mathbf{Set}} \times (TX)^{[\![O_\omega]\!]_{\mathbf{Set}}} \to TX.$$

We see that O_ω gets the role of the type of values returned to the continuation.

Write

$$\widehat{I_\omega} = [\![I_\omega]\!]_{\mathsf{F}}$$

for the chosen interpretation in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, with base set $[\![I_\omega]\!]_{\mathbf{Set}}$. We allow I_ω to be a genuine differentiable type in the sense that it may have non-trivial cotangent fibres. We require O_ω to be a discrete type, in the sense that it has zero cotangent fibres:

$$[\![O_\omega]\!]_{\mathsf{F}} = \Delta[\![O_\omega]\!]_{\mathbf{Set}} = \langle [\![O_\omega]\!]_{\mathbf{Set}}, \underline{1} \rangle\,.$$

For the CHAD rules below we use the finitary case, so $[\![O_\omega]\!]_{\mathbf{Set}}$ is finite; more generally the same formula applies whenever the required products and direct sums exist.

This induces, before quotienting by equations, the raw syntax tree monad with the polynomial form

$$\mathsf{AST}_\Sigma X \simeq \mu Y.\; X \sqcup \coprod_{\omega\in\Sigma} [\![I_\omega]\!]_{\mathbf{Set}} \times Y^{[\![O_\omega]\!]_{\mathbf{Set}}} \simeq \mu Y.\; X \sqcup \coprod_{\omega\in\Sigma} [\![I_\omega]\!]_{\mathbf{Set}} \times \prod_{o\in[\![O_\omega]\!]_{\mathbf{Set}}} Y.$$

The corresponding fibred polynomial lift to $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, for an object $A \in \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ is

$$\widehat{\mathsf{AST}}_{\Sigma} A \simeq \mu B.\ A \sqcup \coprod_{\omega \in \Sigma} \widehat{I}_{\omega} \times B^{\Delta[\![O_\omega]\!]_{\mathbf{Set}}}.$$

Its base is canonically $\mathsf{AST}_\Sigma((A)_{\mathrm{set}})$. More explicitly, over a tree-node base element $\omega(i,k)$, where $i \in [\![I_\omega]\!]_{\mathbf{Set}}$ and $k : [\![O_\omega]\!]_{\mathbf{Set}} \to (B)_{\mathrm{set}}$, the fibre is

$$\left(\widehat{I_\omega}\right)_{\mathrm{fbr}}(i) \oplus \bigoplus_{o \in [\![O_\omega]\!]_{\mathbf{Set}}} (B)_{\mathrm{fbr}}(k(o))\,.$$

This is the point at which the discreteness of O_ω is used: a continuation branch is indexed only by the set-level value o, and the variable $o : O_\omega$ contributes no cotangent of its own. The same polynomial construction applies componentwise to LR.

Thus continuous operation-inputs are harmless, provided they are treated as parameters with their own cotangent fibres; what matters for this direct fibred-polynomial construction is discreteness of the operation-output object O_ω. This covers finite choice, exception operations $\mathrm{raise} : E \to T_E 0$, writer operations such as $\mathrm{tell} : \mathbb{R}^n \to W_{\mathbb{R}^n} 1$, and state/reader operations over finite discrete state/reader objects. It does not directly cover continuous state, continuous reader, or continuous probability, where the canonical operations have genuine output objects with nonzero fibres. For such an output object $\widehat{O}$, a continuation is a morphism $\widehat{O} \to B$, not merely a function $[\![O]\!]_{\mathbf{Set}} \to (B)_{\mathrm{set}}$; hence the simple set-indexed product $B^{\Delta[\![O]\!]_{\mathbf{Set}}}$ above no longer describes the desired continuation object.

H.1.2 Admissible lifted data

A *CHAD-admissible discrete-output presentation* consists of the following data.

First, we assume a signature Σ as above and a set of equations E presenting a strong monad T on **Set** as a quotient of the raw tree monad,

$$q : \mathsf{AST}_\Sigma \twoheadrightarrow T,$$

with q a morphism of strong monads. Equivalently, T is the monad presented by the generic effects of Σ modulo E. (In fact, the equations E are often precisely obtained from the algebra/handler below.)

Second, we assume a chosen handler carrier type ν and a T-algebra

$$h : T\nu \to \nu.$$

This algebra is what discharges the effect in real-output programs. (We can also allow a family of handler carriers and handler algebras, in which case we obtain a strong monad morphism from ASTs to a product of continuation monads, but we avoid this generality to simplify the presentation.)

Third, we assume chosen strong monad lifts

$$T^{\mathsf{F}}, \quad T^{\mathsf{LR}}$$

over T, in the sense that their projections to the underlying set semantics are naturally isomorphic to T and the isomorphisms respect unit, multiplication, and strength. For each operation symbol ω, these lifted monads are equipped with lifted generic effects, for example in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$

$$\widehat{\omega}^{\mathsf{F}}_A : \widehat{I}_\omega \times \left(T^{\mathsf{F}} A\right)^{\Delta[\![O_\omega]\!]_{\mathbf{Set}}} \to T^{\mathsf{F}} A,$$

and analogously in LR. Under the base projection, $\widehat{\omega}^{\mathsf{F}}_A$ is required to be the set-level generic-effect constructor

$$[\![I_\omega]\!]_{\mathbf{Set}} \times (T((A)_{\mathrm{set}}))^{[\![O_\omega]\!]_{\mathbf{Set}}} \to T((A)_{\mathrm{set}})$$

induced by q; the LR-operation satisfies the corresponding logical-relations compatibility. The lifted operations must satisfy the equations E and be coherent with the lifted unit, multiplication, and strength.

Finally, we assume lifted handler algebras

$$h^{\mathsf{F}} : T^{\mathsf{F}} \nu^{\mathsf{F}} \to \nu^{\mathsf{F}}, \qquad h^{\mathsf{LR}} : T^{\mathsf{LR}} \nu^{\mathsf{LR}} \to \nu^{\mathsf{LR}}.$$

They are required to be Eilenberg–Moore algebras for the lifted monads and to project to the set-level handler algebra h, with the evident logical-relations compatibility for h^{LR}.

These assumptions are exactly the data consumed by the CHAD proof. The proof does not use how the lifted monad was constructed. It may be obtained by a direct quotient/coimage construction, by an image construction, or directly from products and coproducts, provided the resulting operations and handler satisfy the compatibility conditions just listed.

H.2 Language extension and CHAD theorem

Fix a CHAD-admissible discrete-output presentation as in Section H.1. The monadic type M is now interpreted by the presented monad T, together with its chosen lifts T^{F} and T^{LR}. The rules below do not construct these lifts from the set-level operations alone; they expose the backwards fibre maps of the chosen lifted generic effects and lifted handler algebras. This point is essential when an operation parameter I_ω has nonzero cotangent fibres. All equations involving the added operations are interpreted modulo the equations E of the presentation and, in the target, through the corresponding lifted operations.

H.2.1 Source operations, equations, and handlers

For every generic effect symbol $\omega : I_\omega \to TO_\omega$, the source language is extended by the operation form

$$\frac{\Gamma \vdash i : I_\omega \qquad \Gamma, o : O_\omega \vdash k_o : \mathrm{M}(\tau)}{\Gamma \vdash \mathrm{op}_\omega(i;\, o.k_o) : \mathrm{M}(\tau)}\ \mathrm{Op}_\omega.$$

The binder $o.k_o$ denotes a finite family of continuation terms indexed by $[\![O_\omega]\!]_{\mathbf{Set}}$. Since O_ω is discrete, extending the context by $o : O_\omega$ changes the base context but adds no cotangent component. Write

$$\omega_X(i, k) := \mu_X\big(T(k)(\omega(i))\big), \qquad k : [\![O_\omega]\!]_{\mathbf{Set}} \to TX,$$

for the quotient-induced Kleisli constructor associated with the generic effect $\omega : I_\omega \to TO_\omega$. The set-level semantics is

$$[\![\mathrm{op}_\omega(i;\, o.k_o)]\!]_{\mathbf{Set}}(\gamma) = \omega_{[\![\tau]\!]_{\mathbf{Set}}}\left([\![i]\!]_{\mathbf{Set}}(\gamma), o \mapsto [\![k_o]\!]_{\mathbf{Set}}(\gamma, o)\right).$$

The source equational theory is extended in three ways. First, the equations E presenting T are imposed on the generic effects. Secondly, the operation form is algebraic, i.e. it commutes with Kleisli extension. Explicitly, for $\Gamma, x : \tau \vdash s : \mathrm{M}(\sigma)$, we impose the equation schema

$$\mathrm{bind}(\mathrm{op}_\omega(i;\, o.k_o)\,; x.s) \equiv \mathrm{op}_\omega(i;\, o.\,\mathrm{bind}(k_o; x.s))\,. \tag{107}$$

The right-hand side is well typed because the variable $o : O_\omega$ is in scope in the continuation branch. Finally, the handler associated with the chosen algebra $h : T\nu \to \nu$ is added as

$$\frac{\Gamma \vdash t : \mathrm{M}(\nu)}{\Gamma \vdash \mathrm{handle}_h(t) : \nu}\,\textsc{Handle}_h,$$

and satisfies the Eilenberg–Moore algebra laws. In term notation, for $\Gamma \vdash v : \nu$, $\Gamma \vdash t : \mathrm{M}(\sigma)$, and $\Gamma, x : \sigma \vdash u : \mathrm{M}(\nu)$, these are the equation schemas

$$\begin{aligned} \mathrm{handle}_h(\mathrm{return}(v)) &\equiv v, \\ \mathrm{handle}_h(\mathrm{bind}(t; x.u)) &\equiv \mathrm{handle}_h(\mathrm{bind}(t; x.\,\mathrm{return}(\mathrm{handle}_h(u))))\,. \end{aligned} \tag{108}$$

These laws are exactly the assertion that handle_h interprets the T-algebra h; the second equation is $h \circ \mu = h \circ Th$ written in Kleisli form.

H.2.2 Linear target operations

The target language contains, for each generic effect ω, a linear constructor

$$\mathrm{op}^w_{\omega,i;\,o.k_o}(u.i'\ ;\ o.z.k'_o)\,.$$

The notation $\underline{I_\omega}$ in the rule below denotes the fibre of the chosen parameter object $\widehat{I_\omega}$; if I_ω is discrete, this is the zero linear type. The constructor represents the reverse fibre component of the chosen lifted operation (the transposed derivative in differentiable examples)

$$\widehat{\omega}^{\mathsf{F}}_A : \widehat{I_\omega} \times \left(T^{\mathsf{F}}A\right)^{\Delta[\![O_\omega]\!]_{\mathbf{Set}}} \to T^{\mathsf{F}}A,$$

and similarly in LR. Thus, for $i \in [\![I_\omega]\!]_{\mathbf{Set}}$ and a branch family $p : [\![O_\omega]\!]_{\mathbf{Set}} \to \left(T^{\mathsf{F}}A\right)_{\mathrm{set}}$, writing $p_o = p(o)$, its reverse fibre map is

$$\left(T^{\mathsf{F}}A\right)_{\mathrm{fbr}}(\omega(i,p)) \longrightarrow \left(\widehat{I_\omega}\right)_{\mathrm{fbr}}(i) \oplus \bigoplus_{o \in [\![O_\omega]\!]_{\mathbf{Set}}} \left(T^{\mathsf{F}}A\right)_{\mathrm{fbr}}(p(o))\,,$$

where $\omega(i,p) = \mu_{(A)_{\mathrm{set}}}(T(p)(\omega(i)))$ is the induced Kleisli constructor. In one-sorted notation it has the following rule, in the same continuation-passing style as categorical^w:

$$\frac{\begin{array}{c}\Gamma, y : I_\omega \vdash \underline{I_\omega}\ \text{type} \qquad \Gamma, x : \tau \vdash \underline{\tau}\ \text{type} \qquad \Gamma \vdash \underline{\rho}\ \text{type} \\ \Gamma \vdash i : I_\omega \quad \Gamma, o : O_\omega \vdash k_o : \mathrm{M}(\tau) \quad \Gamma; u : \underline{I_\omega}[i/y] \vdash i' : \underline{\rho} \quad \Gamma, o : O_\omega; z : \mathrm{M}^w_{k_o:\mathrm{M}(\tau)}\,[x \mapsto \underline{\tau}] \vdash k'_o : \underline{\rho}\end{array}}{\Gamma; v : \mathrm{M}^w_{\mathrm{op}_\omega(i;\,o.k_o):\mathrm{M}(\tau)}\,[x \mapsto \underline{\tau}] \vdash \mathrm{op}^w_{\omega,i;\,o.k_o}(u.i'\ ;\ o.z.k'_o) : \underline{\rho}} \tag{109}$$

The variable u receives the cotangent contribution for the operation parameter, while z receives the cotangent contribution for the corresponding continuation branch. Since O_ω is discrete, extending the context by $o : O_\omega$ does not add a linear component; hence each k'_o returns a cotangent for the same ambient context Γ. The finite direct sum above is where the finiteness of O_ω enters the target syntax. Semantically, if the reverse fibre map sends the incoming monadic cotangent v to $\left\langle u, (z_o)_{o \in [\![O_\omega]\!]_{\mathbf{Set}}} \right\rangle$, then the constructor denotes the finite sum

$$i'(u) + \sum_{o \in [\![O_\omega]\!]_{\mathbf{Set}}} k'_o(z_o) \, .$$

For $O_\omega = 0$, this branch sum is empty.

With this operation former, the CHAD clause for algebraic operations is the following. We use the canonical identification

$$\mathsf{CHAD}(\Gamma, o : O_\omega)_2 \cong \mathsf{CHAD}(\Gamma)_2 \, ,$$

since O_ω is discrete and has zero cotangent fibres (syntactically, $\underline{O_\omega}$ is the terminal zero-cotangent type $\underline{1}$). Then

$$\begin{aligned} &\mathsf{CHAD}[\Gamma](\mathrm{op}_\omega(i;\, o.k_o)) := \\ &\quad \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](i)\ \ \mathrm{of}\ \ \langle i_0, i' \rangle \to \\ &\quad \mathrm{let}\ \ \langle r_o, r'_o \rangle = \mathsf{CHAD}[\Gamma, o : O_\omega](k_o)\ \ \mathrm{in} \\ &\qquad \left\langle \mathrm{op}_\omega(i_0;\, o.r_o)\, , \lambda v\ .\ \mathrm{op}^w_{\omega, i_0;\, o.r_o}(u.i'(u)\ \ ;\ \ o.z.r'_o(z)) \right\rangle . \end{aligned} \tag{110}$$

The let-binding of $\langle r_o, r'_o \rangle$ is meta-notation for the family obtained by translating the continuation in the context $\Gamma, o : O_\omega$. The incoming cotangent v is consumed by op^w_ω, which separates it into the parameter cotangent and the branch cotangents; the backpropagators i' and r'_o then return these contributions to the ambient context. No cotangent for o is omitted: there is none.

H.2.3 Linear handler operations

Similarly, the target language contains a linear constructor $\mathrm{handle}^w_{h,p}$. To avoid overloading the incoming output cotangent with the monadic cotangent produced by the handler algebra, we write this constructor with an explicit binder $w.s$:

$$\frac{\Gamma, x : \nu \vdash \underline{\nu}\ \mathrm{type} \qquad \Gamma \vdash \underline{\rho}\ \mathrm{type} \\ \Gamma \vdash p : \mathrm{M}(\nu) \quad \Gamma; w : \mathrm{M}^w_{p:\mathrm{M}(\nu)}\,[x \mapsto \underline{\nu}] \vdash s : \underline{\rho}}{\Gamma; v : \underline{\nu}[\mathrm{handle}_h(p)\,/x] \vdash \mathrm{handle}^w_{h,p}(w.s) : \underline{\rho}} \tag{111}$$

Semantically, if $B = \nu^{\mathsf{F}}$, the lifted handler algebra $h^{\mathsf{F}} : T^{\mathsf{F}} B \to B$ has reverse fibre map

$$(B)_{\mathrm{fbr}}(h(p)) \longrightarrow \left(T^{\mathsf{F}} B\right)_{\mathrm{fbr}}(p) \, .$$

The constructor $\mathrm{handle}^w_{h,p}$ binds w to the image of v under this reverse fibre map and then evaluates s. Thus it uses the supplied lifted algebra, not merely the underlying set map h. In the FAD instance $\nu = \mathsf{real}^p$, this is the same continuation-passing rule as Equation (43); the explicit binder only makes the two cotangent types visible.

As for FAD, we again impose the transposed derivatives of the equations for the operations and handlers from the source language, e.g., the Eilenberg–Moore laws for the handler algebra.

The CHAD clause is the exact analogue of the canonical FAD handler clause:

$$\mathsf{CHAD}[\Gamma](\mathrm{handle}_h(t)) := \mathrm{case}\ \ \mathsf{CHAD}[\Gamma](t)\ \ \mathrm{of}\ \ \langle p, b \rangle \to \left\langle \mathrm{handle}_h(p)\, , \lambda v\ .\ \mathrm{handle}^w_{h,p}(w.b(w)) \right\rangle . \tag{112}$$

Here b is a backpropagator expecting a monadic cotangent over p. The incoming output cotangent v is first pulled back along the lifted handler algebra to the monadic cotangent w, and only then passed to b.

H.2.4 The generic CHAD result

Theorem H.1 (CHAD for discrete-output algebraic effects)**.** *For every CHAD-admissible discrete-output presentation, the source and target languages, the CHAD translation, and the denotational semantics of Sections 3.1–3.4 extend to the operations of Σ and to the handler h. The fundamental logical-relations lemma continues to hold for the extended language. Consequently, for every well-typed real-output program whose added effects are discharged by the chosen handler, the conclusion of Theorem 4.1 continues to hold.*

Proof. The source interpretation sends op_ω to the quotient-induced Kleisli constructor ω_X, and sends handle_h to the algebra $h : T\nu \to \nu$. The equations E, the algebraicity law (107), and the Eilenberg–Moore laws (108) are sound by construction.

The target interpretation sends M^w, op^w_ω, and $\mathrm{handle}^w_{h,p}$ to the fibre components of the chosen lifted monads, lifted generic effects, and lifted handler algebras in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. The LR-interpretation uses the corresponding supplied LR-data. Hence the target equations are sound because these data are assumed to be lifted strong monads, lifted operations satisfying E, and lifted Eilenberg–Moore algebras.

It remains to check the fundamental lemma for the new typing rules. In the operation case, the induction hypotheses give the related parameter and the related family of branch computations. Applying the lifted generic effect in LR gives the related transformed computation. Its fibre component is precisely the operation represented syntactically by op^w_ω. Because O_ω is discrete, the branch variable contributes no cotangent; the fibre component therefore produces only the parameter cotangent and the finite family of branch cotangents appearing in Equation (110). The displayed backpropagators are exactly the chain rule for these components.

In the handler case, the LR-lifted algebra over h maps a related monadic computation to a related handled value, and its fibre component is represented by $\mathrm{handle}^w_{h,p}$. Thus Equation (112) is again the chain rule. All other cases are the original cases of Theorem 4.1, so the correctness statement follows. □

H.3 Concrete instances

The following examples instantiate the admissible data of Section H.1. In each case the point to check is not merely that the set-level monad is algebraic, but that the chosen fibred monad, the lifted generic effects, and the lifted handler are genuine Eilenberg–Moore data over the set-level ones. We display the $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$-fibre maps; the LR-lifts used in the logical relations proof are obtained by the same constructions with the relation data carried componentwise.

H.3.1 Finite non-determinism via the finite multiset monad

We model multiplicity-sensitive finite non-determinism by the finite multiset monad

$$\mathsf{MFin}X = \coprod_{k \in \mathbb{N}} \left(X^k / \mathfrak{S}_k \right),$$

where $\mathfrak{S}_k$ acts by permuting the entries of a k-tuple. Multiplicity is retained but order is forgotten. If one additionally quotients by idempotence, one obtains the finite-powerset variant of non-determinism, whose algebras are join-semilattices with bottom rather than commutative monoids. We use the multiset, not the idempotent, variant below; in particular, the additive handler sum_n below would not factor through the idempotent quotient unless the chosen target algebra were idempotent.

The unit sends x to the singleton multiset $\{\{x\}\}$, and bind maps each occurrence to a finite multiset and then takes multiset union. An algebra $a : \mathsf{MFin}A \to A$ is equivalently a commutative monoid object on A. Given a, the unit is $a(\varnothing)$ and multiplication is $(a_1, a_2) \mapsto a(\{\{a_1, a_2\}\})$. Conversely, a commutative monoid folds finite multisets by iterated multiplication; the algebra laws are exactly unit, associativity, and commutativity.

The generic-operation presentation may be taken to have operations

$$\mathrm{choose}_k : 1 \to \mathsf{MFin}\{1, \dots, k\}, \qquad * \mapsto \{\{1, \dots, k\}\}\,,$$

for $k \in \mathbb{N}$, modulo the usual unit, associativity, and permutation equations. The induced Kleisli constructor sends a finite family of branch computations to their multiset union; $k = 0$ gives the empty choice.

The quotient by permutations must also be reflected in the fibres. Let $\langle X, A\rangle \in \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, let $m \in \mathsf{MFin}X$, and choose a representative tuple $\vec{x} = (x_1, \dots, x_k)$ of m. The free commutative-monoid lift obtained by quotienting the lifted list monad has fibre

$$\left(\mathsf{MFin}^{\mathsf{F}}\,\langle X, A\rangle\right)_{\mathrm{fbr}}(m) = \left(\bigoplus_{i=1}^{k} A(x_i)\right)^{\mathrm{Stab}(\vec{x})}\,, \tag{113}$$

where $\mathrm{Stab}(\vec{x}) \le \mathfrak{S}_k$ is the subgroup permuting equal entries of $\vec{x}$, acting by the corresponding symmetries of the finite direct sum. This description is independent of the representative. Equivalently, if m is written as a finite-support multiplicity function, then the fibre is canonically the direct sum over the support,

$$\left(\mathsf{MFin}^{\mathsf{F}}\,\langle X, A\rangle\right)_{\mathrm{fbr}}(m) \cong \bigoplus_{x \in \mathrm{supp}(m)} A(x)\,,$$

with the quotient map from ordered lists copying the support cotangent to every indistinguishable occurrence. This last copying is essential: the tempting formula with an independent summand for every occurrence is a labelled-trace semantics before quotienting by stabilisers, not the canonical free commutative-monoid lift on the finite-multiset quotient.

For a morphism $\langle f, \alpha\rangle : \langle X, A\rangle \to \langle Y, B\rangle$, the base map is multiset push-forward. Under the support description, the fibre map over $m \in \mathsf{MFin}X$ sends

$$(b_y)_{y \in \mathrm{supp}(\mathsf{MFin}f(m))} \longmapsto \left((\alpha)_x\,(b_{f(x)})\right)_{x \in \mathrm{supp}(m)}.$$

Thus a cotangent at a target support point is copied to all source support points that map to it, and the quotient map then copies it to the corresponding occurrences. The unit, multiplication, and strength are induced from the lifted list monad and descend because the permutation equations act by symmetries of finite direct sums.

For every n, the additive commutative monoid $\langle \mathbb{R}^n, 0, +\rangle$ gives the handler

$$\mathrm{sum}_n : \mathsf{MFin}\mathbb{R}^n \to \mathbb{R}^n, \qquad \{\{v_1, \dots, v_k\}\} \mapsto \sum_{i=1}^{k} v_i.$$

Its lifted fibre map is characterised by the list quotient: after pulling back to an ordered representative, it sends an incoming cotangent $\xi \in \mathbb{R}^n$ to one copy of ξ at every occurrence. Under the support identification above, this is the invariant cotangent with component ξ at every support point. Hence, when two syntactic branches produce the same base value, the same support cotangent is copied back to both branches and the surrounding context addition accounts for the multiplicity.

The source operation choose_k is therefore interpreted by multiset union of the k branch computations. Its linear operation choose_k^w is the reverse fibre map of the descended union map: it restricts the incoming invariant multiset cotangent to the branch multisets, copying it along collisions. The handler clause for fold_ν, and in particular sum_n, is the generic algebra clause using the lifted commutative-monoid fold. These data satisfy the commutative-monoid equations in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, so the generic CHAD theorem applies.

H.3.2 Exception monads

Fix an exception object $\widehat{E} = \langle E, B \rangle$ in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$; for a purely discrete exception set one takes $B_e = 0$. The underlying set-level exception monad is

$$T_E X = X \sqcup E.$$

Its signature has one generic effect

$$\mathrm{raise} : E \to T_E 0,$$

equivalently one nullary operation for each point of E. Since the output type is 0, this is a discrete-output effect even when the parameter object $\widehat{E}$ has nonzero cotangent fibres. The raw AST monad is already

$$\mathsf{AST}_E X \simeq \mu Y.\, X \sqcup E \times Y^0 \simeq X \sqcup E,$$

so there is no non-trivial quotient to impose.

A T_E-algebra on a set Y is a map $a : Y \sqcup E \to Y$ whose restriction to Y is the identity. Equivalently it is a recovery map

$$h_0 : E \to Y, \qquad a = [\mathrm{id}_Y, h_0].$$

For instance, taking $Y = A \sqcup E'$ allows a handler to recover some exceptions as values in A and translate others into values tagged by a new exception set E'; it is still an algebra for the original monad T_E on the carrier Y.

The exception monad lifts by the coproduct formula

$$T_E^{\mathsf{F}} \langle X, A \rangle = \left\langle X \sqcup E, z \mapsto \begin{cases} A(x)\,, & z = \mathrm{inl}(x), \\ B(e)\,, & z = \mathrm{inr}(e). \end{cases} \right\rangle.$$

The lifted unit, multiplication, and strength are the coproduct maps induced by those of T_E. The lifted raise operation $\widehat{E} \to T_E^{\mathsf{F}} \langle X, A \rangle$ has reverse fibre map the identity

$$B(e) \longrightarrow B(e)\,,$$

because there are no continuation branches.

Let $\langle Y, C\rangle$ be a lifted carrier and let $\widehat{h} : \widehat{E} \to \langle Y, C\rangle$ be a morphism over the recovery map $h_0 : E \to Y$. The lifted algebra over $[\mathrm{id}_Y, h_0]$ is case analysis. Its reverse fibre map sends $\gamma \in C(y)$ over a returned value $\mathrm{inl}(y)$ to γ, while over an exception $\mathrm{inr}(e)$ it sends $\gamma \in C(h_0(e))$ to the fibre map of $\widehat{h}$,

$$\left(\left(\widehat{h}\right)_{\mathrm{fbr}}\right)^{*}_{e}(\gamma) \in B(e)\,.$$

When h_0 is an ordinary differentiable recovery map and $\widehat{h}$ is its cotangent lift, this fibre map is $(Dh_0(e))^{\mathsf{T}}\gamma$. The Eilenberg–Moore laws are immediate from $[\mathrm{id}_Y, h_0] \circ \eta = \mathrm{id}_Y$ and from the fact that multiplication for $X \sqcup E$ is case analysis that leaves the first encountered exception unchanged. Thus exceptions fit the admissible pattern directly; no quotient is needed for the direct coproduct lift. The free-algebra/product-continuation construction also recovers T_E because free exception algebras separate returned values and exception labels, but a single chosen recovery handler need not be separating if it identifies exceptions or misses some cotangent direction.

H.3.3 Writer monads and $\mathbb{R}^n$-accumulation

Let $M = \langle M, e, \cdot\rangle$ be a monoid. We use the right-action convention for the monad on $X \times M$: bind sends $\langle x, m\rangle$, together with $f : X \to Y \times M$, to $\langle y, n \cdot m\rangle$ when $f(x) = \langle y, n\rangle$. Equivalently, this is the usual writer monad for the opposite monoid; for the additive monoids $\mathbb{R}^n$ used below, the two conventions coincide. With this convention, an Eilenberg–Moore algebra for W_M is precisely a right M-action

$$a : A \times M \to A$$

with $a(a_0, e) = a_0$ and $a(a(a_0, m), n) = a(a_0, m \cdot n)$.

The generic effect is

$$\mathrm{tell} : M \to W_M 1, \qquad r \mapsto \langle *, r\rangle\,.$$

The induced Kleisli constructor sends a parameter $r \in M$ and a branch $k(*) = \langle x, s\rangle \in W_M X$ to $\langle x, s \cdot r\rangle$. Hence the operation-output type is 1, so the writer effect is in the discrete-output fragment even when the parameter object M is continuous.

For the accumulation example take $M = \langle \mathbb{R}^n, 0, +\rangle$ with its standard cotangent bundle. The lifted writer monad maps

$$\langle X, A\rangle \mapsto \langle X \times \mathbb{R}^n, (\langle x, r\rangle \mapsto A(x) \oplus \mathbb{R}^n)\rangle\,.$$

For a morphism $\langle f, \alpha\rangle : \langle X, A\rangle \to \langle Y, B\rangle$, the base map is $\langle x, r\rangle \mapsto \langle f(x), r\rangle$, and the reverse fibre map is

$$B(f(x)) \oplus \mathbb{R}^n \longrightarrow A(x) \oplus \mathbb{R}^n, \qquad \langle \beta, \rho\rangle \mapsto \langle (\alpha)_x(\beta), \rho\rangle\,.$$

The lifted multiplication is the transpose derivative of

$$\langle \langle x, s\rangle, r\rangle \mapsto \langle x, s + r\rangle\,,$$

so an incoming cotangent $\langle x^*, u^*\rangle$ is sent to $\langle \langle x^*, u^*\rangle, u^*\rangle$. This proves the lifted monad laws because the displayed maps are the transpose derivatives of the unit, associative addition, and strength maps.

For the lifted tell operation, if the branch computation has base $\langle x, s\rangle$, then the operation has base $\langle x, s + r\rangle$. Its reverse fibre map sends an incoming cotangent $\langle x^*, u^*\rangle$ to the parameter cotangent u^* and the branch cotangent $\langle x^*, u^* \rangle$. Thus the newly written value and the writer value produced by the continuation both receive the same accumulator cotangent.

For the commutative additive accumulation example, the useful carrier is $\sigma \times \mathbb{R}^n$ with action

$$(\sigma \times \mathbb{R}^n) \times \mathbb{R}^n \to \sigma \times \mathbb{R}^n, \qquad \langle \langle a, u \rangle, r \rangle \mapsto \langle a, u + r \rangle. \tag{114}$$

If σ is lifted with fibre A_a, the transposed derivative of Equation (114) sends a cotangent $\langle a^*, u^* \rangle$ at $\langle a, u + r \rangle$ to

$$\langle \langle a^*, u^* \rangle, u^* \rangle \in (A_a \oplus \mathbb{R}^n) \oplus \mathbb{R}^n,$$

where the two copies of u^* correspond to the old accumulated value and the newly written value. The action laws are exactly the Eilenberg–Moore algebra laws for the writer monad, so this gives the lifted handler used by run_τ and its linear operation $\mathrm{run}^w_{\tau,p}$. This primitive handles computations whose returned values already lie in $\tau \times \mathsf{real}^n$; the usual runWriter : $\mathrm{M}\,\tau \to \tau \times \mathsf{real}^n$ is obtained by first mapping $a : \tau$ to $\langle a, 0 \rangle$ and then applying this algebra. For a general differentiable monoid object, the same statements hold with addition replaced by the transpose derivative of the chosen multiplication-order map.

H.4 Scope and non-discrete output limitations

The constructions above are deliberately discrete-output constructions. They allow differentiable operation-input objects, such as continuous exception labels or continuous writer values, because these are parameters I_ω and their cotangents are accounted for at the operation node. They also allow arbitrary zero-fibre output indices ΔO_ω, provided the required set-indexed products and direct sums exist; in the finitary CHAD theorem these output sets are finite.

They do not cover genuine operation-output objects with nonzero cotangent fibres. For such an object $\widehat{O}_\omega$, a continuation is a morphism $\widehat{O}_\omega \to B$, not just a function $P\widehat{O}_\omega \to PB$, so its base point can contain reverse-linear fibre data. If syntax or handlers can inspect that data, the base of the lifted construction need not be the intended set-level monad. The obstruction remains even if genuine powers $B^{\widehat{O}_\omega}$ are supplied.

Lemma H.2 (Genuine output fibres can destroy fibredness)**.** *There are a one-operation raw theory, an output object with nonzero fibre $\widehat{O} \in \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, and objects $A, \widehat{R} \in \mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ for which the following holds. If operation nodes are indexed by genuine continuations into the recursively generated syntax object, then the resulting direct raw-syntax construction is not fibred over the set-level raw monad. Moreover, assuming a genuine power $\widehat{R}^{\widehat{O}}$ whose base points are morphisms $\widehat{O} \to \widehat{R}$, there is a $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$-algebra for the raw operation which does not lie over any set-level unary-operation algebra, and whose induced image and coimage have bases that cannot be obtained from any quotient of the set-level unary-operation monad.*

Proof. Work one-sorted and put

$$\widehat{O} = (\{*\}, \mathbb{R}), \qquad A = (\{*\}, \mathbb{R}).$$

Consider the raw theory with one operation ω, no parameters, output object $\widehat{O}$, and no equations. Its set-level shadow is the free unary-operation theory, since $P\widehat{O} = \{*\}$. Hence the set-level raw monad is

$$SX \cong \mathbb{N} \times X,$$

and, over the one-point set $PA = \{*\}$, there is exactly one depth-one set-level term.

In $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, a morphism $k : \widehat{O} \to A$ consists of the unique base map and a reverse fibre map

$$A_* = \mathbb{R} \longrightarrow \widehat{O}_* = \mathbb{R}.$$

Thus such variable continuations are parametrised by scalars $\lambda \in \mathbb{R}$; write $k_\lambda : \widehat{O} \to A$ for the morphism whose reverse fibre map is multiplication by λ. Let $\widehat{S}$ denote the direct raw construction obtained by replacing the discrete continuation object $B^{\Delta(P\widehat{O})}$ by the genuine power $B^{\widehat{O}}$. The operation constructor in $\widehat{S}A$ is therefore indexed by morphisms $\widehat{O} \to \widehat{S}A$. Composing the k_λ's with the unit $\eta_A : A \to \widehat{S}A$ gives distinct morphisms $\widehat{O} \to \widehat{S}A$, because the variable summand carries the original fibre and η_A has the identity reverse fibre map there. Hence we obtain distinct depth-one base terms

$$t_\lambda = \omega(\eta_A \circ k_\lambda), \qquad \lambda \in \mathbb{R},$$

because the genuine continuation morphism, including its reverse fibre map, is part of the operation-node data and there are no equations identifying different k_λ's. Consequently $P\widehat{S}A$ has continuum many depth-one operation nodes, and is in particular uncountable, whereas $S(PA) \cong \mathbb{N}$ is countable. Thus the direct raw construction cannot be fibred over the set-level raw monad.

The same issue appears for image and coimage semantics once handlers are allowed to observe the fibre component of a genuine continuation. Assume a genuine power object $\widehat{R}^{\widehat{O}}$ is available, with base points the morphisms $\widehat{O} \to \widehat{R}$. Let

$$\widehat{R} = (\mathbb{R}, \mathbb{R})$$

be the constant one-dimensional family over the set $\mathbb{R}$. A base point of $\widehat{R}^{\widehat{O}}$ is then a morphism $\widehat{O} \to \widehat{R}$, equivalently a pair

$$(r, \ell), \qquad r \in \mathbb{R}, \quad \ell : \mathbb{R} \to \mathbb{R} \text{ linear},$$

where r is the base value and ℓ is the reverse fibre map. Define a lifted algebra map

$$h : \widehat{R}^{\widehat{O}} \to \widehat{R}$$

for ω by the base function

$$(r, \ell) \longmapsto \ell(1),$$

and choose the zero reverse fibre map at every base point. This is a morphism in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, since every fibre is a vector space. Its base map depends on the reverse fibre map ℓ, so it does not factor through the projection to the set-level branch value r; hence it is not a lift of any set-level unary-operation algebra. Because the raw theory imposes no equations, this operation structure nevertheless extends uniquely to an algebra for the free raw-syntax monad in $\mathsf{Fam}(\mathbf{Vect}^{\mathrm{op}})$.

For each $\mu \in \mathbb{R}$, let $\rho_\mu : A \to \widehat{R}$ be the valuation with base value 0 and reverse fibre map multiplication by μ. The composite

$$\widehat{O} \xrightarrow{k_\lambda} A \xrightarrow{\rho_\mu} \widehat{R}$$

has base value 0 and reverse fibre map multiplication by $\lambda\mu$, where $\lambda\mu$ denotes the multiplication map by that scalar. Therefore the base denotation of the depth-one term $t_\lambda = \omega(\eta_A \circ k_\lambda)$ under the valuation ρ_μ is

$$(Ph)(0, \lambda\mu) = \lambda\mu.$$

Taking $\mu = 1$ separates all λ. Hence image and coimage contain continuum many depth-one denotations, respectively kernel-pair classes. They cannot be bases of lifts of any quotient of the set-level unary-operation semantics over PA, since every quotient of $S(PA) \cong \mathbb{N}$ is countable and cannot create additional depth-one base points.

Thus nonzero output fibres can make continuation-fibre data visible on bases, exactly what zero-fibre output indices prevent. Genuine nonzero-fibre outputs therefore need extra structure or equations forcing syntax and handlers to quotient or ignore that data. $\square$