# Unraveling the iterative CHAD

FERNANDO LUCATELLI NUNES, Utrecht University, Netherlands
GORDON PLOTKIN, Google DeepMind, USA
MATTHIJS VÁKÁR, Utrecht University, Netherlands

Combinatory Homomorphic Automatic Differentiation (CHAD) was originally formulated as a semantics-driven source-to-source transformation for reverse-mode AD in total (terminating) programming languages. In this work, we extend CHAD to encompass partial languages featuring constructs such as partial (potentially non-terminating) operations, data-dependent conditionals (e.g., real-valued tests), and iteration constructs (e.g., while-loops), while maintaining CHAD's core principle of structure-preserving semantics.

A central contribution is the introduction of iteration-extensive indexed categories, which provide a principled integration of iteration into the target language's op-Grothendieck construction. This is achieved by requiring that iteration in the base category lifts to parameterized initial algebras in the indexed category, yielding a fibred iterative structure that elegantly models while-loops and other iteration constructs in the total category.

By applying the principle inspired by iteration-extensive indexed categories, we extend the CHAD transformation as the unique structure-preserving functor – specifically, an iterative Freyd category morphism – from the freely generated iterative distributive Freyd categorical structure on the source language to the syntactic semantics of the (suitable op-Grothendieck construction of the) target language, with each primitive operation mapped to its derivative.

We establish the correctness of this extended transformation via the universal property of the syntactic categorical semantics of the source language, showing that the differentiated programs compute correct reverse-mode derivatives of their originals.

In summary, our work advances the study of iteration fixpoint operators within the setting of indexed categories. Our results thus contribute to the understanding of iteration constructs in dependently typed languages and its relation with categories of containers. As our primary motivation and application, we generalize CHAD to languages with data types, partial features, and iteration, providing the first rigorous categorical semantics for reverse-mode CHAD in such settings and formally guaranteeing the correctness of the source-to-source CHAD technique.

Additional Key Words and Phrases: Automatic Differentiation, Denotational Semantics, Indexed Categories, Dependent Types, Iteration, Grothendieck Construction, Category of Containers, Programming Languages

**ACM Reference Format:**
Fernando Lucatelli Nunes, Gordon Plotkin, and Matthijs Vákár. 2022. Unraveling the iterative CHAD. 1, 1 (May 2022), 57 pages.

**MSC 2020:** 18C10, 18C15, 18C20, 18D05, 03B70, 03F52, 68Q55, 68N18, 68T07
**ACM CCS 2012:** F.3.2; F.3.3; D.3.1; D.3.2; D.2.4; G.4; I.2.3; I.2.6; G.1.10

Authors' addresses: Fernando Lucatelli Nunes, Department of Information and Computing Sciences, Utrecht University, Netherlands, f.lucatellinunes@uu.nl; Gordon Plotkin, Google DeepMind, Google DeepMind, USA, gdp@inf.ed.ac.uk; Matthijs Vákár, Department of Information and Computing Sciences, Utrecht University, Netherlands, m.i.l.vakar@uu.nl.


XXXX-XXXX/2022/5-ART $15.00
https://doi.org/

# Contents

## INTRODUCTION

*Our motivations in a nutshell.* There are two fundamentally different questions that motivate this work:

(1) how to compute derivatives of functions that are implemented by code that uses iteration (while-loops); we address this question by extending the Combinatory Homomorphic Automatic Differentiation (CHAD) framework to account for programs with iteration in the sense of least (pre)fixed points $\mathtt{iterate}\, f : A \to B$ of morphisms $f : A \to B \sqcup A$; in previous work, the CHAD derivatives of a language construct can be calculated by constructing an interpretation of the construct in suitable categories of containers $\Sigma_C \mathcal{L}^{op}$ for an indexed category $\mathcal{L} : C^{op} \to \mathsf{Cat}$; in this sense, the question is under what conditions and how container categories $\Sigma_C \mathcal{L}^{op}$ interpret iteration;

(2) how to compute inductive types in a dependent type theory; inductive types are usually treated as (parameterized) initial algebras of certain endofunctors; in addition to the usual endofunctors contructable in a simple type theory, such as the ones arising from products $\times$ and coproducts $\sqcup$, dependent types add extra endofunctors $\mathcal{L}(f)$ arising from substitutions; in this picture, we have a category $C$ that models closed types and their programs and an indexed category $\mathcal{L} : C^{op} \to \mathsf{Cat}$ to model dependent types and their programs; this raises the question what the initial algebras $\mu\mathcal{L}(f) \in \mathcal{L}(A)$ of the functors $\mathcal{L}(f) : \mathcal{L}(A) \to \mathcal{L}(A)$ for morphisms $f : A \to A$ in $C$ are; such initial algebras turn out to be trivial in most cases; however, the more general parameterized initial algebras $\mu\mathcal{L}(f) : \mathcal{L}(B) \to \mathcal{L}(A)$ of the functors $\mathcal{L}(B) \times \mathcal{L}(A) \cong \mathcal{L}(B \sqcup A) \to \mathcal{L}(A)$ for morphisms $f : A \to B \sqcup A$ in $C$, in case $\mathcal{L}$ is extensive, turn out to be much more interesting.

In fact, the answers to both questions turn to be closely related in the sense that it is often the case that $C$ supports iteration and parameterized initial algebras $\mu\mathcal{L}(f)$ above exists and equal $\mathcal{L}(\mathtt{iterate}\, f)$. Further, in those cases, we can prove that $\Sigma_C \mathcal{L}^{op}$ supports iteration. We use this theoretical development to give an account of CHAD derivatives of programs with while-loops.

*About motivation (1): CHAD.* Reverse mode Automatic Differentiation (AD) is a fundamental technique harnessing the chain rule to compute derivatives of functions implemented by programs. It plays an integral role in addressing the challenges of efficiently computing derivatives of functions with high-dimensional domains while maintaining numerical stability. Its widespread application in machine learning and scientific computing leaves no doubt about its practical significance.

Within the programming languages (PL) community, the pursuit of a simple, easily implementable, and purely functional reverse AD that can be applied to a wide range of programs at *compile time* through *source-code transformation* has been a central objective. The realization of this goal hinges on the development of an *AD source-code transformation* method underpinned by a well-defined denotational semantics.

Reverse-mode Combinatory Homomorphic Automatic Differentiation (CHAD) [30, 48, 51, 54], a paradigm that can be seen as a formalization and extension of Elliott's simple essence of automatic differentiation [12], implements reverse AD following a principled denotational semantics; namely, we establish a setting where the code transformation can be realized as a uniquely defined structure-preserving functor on the syntax of the source language.

Following this principle, CHAD provides us with a reverse-mode AD whose categorical semantics is the dual of the corresponding forward mode, allowing for a correctness proof using logical relations (LR). The program transformation CHAD has been shown to apply to expressive total languages, involving sum, function and (co)inductive types. The key ideas is to make use of a (target) language with linear dependent types [30] and to observe that its category of containers

$\Sigma_C \mathcal{L}^{op}$ (see, e.g. [2], or the extensive literature on functional programming with lenses) then has enough structure to interpret the source language. The result is that CHAD associates with each source program type a container type $(C, L) \in \Sigma_C \mathcal{L}^{op}$ where $C \in \mathcal{C}$ and $L \in \mathcal{L}(C)$ and that it computes derivatives by storing the primal values in $C$ and accumulating (in reverse) the derivative values into $L$.

A remaining challenge is to apply CHAD to languages with partial features. More precisely, we are particularly interested in understanding CHAD for languages with iteration constructs such as `while`-loops, as they are common features exploited in machine learning and scientific computing. For example, loops are commonly used to implement Taylor approximations to special functions, iterative solvers, and various optimization algorithms. We formalize such iteration via a language construct that computes the least (pre)fixed point $\texttt{iterate}\, f : A \to B$ of a function $f : A \to B \sqcup A$ into a sum type.

If we are to realize the vision of general differential programming languages where all programs can be differentiated, it is essential to account for the interaction of differentiation with iteration. However, incorporating iteration into CHAD presents significant challenges. Iteration introduces partiality into the language, as loops may not terminate, which requires an extension of the concrete denotational semantics and the correctness proofs of the AD transformations, as well as a formalisation of what structure preservation means in this instance. The key point is to characterize precisely when and how categories of containers $\Sigma_C \mathcal{L}^{op}$ interpret iteration. The result is a novel approach that integrates iteration into the CHAD framework while maintaining its functional, compile-time source-code transformation properties.

*About motivation (2): inductive types in dependent type theories.* In a categorical model $\mathcal{C}$ of a simple type theory, inductive types are typically interpreted as least (pre)fixed points $\mu F \in \mathcal{C}$ of certain endofunctors $F : \mathcal{C} \to \mathcal{C}$, in this context also known as *initial algebras*, and more generally as *parameterized initial algebras* $\mu F : \mathcal{C}' \to \mathcal{C}$ when $F : \mathcal{C}' \times \mathcal{C} \to \mathcal{C}$ [17]. These endofunctors typically are limited to the ones that are constructable in the type theory: ones built out of primitive types, products $\times$, coproducts $\sqcup$, and sometimes exponentials $\to$ [45].

In dependent type theory, substitutions of terms in types provide a new source of endofunctors. Such dependent type theories are typically modelled by indexed categories $\mathcal{L} : \mathcal{C}^{op} \to \mathsf{Cat}$, where we think of the fibre categories $\mathcal{L}(C)$ as the categories dependent types and terms in a particular context $C$ and we think of the change-of-base functors $\mathcal{L}(f) : \mathcal{L}(C) \to \mathcal{L}(D)$ for $f : D \to C$ as the modelling substitutions. In fact, given that dependent type theories with sum types tend to be modelled by indexed categories $\mathcal{L}$ that are extensive in the sense that $\mathcal{L}(C \sqcup E) \cong \mathcal{L}(C) \times \mathcal{L}(E)$, we also obtain substitution functors $\mathcal{L}(f) : \mathcal{L}(C) \times \mathcal{L}(E) \to \mathcal{L}(D)$ for $f : D \to C \sqcup E$. The question rises what the parameterized initial algebras $\mu\mathcal{L}(f) : \mathcal{L}(B) \to \mathcal{L}(A)$ of such change-of-base functors $\mathcal{L}(f) : \mathcal{L}(B) \times \mathcal{L}(A) \to \mathcal{L}(A)$ look like. We show that for interesting practical examples, such as families $\mathcal{L} = \mathcal{C}^{(-)} : \mathbf{Set}^{op} \to \mathsf{Cat}; S \mapsto \mathcal{C}^S$ valued in some category $\mathcal{C}$, the indexed categories $\mathcal{L}$ preserve least (pre)fixed points in the sense that $\mu\mathcal{L}(f) = \mathcal{L}(\texttt{iterate}\, f)$. Further, we demonstrate that $\Sigma_C \mathcal{L}^{op}$ then interprets iteration, giving us a method for differentiating iterative programs with CHAD and giving a denotational proof of correctness of the resulting technique.

*Contributions.* Summarizing, in this paper, we make the following contributions:

- we revisit the notions of extensive indexed categories introduced in [30, 31] and motivated by the original notion of extensive categories, *e.g.* [10]. We extend our contributions in this subject, especially showing that finite-coproduct-extensive indexed categories gives us $\Sigma$-bimodels for sums of [30] (see Lemma 4.9);

- given an indexed category $\mathcal{L} : C^{op} \to \mathsf{Cat}$, we observe that initial algebras (resp. terminal coalgebras) of $\mathcal{L}(f) : \mathcal{L}(A) \to \mathcal{L}(A)$ for $f : A \to A$ in $C$; are trivial in case $\mathcal{L}$ has indexed initial (resp. terminal) objects; next, we turn to the related question of parameterized initial algebras of $\mathcal{L}(f) : \mathcal{L}(B) \times \mathcal{L}(A) \cong \mathcal{L}(B \sqcup A) \to \mathcal{L}(A)$ of $f : A \to B \sqcup A$ in case $\mathcal{L}$ is extensive; these turn out to be much more interesting;
- we study how one can incorporate iteration in a dependently typed language in a manner that is compatible with its category of containers. We do that by introducing various levels of structured notions of indexed categories that induce iteration constructs in its (op-)Grothendieck construction;
- we introduce, in Subsect. 4.6, the notion of iteration-extensive indexed category $\mathcal{L} : C^{op} \to \mathsf{Cat}$: this roughly means that $\mathcal{L}$ sends least fixed points to least fixed points in the sense that $B' \mapsto \mathcal{L}(\mathtt{iterate}\ f)(B')$ is the parameterized initial algebra of $\mathcal{L}(f) : \mathcal{L}(B) \times \mathcal{L}(A) \cong \mathcal{L}(B \sqcup A) \to \mathcal{L}(A)$ where $f : A \to B \sqcup A$ induces $\mathtt{iterate}\ f : A \to B$);
- the iteration-extensive indexed category is the basic principle we follow on the study of iteration in dependently typed languages (indexed categories) – we show that every iteration-extensive indexed category induces an iteration that has a universal property on its (op-)Grothendieck construction in the sense of Theorem 4.20;
- for any category $C$ with initial object, we show that $C^{(-)} : \mathsf{PSet}^{op} \to \mathsf{Cat}$ is extensive for iteration, meaning that the Kleisli category of the maybe-monad on $\mathbf{Fam}\,(C^{op}) = \Sigma_{X \in \mathsf{Set}}(C^{op})^X$ interprets iteration; we characterize how it behaves. We also give some other natural examples;
- we introduce a more syntactic (and less structured) version of indexed categories that induce iteration constructs in its (op-)Grothendieck construction; namely, we introduce the concept of iterative indexed category, which consists of indexed categories with the iterative folders, as established in Subsect. 4.9. We show that our notion of iterative folderson a indexed category is in bijection with possible (fibred coarse) iterations in the categories of containers (Lemma 4.44);
- we introduce a more structured (concrete) concept of indexed category that induces iteration on its (op-)Grothendieck construction; namely the finite-$\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproduct-extensive indexed categories introduced in Subsect. 4.8;
- for each notion of iterative indexed category we introduced, we introduced a corresponding notion of iterative Freyd indexed category – the corresponding op-Grothendieck constructions give iterative Freyd categories with iteration – that is to say, we can interpret a call-by-value language with tuples, cotuples and iteration;
- by observing that the case of the category of vector spaces $\mathbf{Vect}_{\perp}^{(-)} : \mathsf{PSet}^{op} \to \mathsf{Cat}$ lets us give a denotational semantics for reverse-mode derivatives in the category of containers $\mathfrak{P}\mathbf{Fam}\,(\mathbf{Vect}^{op})$, we use our theoretical development to derive a reverse CHAD code transformation and correctness proof for programs with iteration.

*Terminology and conventions.* Although much of the work presented in this paper can be extended to more general settings, we concentrate on strictly indexed categories to align with our primary objectives. Thus, unless stated otherwise, by an *indexed category* $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$, we specifically mean a *strictly indexed category*, that is, a strict functor $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$. The same convention applies to variations on the notion of indexed categories. For instance, when we refer to (coproduct-)extensive categories, we mean functors that preserve binary products in $C^{\mathrm{op}}$, as established in [30, Lemma 32].

This restriction is not a limitation but a matter of exposition: it reflects our preferred mode of presentation. Even for our novel contributions, we have verified that the necessary adjustments to encompass the more general setting of (non-strict) indexed categories are straightforward.

Similarly, as our main focus lies in the development of reverse CHAD, we have emphasized constructions that naturally support op-Grothendieck constructions. Although the dualization to the case of Grothendieck construction (forward mode) – and of iteration-extensive categories – is of independent theoretical and practical interest, it largely reduces to an exercise in category theory. We intend to address and explore these dual notions in future and ongoing work, particularly in [27].

## 1 TOTAL CHAD: CONCRETE SEMANTICS

Reverse mode Combinatory Homomorphic Automatic Differentiation (CHAD) is an automatic differentiation macro realized as a program transformation. As such, it is executed at compile-time as a source-code transformation between a *source language* and a *target language*. We direct the reader to [30] for a comprehensive understanding of CHAD within the setting of total languages and to [48] for its efficient implementation.

Herein, we commence by revisiting the fundamental principle of the (concrete) denotational semantics of CHAD for total languages. Specifically, we draw attention to the detailed treatment of the semantics of the target language and differentiable functions within the context of CHAD for total functions, as explicated in [30, Section 6 and 10.4].

### 1.1 Grothendieck constructions and the free comproduct completion

Given the reliance of our target language's concrete denotational semantics on (op-)Grothendieck constructions, we direct the reader to [30, Section 6] and [31] for a detailed discussion within our context.

Let $\mathcal{A} : C^{\mathrm{op}} \to \mathsf{Cat}$ be a (strictly) indexed category. We denote by $\Sigma_C \mathcal{A}$ the Grothendieck construction of $\mathcal{A}$. More importantly for our setting, we denote by $\Sigma_C \mathcal{A}^{\mathrm{op}}$ the op-Grothendieck construction, that is to say, the Grothendieck construction of the (strictly) indexed category $\mathcal{A}^{\mathrm{op}}$ defined by $\mathcal{A}^{\mathrm{op}}(f) = (\mathcal{A}(f))^{\mathrm{op}}$, which can be seen as the composition $\mathrm{op} \circ \mathcal{A} : C^{\mathrm{op}} \to \mathsf{Cat}$. More explicitly, the objects of $\Sigma_C \mathcal{A}^{\mathrm{op}}$ are pairs $(M \in \mathrm{obj}\,(C), X \in \mathcal{A}\,(M))$, and a morphism $(M, X) \to (N, Y)$ consists of a pair

$$(f : M \to N, f' : \mathcal{A}(f)(Y) \to X) \in C\,(M, N) \times \mathcal{A}\,(M)\,(\mathcal{A}(f)(Y), X). \tag{1.1}$$

Recall that, if $(f, f') : (M, X) \to (N, Y)$ and $(g, g') : (N, Y) \to (K, Z)$ are morphisms of $\Sigma_C \mathcal{A}^{\mathrm{op}}$, the composition in $\Sigma_C \mathcal{A}^{\mathrm{op}}$ is given by

$$(g, g') \circ (f, f') = (g \circ f, f' \circ \mathcal{A}\,(f)\,(g'))\,. \tag{1.2}$$

Furthermore, as many of the Grothendieck constructions we encounter can be understood as free coproduct completions, we recall that the free coproduct completion $\mathbf{Fam}\,(C^{\mathrm{op}})$ of the opposite of the category $C$ is the op-Grothendieck construction of the indexed category

$$C^{(-)} : \mathsf{Set}^{\mathrm{op}} \to \mathsf{Cat}; \qquad A \mapsto C^A, \qquad (f : A \to B) \mapsto \left(C^f : C^B \to C^A\right), \tag{1.3}$$

where, for each set $A$ treated as a discrete category, we denote by $C^A$ the category of functors from $A$ to the category $C$, and we denote by $C^f$ the functor $C^B \to C^A$ defined by $X \mapsto X \circ f$. For the interested reader, we suggest consulting [30, Section 9.2], [40, Section 1], [4], [28, Section 8.5], and [44, Section 1.2] for fundamental definitions and further exploration.

Explicitly, objects of $\mathbf{Fam}\,(C^{\mathrm{op}})$ take the form of pairs $(A, X)$, where $A$ is a set and $X$ denotes a functor $X : A \to C$, treating $A$ as a discrete category; *i.e.* $A$ is a set and $X$ is an $A$-indexed family of objects in $C$. A morphism $\mathbf{f} : (A, X) \to (B, Y)$ in $\mathbf{Fam}(C)$ is a pair

$$\mathbf{f} = (f, f') \in \mathsf{Set}(A, B) \times \mathsf{NAT}(X, Y \circ f) \tag{1.4}$$

where $f : A \to B$ is a function and $f'$ is a family

$$f' = \left(f'_a : Y(f(a)) \to X(a)\right)_{a \in A} \tag{1.5}$$

of morphisms in $C$. Finally, recall that, if $\mathbf{f} : (A, X) \to (B, Y)$ and $\mathbf{g} : (B, Y) \to (C, Z)$ are morphisms in $\mathbf{Fam}(C^{\mathrm{op}})$,

$$\mathbf{g} \circ \mathbf{f} = (g, g') \circ (f, f') = (g \circ f, (gf)'), \tag{1.6}$$

where $(gf)'_a = f'_a \circ g'_{f(a)} : Z(g(f(a))) \to X(a)$.

### 1.2 Differentiable functions

*We consider the category $\mathcal{VM}an$ of differentiable manifolds of variable dimension, called herein manifolds for short, and differentiable functions between them.* The category $\mathcal{VM}an$ can be realized as the free coproduct completion $\mathbf{Fam}(\mathcal{M}an)$ of the usual category $\mathcal{M}an$ of connected differentiable manifolds of finite dimension.

In practice, in order to provide a concrete semantics for CHAD, we only need Euclidean spaces and (free) coproducts (copairing) of Euclidean spaces. Hence, the reader is free to think of $\mathcal{VM}an$ as the free coproduct completion $\mathbf{Fam}(\mathfrak{E})$ of the category $\mathfrak{E}$ consisting of the euclidean spaces $\mathbb{R}^n$ and differentiable functions between them.

Recall that, if $f : M \to N$ is a morphism in $\mathcal{VM}an$, we have a notion of derivative of $f$, *e.g.* [22] for the classical case. More precisely, the derivative of $f$ at $w \in M$ is given by a linear transformation

$$f'(w) : \mathcal{T}_M w \to \mathcal{T}_N f(w) \tag{1.7}$$

where $\mathcal{T}_M w \cong \mathbb{R}^n$ and $\mathcal{T}_N f(w) \cong \mathbb{R}^m$ are the tangent spaces of $M$ at $w$ and of $N$ at $f(w)$, respectively. The *coderivative* (or the reverse-mode derivative) of $f$, denoted simply as $\partial_w(f)$ herein, is the transpose (or dual), denoting herein by $\partial_w(f)$, of the linear transformation $f'(w)$; namely:

$$\partial_w(f) : \mathcal{T}^*_N f(w) \to \mathcal{T}^*_M w \tag{1.8}$$

is defined between the linear duals[1] of the respective tangent spaces, called *cotangent spaces* $\mathcal{T}^*_N f(w)$ and $\mathcal{T}^*_M w$, by precomposing $f'(w)$. Observe that given our identifications $\mathcal{T}_M w \cong \mathbb{R}^n$ and $\mathcal{T}_N f(w) \cong \mathbb{R}^m$, $f'(w)$ has a representation as an $m \times n$-matrix and $\partial_w(f)$ is then represented by its matrix transpose. We denote by $\partial^*(f)$ the family of coderivatives, defined as

$$\partial^*(f) := \left(\partial_w(f) : \mathcal{T}^*_N f(w) \to \mathcal{T}^*_M w\right)_{w \in M}. \tag{1.9}$$

With this notation, the basic tenet of CHAD is to pair up the primal $f$ with $\partial^*(f)$; namely we denote

$$\mathfrak{D}f = (f_0, \partial^*(f)), \tag{1.10}$$

where $f_0$ is the underlying function of $f$. The pair $(f_0, \partial^*(f))$, which we denote by $(f, \partial^*(f))$ by abuse of language, is called herein *CHAD-derivative*: it is realized as a morphism in the category $\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ which plays the role of the concrete denotational semantics of the target language of CHAD, as explained below.

[1] Recall that the linear dual $V^*$ of an $\mathbb{R}$-vector space is defined as the vector space $V \multimap \mathbb{R}$ of linear transformations from $V$ to $\mathbb{R}$.

### 1.3 CHAD-derivatives as morphisms

We start by recalling some basic aspects of the concrete model for the target language. Our focus lies on the category $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$, which represents the free coproduct completion of the opposite of the category of vector spaces (and linear transformations).

Recall that $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ emerges as the *op-Grothendieck construction* of the indexed category

$$\mathbf{Vect}^{(-)} : \mathsf{Set}^{\mathrm{op}} \to \mathsf{Cat}; \qquad A \mapsto \mathbf{Vect}^{A}, \qquad (f : A \to B) \mapsto \left(\mathbf{Vect}^{f} : \mathbf{Vect}^{B} \to \mathbf{Vect}^{A}\right). \quad (1.11)$$

Explicitly, objects of $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ are pairs $(A, X)$ where $A$ is a set and $X$ is a functor $X : A \to \mathbf{Vect}$. A morphism $\mathbf{f} : (A, X) \to (B, Y)$ in $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ is a pair $\mathbf{f} = (f, f')$ where $f : A \to B$ is a function and $f'$ is a family of linear transformations

$$f' = \left(f'_a : Y\left(f(a)\right) \to X(a)\right)_{a \in A}. \quad (1.12)$$

We observe that, for each object $M$ of $\mathcal{VM}an$, we can define an object $\left(M, \mathcal{T}^*_M\right)$ where $M$ is the underlying set of $M$ and $\mathcal{T}^*_M$ is the functor ($M$-indexed family)

$$\mathcal{T}^*_M : M \to \mathbf{Vect}, \qquad w \mapsto \mathcal{T}^*_M w \quad (1.13)$$

of the cotangents on $M$. Moreover, by the above, it is clear that, whenever $f : M \to N$ is a morphism in $\mathcal{VM}an$, the CHAD-derivative $\mathfrak{D}f = (f, \partial^*(f))$ is a morphism

$$\mathfrak{D}f = (f, \partial^*(f)) : (f, \partial^*(f)) : \left(M, \mathcal{T}^*_M\right) \to \left(N, \mathcal{T}^*_N\right) \quad (1.14)$$

between the objects $\left(M, \mathcal{T}^*_M\right)$ and $\left(N, \mathcal{T}^*_N\right)$ in $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$, where, by abuse of notation, the first coordinate of $(f, \partial^*(f))$ is the underlying function of $f$.

### 1.4 CHAD-derivative as a structure-preserving functor

The basic principle of CHAD is to exploit structure-preserving functors and, more than that, the unique ones arising from the universal properties of the languages (or, more precisely, from the freely generated categorical structures over the languages). The reason that we can follow this approach is that the CHAD-derivative happens to be a functor that preserves most of the present categorical structure. Once we have demonstrated that the CHAD-derivative preserves the relevant categorical structure, we can implement it in code and derive a simple correctness proof from the fact that denotational semantics (and logical relations) also give structure preserving functors.

More precisely, within the context of simply-typed languages with variant and product types, Theorem 1.1 is pivotal to proceed with our denotational correctness proof.

Recall that if $(A, X)$ and $(B, Y)$ are objects of $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$, then $(A \sqcup B, [X, Y])$ is the coproduct and $\left(A \times B, X \underline{\times} Y\right)$, where $X \underline{\times} Y(w, w') = X(w) \times Y(w')$, is the product (see, for instance, [30, Section 10.2] and [31]). We can, then, verify that the CHAD derivative preserves products and coproducts:

Theorem 1.1. *The association*

$$\mathfrak{D} : \mathcal{VM}an \to \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \qquad M \mapsto \left(M, \mathcal{T}^*_M\right), \qquad f \mapsto (f, \partial^*(f)) \quad (1.15)$$

*defines a (strictly) bicartesian functor.*

Proof. The *chain rule* for derivatives implies that $\mathfrak{D}$ is indeed a functor. The preservation of products and coproducts follow from basic properties of derivatives, namely that derivative of a tupled function is the tupling of the derivatives, and that derivatives only depend on local information. □

*Definition 1.2 (CHAD-derivative).* Herein, the functor $\mathfrak{D}$ established in Theorem 1.1 is called the *(total) CHAD-derivative functor.*

## 2 DENOTATIONAL CORRECTNESS OF CHAD FOR TOTAL LANGUAGES

We can, now, briefly recall the correctness result of CHAD for total languages with variant and product types. CHAD is realized as structure-preserving program transformation between the source language, a simply-typed language, and the op-Grothendieck construction of the target language, which is a dependently typed language.

We start by recalling our source language and target language, and their respective categorical semantics.

### 2.1 Source total language

We consider a standard total functional programming language with variant and product types. This is constructed over ground types $\mathbf{real}^n$ for arrays of real numbers with static length $n$ (for $n \in \mathbb{N}$), and sets of primitive operations $\mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$ for each $(k_1, k_2) \in \mathbb{N} \times \mathbb{N}$ and each $(\mathbf{n}, \mathbf{m}) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2}$. For convenience, we denote by

$$\mathrm{Op} := \bigcup_{(k_1,k_2) \in \mathbb{N}\times\mathbb{N}} \; \bigcup_{(\mathbf{n},\mathbf{m}) \in \mathbb{N}^{k_1}\times\mathbb{N}^{k_2}} \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$$

the set of all primitive operations. If $\mathbf{n} = (n_1, \ldots, n_{k_1})$ and $\mathbf{m} = (m_1, \ldots, m_{k_2})$, a primitive operation op in $\mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$ intends to implement a differentiable function

$$[\![\mathrm{op}]\!] : \mathbb{R}^{\mathbf{n}} \to \mathbb{R}_{\mathbf{m}} \tag{2.1}$$

where we denote $\mathbb{R}^{\mathbf{n}} := \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_{k_1}}$ and $\mathbb{R}_{\mathbf{m}} := \mathbb{R}^{m_1} \sqcup \cdots \sqcup \mathbb{R}^{m_{k_2}}$. The reader can keep the following examples of primitive operations in mind:

- constants $\underline{c} \in \mathrm{Op}^0_n$ for each $c \in \mathbb{R}^n$, for which $[\![\underline{c}]\!] = \underline{c}$;
- elementwise addition and product $(+), (*) \in \mathrm{Op}^{(n,n)}_n$ and matrix-vector product $(\cdot) \in \mathrm{Op}^{(n\cdot m, m\cdot r)}_{n\cdot r}$;
- operations for summing all the elements in an array: $\mathsf{sum} \in \mathrm{Op}^n_1$;
- some non-linear functions like the sigmoid function $\varsigma \in \mathrm{Op}^1_1$.

Primitive operations with coproducts in the codomain will be more important below, after adding partial features to our source language, in Section A. The reader interested in the detailed rules for this standard functional programming language can consult Appendix A.

### 2.2 Categorical semantics of the source total language

A bicartesian category $\mathcal{C}$ is *distributive* if, for each triple of objects $A, B, C$ in $\mathcal{C}$, the canonical morphism

$$[A \times \iota_B, A \times \iota_C] : (A \times B) \sqcup (A \times C) \to A \times (B \sqcup C) \tag{2.2}$$

is invertible. Alternatively, there is an isomorphism $(A \times B) \sqcup (A \times C) \cong A \times (B \sqcup C)$ natural in $A, B, C$ (see, for instance, [20, 36, 37, 40]).

While not strictly required in our context, for the sake of simplicity and convenience, we consistently adopt the assumption that *distributive categories* have chosen (finite) products and coproducts. Conveniently, we require that *structure-preserving functors* between these distributive categories strictly preserve finite products and coproducts, that is to say, *(strictly) bicartesian functors.*

The primitive types and operations of the source language can be framed within a structured graph/computad, *e.g.* [33, Section 4], [34, 1.6], [9] and [50]. We can take the freely generated *distributive category* $\mathbf{Syn}_{tot}$ over this structured computad (primitive types and operations) as the abstract categorical semantics as our representation of the (source language) syntax.[2] More explicitly, the universal property of $\mathbf{Syn}_{tot}$ can be described as follows:

Theorem 2.1. *The distributive category* $\mathbf{Syn}_{tot}$ *corresponding to our (total) source language has the following universal property. Given any distributive category* $C$*, for each pair* $(K, \mathtt{h})$ *where* $K = (K_n)_{n\in\mathbb{N}}$ *is a family of objects in* $C$ *and* $\mathtt{h} = \left(h_{\mathrm{op}}\right)_{\mathrm{op}\in\mathrm{Op}}$ *is a consistent family of morphisms in* $C$*, there is a unique structure-preserving functor (strictly bicartesian functor)*

$$H : \mathbf{Syn}_{tot} \to C \tag{2.3}$$

*such that* $H(\mathbf{real}^n) = K_n$ *and* $H(\mathrm{op}) = h_{\mathrm{op}}$ *for any* $n \in \mathbb{N}$ *and any primitive operation* $\mathrm{op} \in \mathrm{Op}$.

In order to establish a (denotational) correctness proof, we give concrete semantics to our source language. For the purposes of this paper, we define the concrete semantics of the source language on the category $\mathcal{VM}an$ of (differentiable) manifolds (with variable dimensions). More precisely, since $\mathcal{VM}an$ is a distributive category, by the universal property of $\mathbf{Syn}_{tot}$, we can define the concrete semantics as a structure-preserving functor

$$[\![-]\!] : \mathbf{Syn}_{tot} \to \mathcal{VM}an. \tag{2.4}$$

Theorem 2.2 (Semantics Functor). *There is only one (strictly) bicartesian functor* (2.4) *such that, for each* $n \in \mathbb{N}$, $[\![\mathbf{real}^n]\!] = \mathbb{R}^n$ *and, for each* $(\mathbf{n}, \mathbf{m}) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2}$, $[\![\mathrm{op}]\!]$ *is the differentiable function (morphism of* $\mathcal{VM}an$*)* $\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_{k_1}} \to \mathbb{R}^{m_1} \sqcup \cdots \sqcup \mathbb{R}^{m_{k_2}}$ *that* op *intends to implement.*

Proof. Since $\mathcal{VM}an$ is a distributive category, the result follows from the universal property of $\mathbf{Syn}_{tot}$ established in Theorem 2.1. □

### 2.3 Target total language and its categorical semantics

The target language of CHAD is a variant of the dependently typed enriched effect calculus [52, Chapter 5]. Its cartesian types, linear types, and terms are generated by the grammar of Fig. A.1 and B.1, making the target language a proper extension of the source language. We describe the full language in more detail in Appendix B.

We work with linear operations $\mathsf{lop} \in \mathsf{LOp}^{\mathbf{r};\mathbf{n}}_{\mathbf{m}}$, which are intended to represent functions that are linear (in the sense of respecting $0^{\mathrm{v}}$ and +) in the arguments corresponding to $\mathbf{n} = \left\{n_1, \ldots, n_{k_1}\right\}$ but not in the arguments corresponding to $\mathbf{m} = \left\{m_1, \ldots, m_{k_2}\right\}$.

We write

$$\mathbf{LDom}(\mathsf{lop}) \stackrel{\mathrm{def}}{=} \mathbf{real}^{n_1} \times \ldots \times \mathbf{real}^{n_{k_1}} \qquad \text{and} \qquad \mathbf{CDom}(\mathsf{lop}) \stackrel{\mathrm{def}}{=} \mathbf{real}^{m_1} \times \ldots \times \mathbf{real}^{m_{k_2}}$$

for $\mathsf{lop} \in \mathsf{LOp}^{\mathbf{r};\mathbf{n}}_{\mathbf{m}}$. The details of the typing rules and equational are described in Appendix B, and it is a fragment of the target language presented in [30].

To serve as a practical target language for the automatic derivatives of all programs from the source language, we make the following assumption: for each $(k_1, k_2) \in \mathbb{N}$, each $(\mathbf{n}, \mathbf{m}) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2}$, and each $\mathrm{op} \in \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$, we have a chosen program

$$\begin{array}{l} x_1 : \mathbf{real}^{n_1}, \ldots, x_{k_1} : \mathbf{real}^{n_{k_1}}; \mathsf{v} : \mathbf{case}\,\mathrm{op}(x_1, \ldots, x_{k_1})\,\mathbf{of}\,\{\mathbf{in}_1\,\_ \to \underline{\mathbf{reals}}^{m_1} \mid \cdots \mid \mathbf{in}_{k_2}\,\_ \to \underline{\mathbf{reals}}^{m_{k_2}}\} \\ \quad \vdash D\mathrm{op}(x_1, \ldots, x_{k_1})(\mathsf{v}) : \underline{\mathbf{reals}}^{n_1} \times \cdots \times \underline{\mathbf{reals}}^{n_{k_1}} \end{array} \tag{2.5}$$

that intends to implement a function that gives the (transpose) derivative of op. In particular, in case $k_2 = 1$, we have that

$$x_1 : \mathbf{real}^{n_1}, \ldots, x_{k_1} : \mathbf{real}^{n_{k_1}}; \mathsf{v} : \underline{\mathbf{reals}}^{m_1} \vdash D\mathrm{op}(x_1, \ldots, x_{k_1})(\mathsf{v}) : \underline{\mathbf{reals}}^{n_1} \times \cdots \times \underline{\mathbf{reals}}^{n_{k_1}}.$$

[2] See [7, 9, 25, 35, 37, 39–41, 50] for works on freely generated categorical structures and their framing in two-dimensional monad theory; we intend to provide a detailed exposition of these foundations in our precise setting in future work.

There is a freely generated strictly indexed category on the target language, endowed with a suitable structure (of dependently typed enriched effect calculus, *e.g.* [52, Chapter 5] and [30, Section 7]). More precisely, in the terminology of [30], this strict indexed category, denoted by $\mathbf{LSyn}_{tot} : \mathbf{CSyn}_{tot}^{\mathrm{op}} \to \mathsf{Cat}$ herein, is a $\Sigma$-bimodel for *tuple and variant types* that is freely generated on the primitive linear operations described above.

This provides us with Lemma 2.3 on the op-Grothendieck construction $\Sigma_{\mathbf{CSyn}_{tot}}\mathbf{LSyn}_{tot}^{op}$ of the syntactic categorical semantics $\mathbf{LSyn}_{tot} : \mathbf{CSyn}_{tot}^{\mathrm{op}} \to \mathsf{Cat}$ of the target language. This is particularly useful, since the basic tenet of CHAD is to follow the recipe given by the concrete semantics, and give the macro as a structure preserving functor from the source language category and the op-Grothendieck construction of the target language, following the concrete semantics given by Theorem 1.1.

Lemma 2.3. $\Sigma_{\mathbf{CSyn}_{tot}}\mathbf{LSyn}_{tot}^{\mathrm{op}}$ *is a distributive category.*

Proof. We refer the reader to [30, Section 6] for more details. □

Finally, we can define the concrete categorical semantics of our target language, in terms of structure-preserving indexed functor $\mathbf{LSyn}_{tot} \to \mathbf{Vect}^{(-)}$ . If the reader is not familiar with indexed functors, we stress that we the induced strictly bicartesian functor (2.7) plays the most important role in our development.[3]

Before establishing the concrete semantics of our target language in Lemma 2.4, we introduce the following notation for each $\mathbf{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$. We denote by

$$\overline{\mathbb{R}^{\mathbf{n}}} : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k} \to \mathbf{Vect}, \qquad \overline{\mathbb{R}_{\mathbf{n}}} : \mathbb{R}^{n_1} \sqcup \cdots \sqcup \mathbb{R}^{n_k} \to \mathbf{Vect} \tag{2.6}$$

respectively, the constant functor/family equal to $\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k}$ and the family/functor defined by $\overline{\mathbb{R}_{\mathbf{n}}}(x) = \mathbb{R}^{n_t}$ if $x \in \mathbb{R}^{n_t}$.

Lemma 2.4 (Concrete semantics of the target language). *The strict indexed category* $\mathbf{Vect}^{(-)} : \mathsf{Set}^{\mathrm{op}} \to \mathsf{Cat}$ *is a* $\Sigma$*-bimodel for tuple and variant types. Therefore, there is a semantics indexed functor* $\mathbf{LSyn}_{tot} \to \mathbf{Vect}^{(-)}$ *which induces a structure-preserving (in particular, strictly bicartesian) functor*

$$[\![-]\!] : \Sigma_{\mathbf{CSyn}_{tot}}\mathbf{LSyn}_{tot}^{op} \to \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \tag{2.7}$$

*such that*

*a) for each n-dimensional array* $\mathbf{real}^n \in \mathbf{Syn}_{tot}$, $[\![\mathbf{real}^n]\!] \overset{\mathrm{def}}{=} \mathbb{R}^n \in \mathrm{ob}\,(\mathsf{Set})$*;*

*b) for each n-dimensional array* $\mathbf{real}^n \in \mathbf{Syn}_{tot}$,

$$\overline{[\![\mathbf{real}^n]\!]} \overset{\mathrm{def}}{=} \overline{\mathbb{R}^n} \in \mathbf{Vect}^{\mathbb{R}^n}.$$

*As consequence, for each* $\mathbf{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$*, we have that*

$$\overline{[\![\mathbf{real}^{\mathbf{n}}]\!]} \overset{\mathrm{def}}{=} \overline{[\![\mathbf{real}^{\mathbf{n_1}} \times \cdots \times \mathbf{real}^{\mathbf{n_k}}]\!]} = \overline{\mathbb{R}^{\mathbf{n}}} \in \mathbf{Vect}^{\mathbb{R}^{\mathbf{n}}};$$

*c) for each primitive* $\mathrm{op} \in \mathrm{Op}_{\mathbf{m}}^{\mathbf{n}}$*, where* $\mathbf{n} = \left(n_1, \ldots, n_{k_1}\right)$ *and* $\mathbf{m} = \left(m_1, \ldots, m_{k_2}\right)$*:*

*i)* $[\![\mathrm{op}]\!] : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_{k_1}} \to \mathbb{R}^{m_1} \sqcup \cdots \sqcup \mathbb{R}^{m_{k_2}}$ *is the map in* $\mathbf{Set}$ *corresponding to the operation that* op *intends to implement;*

*ii)* $[\![D\mathrm{op}]\!] \in \mathbf{Vect}^{\left([\![\mathrm{real}^{n_1}]\!] \times \cdots \times [\![\mathrm{real}^{n_{k_1}}]\!]\right)}\left(\underline{\overline{[\![\mathbf{real}_{\mathbf{m}}]\!]}} \circ [\![\mathrm{op}]\!], \overline{[\![\mathbf{real}^{\mathbf{n}}]\!]}\right)$*, where* $\underline{\overline{[\![\mathbf{real}_{\mathbf{m}}]\!]}} = \overline{\mathbb{R}_{\mathbf{m}}}$ *is the family of linear transformations that* $D\mathrm{op}$ *intends to implement.*

Proof. We refer the reader to [30, Section 6 and Section 7] for more details. □

[3] We refer the reader, for instance, to [30, Section 6.9] for indexed functors.

### 2.4 Macro and correctness

CHAD is a program transformation between the source language discussed in Section 2.1 and the target language describe in 2.3. We firstly establish how to correctly implement the CHAD-derivatives of the primitive (types and) operations, respecting the concrete semantics defined in 2.2 and Lemma 2.4. Then, we define CHAD as the only *structure-preserving transformation* between the source language and the total category (op-Grothendieck construction) obtained from the target language that extends the implementation on the primitive operations. This allows for a straightforward correctness proof (presented in Theorem 2.6).

As discussed earlier, our present approach starts with primitive operations that are differentiable over the entire domain. For clarity and conciseness, we present the macro in this simplified setting. However, the approach can be generalized to accommodate primitive operations with singularities, or even more complex settings, without affecting the correctness proof. This broader generalization is left to future work.

We refer to Subsect. refsec:MACRO for the precise definition of the our macro $\underleftarrow{\mathbb{D}}(-)$, restricting its attention to the total language fragment of the source language and target languages (for a detailed revision of CHAD for total languages, we refer the reader to [30, Section 8]). At the categorical semantics level, the (total fragment) of the macro defined in Subsect. 5.4 corresponds to the following functor.

Theorem 2.5. *There is only one (strictly) bicartesian functor*

$$\underleftarrow{\mathbb{D}} : \mathbf{Syn}_{tot} \to \Sigma_{\mathbf{CSyn}_{tot}} \mathbf{LSyn}_{tot}^{op} \tag{2.8}$$

*such that, for each* $n \in \mathbb{N}$,

$$[\![ \underleftarrow{\mathbb{D}}(\mathbf{real}^n) ]\!] = \left( \mathbb{R}^n, \overline{\mathbb{R}^n} \right), \tag{2.9}$$

*and, for each* $\mathrm{op} \in \mathrm{Op}$, $\underleftarrow{\mathbb{D}}(\mathrm{op})$ *implements the CHAD-derivative of the primitive operation* $\mathrm{op} \in \mathrm{Op}$ *of the source language, that is to say,*

$$\underleftarrow{\mathbb{D}}(\mathrm{op}) = (\mathrm{op}, D\mathrm{op}) \qquad \textit{or, in other words,} \qquad [\![ \underleftarrow{\mathbb{D}}(\mathrm{op}) ]\!] = \mathfrak{D}[\![\mathrm{op}]\!], \tag{2.10}$$

*where* $\mathfrak{D}$ *is the CHAD-derivative functor defined in 1.2. The functor* $\underleftarrow{\mathbb{D}}$ *corresponds to the total fragment of the macro* $\underleftarrow{\mathbb{D}}()$ *defined in Subsect. 5.4.*

Proof. Since $\Sigma_{\mathbf{CSyn}_{tot}} \mathbf{LSyn}_{tot}^{op}$ is a distributive category, the result follows from the universal property of $\mathbf{Syn}_{tot}$ (Theorem 2.1). □

Finally, we can state and prove correctness of CHAD for the total language setting above. Firstly, observe that:

Theorem 2.6. *The diagram below commutes.*

$$\begin{array}{ccc} \mathbf{Syn}_{tot} & \xrightarrow{\underleftarrow{\mathbb{D}}} & \Sigma_C \mathbf{LSyn}_{tot}^{op} \\ {\scriptstyle [\![-]\!]}\downarrow & & \downarrow{\scriptstyle [\![-]\!]} \\ \mathcal{VM}an & \xrightarrow[\mathfrak{D}]{} & \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \end{array} \tag{2.11}$$

Proof. By the universal property of the distributive category $\mathbf{Syn}_{tot}$ established in Theorem 2.1, since $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ is a distributive category, we have that there is only one (strictly) bicartesian functor

$$\mathfrak{C} : \mathbf{Syn}_{tot} \to \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \tag{2.12}$$

such that

c1) for each $n \in \mathbb{N}$, $\mathfrak{C}\left(\mathbf{real}^n\right) = \left(\mathbb{R}^n, \overline{\mathbb{R}^n}\right)$;

c2) $\mathfrak{C}(\mathrm{op}) = \mathfrak{D}[\![\mathrm{op}]\!]$ for each $\mathrm{op} \in \mathrm{Op}$.

Since both $\mathfrak{D}[\![-]\!] : \mathbf{Syn}_{tot} \to \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ and $[\![\overleftarrow{\mathbb{D}}(-)]\!] : \mathbf{Syn}_{tot} \to \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ are strictly bicartesian functors such that c1) and c2) hold, we get that $\mathfrak{D}[\![-]\!] = \mathfrak{C} = [\![\overleftarrow{\mathbb{D}}(-)]\!]$. That is to say, Diagram (2.11) commutes. □

Corollary 2.7. *For any well-typed program $x : \tau \vdash t : \sigma$, we have that $[\![\overleftarrow{\mathbb{D}}(t)]\!] = \mathfrak{D}[\![t]\!]$.*

Proof. It follows from the fact that $[\![\overleftarrow{\mathbb{D}}(t)]\!] = \mathfrak{D}[\![t]\!]$ for any morphism $t$ of $\mathbf{Syn}_{tot}$. □

## 3 ITERATIVE CHAD: BASIC CONCRETE SEMANTICS

The goal of the present work is to extend CHAD to accommodate languages with partial features. Particularly, we focus on iteration constructs, which are relevant for efficiently implementing CHAD in languages that leverage parallelism and support iteration, such as Accelerate. Extensions to languages with recursive features, including CHAD for PCF [43] and FPC [14], are left for future work.

CHAD is fundamentally principled by a concrete denotational semantics that adheres to the principle of computing CHAD-derivatives in a functorial and *structure-preserving* way. We aim to follow the same tenet while introducing partial features to our framework. The initial step, therefore, is to define the concrete denotational semantics for our extended setting.

Establishing denotational semantics for derivatives of partially defined functions presents several possibilities, especially regarding differentiability over domains [16] and the so-called *if-problem* [6]. For the sake of clarity and conciseness, we extend of our previous work on CHAD for total languages by considering primitive operations that are "differentiable on their entire domain" and by following Abadi-Plotkin solution to the if-problem [1], as we outline below.

Our approach aligns with our previous work on AD in dual-numbers style [38, 39], and is the natural *extension* of our work on CHAD for total languages [30]. Furthermore, our approach is suitable for several practical AD applications, particularly in the context of our implementations [47, 48]. Although outside of our scope, we claim that CHAD could be adapted to various concrete semantics, including alternative ones such as [16] along similar lines to the development for dual numbers AD in [39, Section 10]. Adapting CHAD to these concrete semantics will provide us with different approaches to the if-problem and non-differentiability. A detailed exploration of these extensions is deferred to future work.

### 3.1 Derivatives of partially defined functions, and the if-problem

We extend CHAD to languages with partial features by closely following our approach for total languages. Specifically, we adapt the previously established concrete semantics to accommodate the partial constructs. For clarity and conciseness of our exposition, we focus here on applying CHAD to a (fine-grain) call-by-value language, noting that similar work for call-by-push-value and call-by-need languages are possible, albeit with additional technical details.[4]

The primary challenge in this extension arises from addressing partiality in the presence of differentiability. We need to formally define the denotational semantics for the derivatives of functions that are only partially defined. In essence, while extending the concrete semantics presented in Section 1, our work is about providing a domain structure to $\mathcal{VMan}$.

[4]We refer the reader to [23, 24, 38, 39, 52] for more details on the semantics of call-by-value and call-by-push-value languages.

Extending the concrete semantics to a call-by-value language involves providing semantics not just for values but also for computations, which exhibits partiality. As discussed, we have two possible approaches to extend our concrete semantics in this setting.

The first approach is to extend the category $\mathcal{VM}an$ to include more morphisms, effectively relaxing the semantics of values to encompass functions that are not differentiable over their entire domains, which would add partial differentiability to our total language, or even consider more intricate alternative semantics in the line of [16].

*We emphasize that CHAD is compatible with alternative semantics since the basic structure-preserving tenet of CHAD holds in any suitable semantics for automatic differentiation.* Nevertheless, we adopt the second approach which we outline below: it consists of maintaining our existing choice for the semantics of values – restricting them to total, differentiable functions (morphisms of $\mathcal{VM}an$) – and handling partiality and non-differentiability at the level of computations by considering a standard domain structure of open subsets. By adopting this approach, we are able to give a straightforward exposition of the basic principles of CHAD for iteration, ensuring compatibility with our previous expositions on CHAD [30, 54].

*3.1.1 Freyd-categorical structure over $\mathcal{VM}an$.* Freyd categories provide the standard categorical semantics for (fine-grain) call-by-value languages. We quickly recall the definition of *Freyd categories*[5] below, although we refer to [23, 49] for further details.

*Definition 3.1 (Freyd category).* A *(distributive) Freyd category*, herein, is a quadruple $(\mathcal{V}, \mathcal{C}, j, \otimes)$ where $\mathcal{V}$ is a distributive category, and:

- $\otimes$ is an action[6] of the cartesian category $\mathcal{V}$ on the *category* $\mathcal{C}$ that distributes over (finite) coproducts in the sense that $A \otimes (-)$ preserves (finite) coproducts, denoted as

$$(-\otimes -) : \mathcal{V} \times \mathcal{C} \to \mathcal{C}; \tag{3.1}$$

- $j : \mathcal{V} \to \mathcal{C}$ is a strictly (finite) coproduct-preserving functor that is identity on objects such that $j(A \times B) = A \otimes B$ for any pair $A, B \in \mathrm{obj}\mathcal{C}$.

A Freyd category morphism $(\mathcal{V}, \mathcal{C}, j, \otimes) \to (\mathcal{V}', \mathcal{C}', j', \otimes')$ between Freyd categories is a pair $\left(F, \hat{F}\right)$ of functors such that $F$ is strictly bicartesian, and 3.2 commutes.[7] It should be noted that Freyd categories and Freyd category morphisms form a category.

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\hat{F}} & \mathcal{C}' \\ {\scriptstyle j}\uparrow & & \uparrow{\scriptstyle j'} \\ \mathcal{V} & \xrightarrow[F]{} & \mathcal{V}' \end{array} \tag{3.2}$$

The aim of this section is to extend the (semantical) CHAD-derivative functor presented in Theorem 1.1 to a Freyd category morphism, establishing how the CHAD-derivative interacts with non-termination, non-differentiability and iteration.

The starting point is to extend the semantics of the source language to include computations that involve non-termination and non-differentiable functions. In order to do so, we consider the domain structure induced by open subsets on $\mathcal{VM}an$.

[5] What we call Freyd category herein is usually called *distributive Freyd category*, *e.g.* [49].
[6] Recall that this means that $\otimes$ comes from a strictly monoidal functor from the cartesian category $(\mathcal{V}, \times, 1)$ to the monoidal category of endofunctors $([\mathcal{C}, \mathcal{C}], \circ, \mathrm{id})$.
[7] It should be noted that, under the conditions of the definition, we can conclude that $\mathcal{C}$ has finite coproducts and $\hat{F}$ preserves finite coproducts, *e.g.* [36].

*Definition 3.2 (Open Domain Structure).* We consider the subcategory $\mathfrak{D}\mathcal{M}an$ of $\mathcal{VM}an$ consisting of all objects of $\mathcal{VM}an$ and open embeddings/inclusions $V \subseteq M$ as morphisms.

The pair $(\mathcal{VM}an, \mathfrak{D}\mathcal{M}an)$ is a domain structure for $\mathcal{VM}an$ (see, for instance, [13, Def. 3.1.1]). It gives rise to the category of partial maps $\mathfrak{P}\mathcal{M}an$, and we roughly denote $\mathfrak{P}\mathcal{M}an = (\mathcal{VM}an, \mathfrak{D}\mathcal{M}an)$.

In other words, $\mathfrak{P}\mathcal{M}an$ is the category of manifolds, where the morphisms are differentiable functions that are only defined on open subsets. Explicitly, the category $\mathfrak{P}\mathcal{M}an$ has manifolds as objects. A morphism $\mathtt{f} : M \to N$ in $\mathfrak{P}\mathcal{M}an$ is a pair $\mathtt{f} = (V, f)$ where $V \subseteq M$ is an open subset, and $f : V \to N$ is a morphism in $\mathcal{VM}an$. In this setting, $\mathtt{domain}\,(\mathtt{f}) := V$ is called the domain of $\mathtt{f}$, while $f$ is the underlying function.

Composition of morphisms in $\mathfrak{P}\mathcal{M}an$ is given by the usual pullback/span style. Explicitly, given morphisms $\mathtt{f} = (V, f) : M \to N$ and $\mathtt{g} = (W, g) : N \to K$, the composition is given by

$$\mathtt{g} \circ \mathtt{f} = \left(V \cap f^{-1}(W), \mathtt{gf}\right) \tag{3.3}$$

where $\mathtt{gf} : V \cap f^{-1}(W) \to K$ is defined by $\mathtt{gf}(x) = g(f(x))$ for each $x \in V \cap f^{-1}(W)$.

Finally, let $\mathfrak{p} : \mathcal{VM}an \to \mathfrak{P}\mathcal{M}an$ be the obvious inclusion functor that sends $f : M \to N$ to $(M, f)$. We have the Freyd category

$$\mathcal{FM}an := (\mathcal{VM}an, \mathfrak{P}\mathcal{M}an, \mathfrak{p}, \otimes)\,, \tag{3.4}$$

where, for each morphism $f : M \to N$ of $\mathcal{VM}an$ and each morphism $\mathtt{g} = (W, g) : R \to K$ of $\mathfrak{P}\mathcal{M}an$,

$$f \otimes (W, g) := (M \times W, \mathfrak{p}\,(f \times g) : \mathfrak{p}\,(M \times W) \to \mathfrak{p}\,(N \times K)) : \mathfrak{p}\,(M \times R) \to \mathfrak{p}\,(N \times K)\,.$$

$\mathcal{FM}an$ serves as a concrete model for our (fine grain) call-by-value language.

*3.1.2 Enrichment of $\mathfrak{P}\mathcal{M}an$.* Recall that a pointed $\omega$-cpo (pointed and $\omega$-chain complete partially ordered set) is a partially ordered set that has least element, *usually denoted by* $\bot$, and colimits (least upper bounds) of all $\omega$-chains (countable ascending sequences). A morphism between pointed $\omega$-cpos is a monotonic function that preserves colimits of $\omega$-chains and the bottom element. We denote the category of pointed $\omega$-cpos and their morphisms by $\boldsymbol{\omega}\mathbf{Cpo}_\bot$.

*Remark 3.3.* It should be noted that $\boldsymbol{\omega}\mathbf{Cpo}_\bot$ has coproducts and products. More precisely, by denoting $(-)_0 : \boldsymbol{\omega}\mathbf{Cpo}_\bot \to \mathsf{Cat}$ the functor that takes each pointed $\omega$-cpo to the underlying category, we have the following. For each $(A, B) \in \mathrm{obj}\left(\boldsymbol{\omega}\mathbf{Cpo}_\bot\right) \times \mathrm{obj}\left(\boldsymbol{\omega}\mathbf{Cpo}_\bot\right)$, we have that $(A \times B)_0 = (A)_0 \times (B)_0$, and $(A \sqcup B)_0$ is obtained by taking the coproduct of $A_0$ and $B_0$ and identifying the initial objects of $A_0$ and $B_0$. Explicitly, $A \times B$ is the product of sets with the product order in which $(a, b) \leq_{A\times B} (a', b')$ iff $a \leq_A a'$ and $b \leq_B b'$, where $\bot_{A\times B} = (\bot_A, \bot_B)$. $A \sqcup B$ is the wedge sum of pointed sets with the union of the orders.

We consider the category $\boldsymbol{\omega}\mathbf{Cpo}_\bot$ equipped with a monoidal product known as the smash product, denoted by $\star$. Given pointed $\omega$-cpos $A$ and $B$, the smash product $A \star B$ is constructed by taking the cartesian product $A \times B$ and, then, identifying all pairs where at least one of the components is $\bot$. Formally, we define an equivalence relation $\sim$ on $A \times B$ generated by:

$$(\bot, b) \sim (a, \bot) \text{ for any } (a, b) \in A \times B.$$

The elements of $A \star B$ are the equivalence classes of this relation, while the partial order is the order on $A \times B$ modulo this equivalence relation. We can think about $A \star B$ as

$$((A - \{\bot\}) \times (B - \{\bot\})) \cup \{\bot\}$$

with $\bot$ as the least element and the usual cartesian product order on the rest of the elements. Finally, the identity/unit object of this smash product is given by the pointed $\omega$-cpo with two elements $2 \stackrel{\text{def}}{=} \{\top, \bot\}$, the bottom and top elements.

Herein, an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched category $C$ is a category enriched over the monoidal category $(\boldsymbol{\omega}\mathbf{Cpo}_{\perp}, \star, 2)$. Concretely, this means that for any objects $A, B, C$ in $C$, the hom-object $C(A, B)$ is an object of $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$, and the composition operations

$$\circ : C(B, C) \star C(A, B) \to C(A, C)$$

are $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-morphisms.[8] *In the setting above, we denote by $\perp$ the least element in the hom-object $C(A, B)$ for any $(A, B) \in \mathrm{obj}C \times \mathrm{obj}C$.*

Theorem 3.4. *The monoidal category $(\boldsymbol{\omega}\mathbf{Cpo}_{\perp}, \star, 2)$ is (monoidal) closed, where the order in the set $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}(W, Y)$ of $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-functors is defined pointwise. In other words, $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$ is $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched.*

Lemma 3.5. *$\mathfrak{P}\mathcal{M}an$ is naturally $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched when endowed with the domain-order for the morphisms. That is to say, given $\mathtt{f}, \mathtt{g} : M \to N$, we say that $\mathtt{f} \leq \mathtt{g}$ if $\mathtt{domain}(\mathtt{f}) \subseteq \mathtt{domain}(\mathtt{g})$ and $f(x) = g(x)$ for any $x \in \mathtt{domain}(\mathtt{f})$.*

Proof. Given an $\omega$-chain of domains (open subsets of a manifold $M$), the union of all of them is an open set. Therefore the colimit of

$$\mathtt{f}_0 = (U_0, f_0) \leq \mathtt{f}_1 = (U_1, f_1) \leq \mathtt{f}_2 = (U_2, f_2) \leq \cdots \leq \mathtt{f}_n = (U_n, f_n) \leq \cdots$$

in $\mathfrak{P}\mathcal{M}an(M, N)$ is defined by

$$\mathtt{f} = \left( \bigcup_{t \in \mathbb{N}} U_t, f \right) : M \to N, \qquad f(x) := f_t(x), \text{ if } x \in U_t, \tag{3.5}$$

which is also a morphism of $\mathfrak{P}\mathcal{M}an$. The least element of $\mathfrak{P}\mathcal{M}an(M, N)$ is given by the only morphism $\perp := (\emptyset, \emptyset \to N) : M \to N$. □

*Definition 3.6 ($\omega$-Freyd category).* A quadruple $(\mathcal{V}, C, j, \otimes)$ is an *$\omega$-Freyd category* if it is a Freyd category such that $C$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched category.

A pair $\left(F, \hat{F}\right)$ an *$\omega$-Freyd category morphism* if $\left(F, \hat{F}\right)$ is a Freyd category morphism between $\omega$-Freyd categories, and $\hat{F}$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched functor.

Theorem 3.7. *$\mathcal{F}\mathcal{M}an := (\mathcal{V}\mathcal{M}an, \mathfrak{P}\mathcal{M}an, \mathfrak{p}, \otimes)$ is an $\omega$-Freyd category.*

*3.1.3 A word on the if-problem.* By defining the domain structure $\mathfrak{P}\mathcal{M}an = (\mathcal{V}\mathcal{M}an, \mathfrak{D}\mathcal{M}an)$, we inherently establish our approach to the *if-problem,* [6] which concerns handling singularities arising from branches in conditional expressions.

In our semantics, we restrict ourselves to open domains. Therefore, in our language, any construct of the form `if-then-else` can only define functions whose domains are open sets and, hence, undefined in singularities.

This approach follows the solution proposed by Abadi and Plotkin [1] and is general enough to encompass our practical goals. We emphasize that we have made this choice for clarity and simplicity – however, as stressed above, CHAD is also compatible with other semantic frameworks such as that of [16] or [39, Section 10].

[8] We do not delve deeply into the theory of enriched categories here. Interested readers are directed to [11, 13, 19, 24, 26, 28, 29, 39, 55] for comprehensive treatments of the general theory of enriched categories and the specific case of $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enrichment.

### 3.2 Concrete semantics for the target language

In order to follow our structure-preserving principle, we define our (concrete) CHAD-derivative as a structure-preserving functor in our extended setting – that is, we define CHAD as a suitable Freyd category morphism. In order to do so, we extend the concrete categorical semantics of our target language into a suitable Freyd-category, that handles partially defined differentiable functions.

*3.2.1 Basics of the concrete semantics of the target language.* Although we do not assume any knowledge on indexed monads to further understand this work, we stress that, denoting by 1 the set containing a single element $\perp$, we model computations in our target language by considering the indexed monad[9] $\mathfrak{T}$ on the indexed category $\mathbf{Vect}^{(-)} : \mathbf{Set}^{\mathrm{op}} \to \mathsf{Cat}$ induced by the monad $(- \sqcup 1) : \mathbf{Set} \to \mathbf{Set}$. More precisely, $\mathfrak{T}$ is defined by the indexed functor

$$\begin{array}{ccc} & \mathsf{Cat} & \\ {\scriptstyle \mathbf{Vect}^{(-)}}\nearrow & \overset{\psi}{\Longrightarrow} & \nwarrow{\scriptstyle \mathbf{Vect}^{(-)}} \\ \mathbf{Set}^{\mathrm{op}} & \xrightarrow[(-\sqcup 1)]{} & \mathbf{Set}^{\mathrm{op}} \end{array} \tag{3.6}$$

with the obvious multiplication and unit. Here, $\psi$ is the natural transformation defined pointwise by

$$\psi_A : \mathbf{Vect}^A \to \mathbf{Vect}^{(A\sqcup 1)}, \qquad (X : A \to \mathbf{Vect}) \mapsto \left([X, \overline{0}] : A \sqcup 1 \to \mathbf{Vect}\right) \tag{3.7}$$

where $\overline{0} : 1 \to \mathbf{Vect}$ is the single family consisting of the zero object of $\mathbf{Vect}$, and $[X, \overline{0}] : M \sqcup 1 \to \mathbf{Vect}$ is the family defined by $X$ and $\overline{0}$.

More importantly to our exposition, the (indexed) monad $\mathfrak{T}$ induces the monad

$$\mathcal{T}(M, X) = (M, X) \sqcup \mathbf{1} = \left(M \sqcup 1, [X, \overline{0}]\right) \tag{3.8}$$

on $\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, where we denote by $\mathbf{1}$ the terminal object $\left(1, \overline{0}\right)$ of $\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. The Kleisli category of the monad $\mathcal{T}$, denoted by $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, is the standard way of modelling partial computations on $\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, by adjoining an extra point (terminal) to the codomain of morphisms, capturing non-termination in the usual way.

Explicitly, recall that a morphism $f : (A, X) \to (B, Y)$ in $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ corresponds to a morphism

$$(\mathtt{f}, \mathtt{f}') : (A, X) \to \left(B \sqcup 1, [Y, \overline{0}]\right)$$

in $\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$. In this setting, we define the domain of $f$ (or $\mathtt{f}$) as $\mathtt{domain}(f) \coloneqq \mathtt{f}^{-1}(B)$; that is, the set of all $a \in A$ such that $\mathtt{f}(a) \in B$. We say that $f$ (or $\mathtt{f}$) is *defined* at $a \in A$ if $\mathtt{f}(a) \neq \perp$; we say that $f$ (or $\mathtt{f}$) is undefined at $a$ otherwise. It should be noted that, when $f$ is undefined at $a \in A$, we have that $\mathtt{f}'_a$ is the unique morphism $0 \to X(a)$ from the zero vector space.

Lemma 3.8. *There is a bijection between morphisms $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})((A, X), (B, Y))$ and pairs $\left(V, \hat{f}\right)$ where $V \subseteq A$ and $\hat{f} = (f, f')$ is a morphism $(V, X|_V) \to (B, Y)$ in $\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, where we denote by $X|_V$ the restriction of $X$ to $V$.*

Proof. It is enough to see that we have such a correspondence when we set

$$\mathbf{f} \mapsto (\mathtt{domain}(\mathbf{f}), f, f') \tag{3.9}$$

[9] The interested reader can find more about indexed functors, for instance, in [30, 52].

where $(f, f')$ is the appropriate restriction of the morphism $(\mathtt{f}, \mathtt{f}') : (A, X) \to \left(B \sqcup 1, [Y, \overline{0}]\right)$ that corresponds to $f$ in $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$. Explicitly, $f := \mathtt{f}|_V$ and $f'_a := \mathtt{f}'_a$ for each $a \in V \subseteq A$. □

*Henceforth, we denote by* $\mathbf{f} = (V, f, f') : (A, X) \to (B, Y)$ *the morphisms of* $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$. With this notation, if $\mathbf{f} = (V, f, f') : (A, X) \to (B, Y)$ and $\mathbf{g} = (W, g, g') : (B, Y) \to (C, Z)$ are morphisms of $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$, the composition is given by

$$\mathbf{g} \circ \mathbf{f} := \left(V \cap f^{-1}(W), (gf)_{VW}, (gf)'\right) \tag{3.10}$$

where $\left((gf)_{VW}, (gf)'\right) : \left(V \cap f^{-1}(W), X_{V\cap f^{-1}(W)}\right) \to (C, Z)$ is defined by $(gf)_{VW}(a) := g(f(a))$ and $(gf)'_a := f'_a \circ g'_{f(a)}$ for each $a \in V \cap f^{-1}(W)$.

The above, together with Lemma 3.8, shows that $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ is obtained from the domain structure induced by subset inclusions (with identities as second coordinate) in $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$. More importantly to our setting, denoting by $\mathfrak{j} : \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \to \mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ the standard Kleisli inclusion functor, we have the Freyd category

$$\mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) := \left(\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \mathfrak{j}, \otimes\right), \tag{3.11}$$

where, for each morphism $(f, f') : (A, X) \to (B, Y)$ in $\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ and each morphism $(W, g, g') : (C, W) \to (D, Z)$ in $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$,

$$(f, f') \otimes (W, g, g') := \left(A \times W, \mathfrak{j}((f, f') \times (g, g'))\right).$$

*3.2.2* $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ *as an op-Grothendieck construction.* The category $\mathsf{PSet}$ of sets and partially defined functions can be seen as a full subcategory of the category $\mathsf{Cat}_{\omega\perp}$ of the categories with initial objects and colimits of $\omega$-chains, and structure-preserving functors as morphisms. More precisely, we have a full inclusion functors

$$\mathsf{PSet} \xrightarrow{(-)_\perp} \boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{Cat}_{\omega\perp}, \tag{3.12}$$

in which the second arrow is the usual inclusion, while the first one is defined as follows. It takes each set $A$ to the pointed pointed $\omega$-cpo $(A)_\perp$ given by $A \sqcup \{\perp\}$ where $\perp$ is the bottom element, and $x \leq y$ implies either $x = y$ or $x = \perp$; each partially defined morphism $f : A \to B$ is taken, then, to the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-morphism

$$(f)_\perp : (A)_\perp \to (B)_\perp \tag{3.13}$$

defined by $(f)_\perp(x) = f(x)$ in case $x$ is in the domain of definition of $f$, and $(f)_\perp(x) = \perp$ otherwise.

By abuse of notation, we also denote by $(-)_\perp : \mathsf{PSet} \to \mathsf{Cat}_{\omega\perp}$ the composition of the functors (3.12).

The category $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ can be seen as an op-Grothendieck construction of the indexed category $\mathbf{Vect}_\perp^{(-)}$, which is defined by

$$\mathbf{Vect}_\perp^{(-)} \stackrel{\text{def}}{=} \mathsf{Cat}_{\omega\perp}\left((-)_\perp, \mathbf{Vect}\right) : \mathsf{PSet}^{\mathrm{op}} \to \mathsf{Cat}. \tag{3.14}$$

*3.2.3* $\boldsymbol{\omega}\mathbf{Cpo}_\perp$*-enrichment.* $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ naturally inherits an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enrichment induced by the category of sets with partially defined functions. More precisely, we have that

$$(\mathtt{f}, \mathtt{f}') \leq (\mathrm{g}, \mathrm{g}')$$

in $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)((A, X), (B, Y))$ if the following conditions are satisfied:

- $\mathtt{domain}\,(f) \subseteq \mathtt{domain}\,(g)$;
- for each $a \in \mathtt{domain}\,(f)$, $\mathtt{f}(a) = \mathrm{g}(a)$ and $\mathtt{f}'_a = \mathrm{g}'_a$.

This standard ordering reflects the idea that $(\mathrm{g}, \mathrm{g}')$ extends $(\mathtt{f}, \mathtt{f}')$ while fully agreeing in the common domain.

Theorem 3.9. *With the above,* $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ *is an* $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$*-enriched category and, hence,* $\mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ *is an* $\omega$*-Freyd category.*

### 3.3 Partial CHAD-derivative as a morphism

Let $\mathrm{g} = (V, g) : M \to N$ be a morphism in $\mathfrak{P}\mathcal{M}an$, we define the morphism

$$\mathfrak{D}_{\omega}\mathrm{g} = \left(\mathrm{g}_0, \partial^{\omega}(\mathrm{g})\right) : \left(M, \mathcal{T}_M^*\right) \to \left(N, \mathcal{T}_N^*\right), \tag{3.15}$$

called the CHAD-derivative of g. We start by considering the (total) CHAD-derivative

$$\mathfrak{D}g : \left(V, \mathcal{T}_V^*\right) \to \left(N, \mathcal{T}_N^*\right)$$

of $g : V \to N$. We define, then, $\mathfrak{D}_{\omega}\mathrm{g}$ to be the canonical extension of $\mathfrak{D}g$ as follows:

- $\mathrm{g}_0$ is just the underlying partial (set) function of g;
- $\partial_a^{\omega}(\mathrm{g}) \coloneqq \partial_a(g)$ if $a \in V$, and $\partial_a^{\omega}(\mathrm{g}) : 0 \to \mathcal{T}_M^* a$ otherwise.

### 3.4 CHAD-derivative as a structure-preserving Freyd category morphism

The association

$$\mathfrak{D}_{\omega} : \mathfrak{P}\mathcal{M}an \to \mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \qquad M \mapsto \left(M, \mathcal{T}_M^*\right), \qquad \mathrm{g} \mapsto \mathfrak{D}_{\omega}\mathrm{g} \tag{3.16}$$

defines a functor. Indeed, while identity is clearly preserved, we have that $\mathfrak{D}_{\omega}\mathtt{h} \circ \mathfrak{D}_{\omega}\mathrm{g} = \mathfrak{D}_{\omega}(\mathtt{h} \circ \mathrm{g})$ for any pair of composable morphisms $\mathtt{h} : N \to K, \mathrm{g} : M \to N$ in $\mathcal{VM}an$, since

- $\mathtt{h} \circ \mathrm{g}$ is defined at $a \in M$ if and only if $\mathfrak{D}_{\omega}\mathtt{h} \circ \mathfrak{D}_{\omega}\mathrm{g}$ is defined at $a \in M$ if and only if $\mathfrak{D}_{\omega}(\mathtt{h} \circ \mathrm{g})$ is defined at $a \in M$;
- denoting $\mathfrak{D}_{\omega}\mathtt{h} \circ \mathfrak{D}_{\omega}\mathrm{g} = (s, s')$, we have that
  - by the chain rule, if $\mathtt{h} \circ \mathrm{g}$ is defined at $a \in M$, $s'_a = \partial_a^{\omega}(\mathtt{h} \circ \mathrm{g})$;
  - if $\mathtt{h} \circ \mathrm{g}$ is undefined at $a \in M$, $s'_a : 0 \to \mathcal{T}_M^* a$.

Moreover, by the above, it is straightforward to conclude that $\mathfrak{D}_{\omega}$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched functor, since it preserves the order, the colimits of $\omega$-chains, and the bottom morphisms. Therefore:

Theorem 3.10. (3.16) *is an* $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$*-enriched functor.*

Finally, since $\mathfrak{D}_{\omega}$ is an extension of $\mathfrak{D}$, the square (3.18) is commutative, making the pair

$${}_{cha}\mathfrak{D} \coloneqq (\mathfrak{D}, \mathfrak{D}_{\omega}) : \mathcal{FM}an \to \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \tag{3.17}$$

a Freyd category morphism.

$$\begin{array}{ccc} \mathfrak{P}\mathcal{M}an & \xrightarrow{\mathfrak{D}_{\omega}} & \mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \\ \uparrow{\scriptstyle \mathfrak{p}} & & \uparrow{\scriptstyle \mathfrak{j}} \\ \mathcal{VM}an & \xrightarrow[\mathfrak{D}]{} & \mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \end{array} \tag{3.18}$$

Theorem 3.11. *The CHAD-derivative* ${}_{cha}\mathfrak{D} = (\mathfrak{D}, \mathfrak{D}_{\omega})$ *defines an* $\omega$*-Freyd category morphism.*

### 3.5 What about iteration?

Coarse *iteration* constructs,[10] as understood here, allow for the implementation of algorithms with a dynamic stopping criteria. The precise syntax is given in Section 5, while the usual denotational semantics in terms of sets is given by the following: for each partially defined function

$$f : A \to B \sqcup A, \tag{3.19}$$

[10] This is the usual notion of iteration [8, 46], but free of the usual equational rules, *e.g.* [39].

we compute the partial function $\mathtt{it}\, f : A \to B$ which is obtained as follows: firstly, we consider the function $[\iota_B, f] : B \sqcup A \to B \sqcup A$, where $\iota_B : B \to B \sqcup A$ is the coprojection (coproduct inclusion). Secondly, given $a \in A$, we define $\mathtt{it}\, f(a) = b$ if there are $b \in B$ and $n \in \mathbb{N}$ such that $[\iota_B, f]^{(n)} \circ \iota_A(a) = b$; otherwise $\mathtt{it}\, f$ is not defined at $a$. This defines iteration in the category PSet of sets and partially defined functions.

More generally, we start by giving a general definition of a category with iteration, and then we establish this within the specific context of Freyd categories.

*Definition 3.12 (Coarse iteration).* An *iterative category* is a pair $(\mathcal{C}, \mathtt{it})$, where $\mathcal{C}$ is a category with finite coproducts and $\mathtt{it}$ is a family of morphisms called *context-free iteration.*

$$\left(\mathtt{it}^{(A,B)} : \mathcal{C}\,(A, B \sqcup A) \to \mathcal{C}\,(A, B)\right)_{(A,B)\in \mathrm{obj}\mathcal{C}\times\mathrm{obj}\mathcal{C}} \tag{3.20}$$

An *iterative category morphism* $\hat{F} : (\mathcal{C}, \mathtt{it}) \to (\mathcal{C}', \mathtt{it}')$ is a functor

$$\hat{F} : \mathcal{C} \to \mathcal{C}' \tag{3.21}$$

that strictly preserves finite products and iteration, meaning that, for each $(A, B) \in \mathrm{ob}\mathcal{C} \times \mathrm{ob}\mathcal{C}$ and each $g \in \mathcal{C}\,(A, B \sqcup A)$,

$$\mathtt{it}^{\left(\hat{F}(A),\hat{F}(B)\right)}\left(\hat{F}(g)\right) = \hat{F}\left(\mathtt{it}^{(A,B)} g\right), \tag{3.22}$$

which is usually simply denoted by

$$\mathtt{it}\left(\hat{F}(g)\right) = \hat{F}(\mathtt{it}\, g).$$

*Definition 3.13 (Basic fixed point equation).* An *iterative category* $(\mathcal{C}, \mathtt{it})$ satisfies the *basic fixed point equation* if the following equation is satisfied: for any morphism $f \in \mathcal{C}\,(A, B \sqcup A)$,

$$[\mathrm{id}_B, \mathtt{it}^{(A,B)} f] \circ f = \mathtt{it}^{(A,B)} f. \tag{3.23}$$

*Remark 3.14.* Iterative categories and iterative category morphisms form, of course, a category. Therefore, we have an induced notion of iterative category isomorphism.

*Remark 3.15.* While we emphasize the adoption of *coarse iteration* as in Definition 3.12, it is worth noting that additional equational rules could be imposed. This flexibility arises because the concrete denotational semantics of our languages inherently satisfy the usual equational laws: including specifically the basic fixed point equation, established in (3.23).

In particular, the fact that the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched (concrete) models for dependently typed languages with iteration adhere to the equation presented in (3.23) is crucial for establishing Theorem 4.36, which serves as the foundation for establishing a dependently typed language with iteration and, hence, deriving the syntactic (equational) iteration derivative in our target language. Consequently, readers have the choice to decide whether to require the iteration (in the syntax) to satisfy (3.23).

*Definition 3.16 (Iterative Freyd category).* An *iterative Freyd category* is a quintuple $(\mathcal{V}, \mathcal{C}, j, \otimes, \mathtt{it})$ where $(\mathcal{V}, \mathcal{C}, j, \otimes)$ is a Freyd category and $(\mathcal{C}, \mathtt{it})$ is an iterative category.

Furthermore, a Freyd category morphism $\left(F, \hat{F}\right) : (\mathcal{V}, \mathcal{C}, j, \otimes) \to (\mathcal{V}', \mathcal{C}', j', \otimes')$ is an *iterative Freyd category morphism* if $\hat{F} : (\mathcal{C}, \mathtt{it}) \to (\mathcal{C}', \mathtt{it}')$ is an iterative category morphism.

The iterative Freyd categories and iterative Freyd category morphisms form a category: the iterative Freyd category morphism composition is given by the pointwise composition.

*Definition 3.17 (Iteration with context).* Given an *iterative Freyd category* $(\mathcal{V}, C, j, \otimes, \mathtt{it})$, we define the family of functions

$$\left(\mathtt{it}\,_C^{(A,B)} : C\,(C \otimes A, B \sqcup A) \to C\,(C \otimes A, B)\right)_{(C,B,A)\in \mathrm{obj}C\times\mathrm{obj}C\times\mathrm{obj}C} \tag{3.24}$$

by the following.

For each $(C, B, A) \in \mathrm{ob}\mathcal{V} \times \mathrm{ob}\mathcal{V} \times \mathrm{ob}\mathcal{V}$ and each $g \in C\,(C \otimes A, B \sqcup A)$, denoting by

$$\pi_C : C \times A \to C, \pi_B : C \times B \to B, \mathrm{diag}_{C\times A} : C \times A \to (C \times A) \times (C \times A)$$

the respective projection morphisms and diagonal morphism in the cartesian category $\mathcal{V}$, we define

$$\mathtt{it}\,_C^{(A,B)} g \stackrel{\mathrm{def}}{=} \left(\mathtt{it}\,^{(A,B)}\left([\overline{\pi}_B, \mathrm{id}_{C\otimes A}] \circ \kappa \circ (\!|\pi_C, g|\!)\right)\right) \tag{3.25}$$

in which

$$\kappa : C \otimes (B \sqcup A) \xrightarrow{\cong} (C \otimes B) \sqcup (C \otimes A) \tag{3.26}$$

is the isomorphism induced by the the distribution of the action $\otimes : \mathcal{V} \times C \to C$ over coproducts, and

$$(\!|\pi_C, g|\!) \stackrel{\mathrm{def}}{=} (\pi_C \otimes g) \circ j\left(\mathrm{diag}_{C\times A}\right). \tag{3.27}$$

We call (3.24) the *context-sensitive iteration of* $(\mathcal{V}, C, j, \otimes, \mathtt{it})$.

*Remark 3.18.* It should be noted that an *iterative Freyd category morphism* $\left(F, \hat{F}\right) : (\mathcal{V}, C, j, \otimes) \to (\mathcal{V}', C', j', \otimes')$ preserves context-sensitive iteration; namely,

$$\mathtt{it}\,_{\hat{F}(C)}^{\left(\hat{F}(A),\hat{F}(B)\right)}\left(\hat{F}(g)\right) = \hat{F}\left(\mathtt{it}\,_Z^{(A,B)} g\right), \tag{3.28}$$

for any triple $(A, B, C) \in \mathrm{obj}C \times \mathrm{obj}C \times \mathrm{obj}C$.

Definition 3.17 shows how iterative Freyd categories provide us with a context-sensitive iteration out of context-free iterations.

### 3.6 Concrete models: enriched iteration

We can recover the setting of our main example, specializing to $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched categories and $\omega$-Freyd categories as follows. Let $C$ be an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched category with finite ($\boldsymbol{\omega}\mathbf{Cpo}_\perp$-)coproducts. Given a morphism $f \in C\,(A, B \sqcup A)$, we consider the *colimit*, denoted by $\underline{\mathtt{f}}$, of the following $\omega$-chain

$$[\mathrm{id}_B, \perp] \circ \mathbf{g} \le [\mathrm{id}_B, \perp] \circ \mathbf{g}^2 \le \cdots \le [\mathrm{id}_B, \perp] \circ \mathbf{g}^n \le \cdots \tag{3.29}$$

in $C\,(B \sqcup A, B)$, where

$$\mathbf{g} \coloneqq [\iota_Y, f] : B \sqcup A \to B \sqcup A. \tag{3.30}$$

We, finally, define the context-free iteration by

$$\mathtt{it}\,^{(A,B)} f \coloneqq \underline{\mathtt{f}} \circ \iota_A. \tag{3.31}$$

Theorem 3.19. *Every $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched category $C$ with finite $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts has a natural underlying iterative category $(C, \mathtt{it})$ once we define $\mathtt{it}$ by* (3.31). *Moreover, the iterative category $(C, \mathtt{it})$ satisfies the basic fixed equation.*

*Furthermore, if an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-functor $\hat{F} : C \to C'$ preserves finite $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts, then $\hat{F}$ induces an iterative category morphism*

$$(C, \mathtt{it}) \to (C', \mathtt{it}')$$

*between the underlying iterative categories.*

Proof. In the conditions above, $\hat{F}$ preserves iteration since $\hat{F}$ preserves finite $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts, $\perp$ and the colimits of $\omega$-chains of morphisms. □

*Definition 3.20 (Enriched iteration).* Let $C$ be an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched category with finite $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts. The iteration $\mathtt{it}$ of the underlying iterative category $(C, \mathtt{it})$, as established in Theorem 3.19, is called $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$*-enriched iteration.*

*Remark 3.21.* Since the categories $\mathsf{PSet}$ and $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$ are $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched, we have natural notions of iteration as defined in Theorem 3.19. Moreover the inclusion

$$(-)_{\perp} : \mathsf{PSet} \to \boldsymbol{\omega}\mathbf{Cpo}_{\perp},$$

which takes each set $A$ to the pointed $\omega$-cpo $(A)_{\perp}$ defined by freely adding a bottom element to the corresponding discrete $\omega$-cpo $A$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-functor and, hence, an iterative category morphism.

Corollary 3.22. *Every $\omega$-Freyd category $(\mathcal{V}, C, j, \otimes)$ has an underlying iterative Freyd category $(\mathcal{V}, C, j, \otimes, \mathtt{it})$ once we endow it with the iterative category $(C, \mathtt{it})$ defined in Theorem 3.19.*

*Furthermore, any $\omega$-Freyd category morphism $\left(F, \hat{F}\right) : (\mathcal{V}, C, j, \otimes) \to (\mathcal{V}', C', j', \otimes')$ is an iterative Freyd category morphism between the underlying iterative Freyd categories.*

Proof. It follows from Theorem 3.19. □

*Remark 3.23.* Whenever we refer to an $\omega$-Freyd category $(\mathcal{V}, C, j, \otimes)$, we consider the underlying iterative Freyd category $(\mathcal{V}, C, j, \otimes, \mathtt{it})$ defined above. Furthermore, it should be noted that, as in the case of any iterative Freyd category , we have the (context-sensitive) iteration as in Definition 3.17.

*3.6.1 Concrete iterative CHAD.* By extending the CHAD-derivative functor to a Freyd category morphism ${}_{cha}\mathfrak{D}$, we have established how the CHAD-derivative interacts with computations involving non-termination and non-differentiability in our setting. More precisely, we have shown that ${}_{cha}\mathfrak{D}$ preserves the domain structure we defined – specifically, the domain structure characterized by open sets. This preservation demonstrates that we can maintain our structure-preserving principle even in settings where totality does not hold.

Moreover, $\mathcal{F}\mathcal{M}an$ and $\mathcal{F}\mathbf{Fam}\,(\mathbf{Vect}^{\mathrm{op}})$ are $\omega$-Freyd categories by Theorem 3.7 and Theorem 3.9, we get that both are iterative Freyd categories by Theorem 3.22. The iteration in both iterative Freyd categories resemble that in $\mathsf{PSet}$ described above. More precisely, in the case of $\mathfrak{P}\mathbf{Fam}\,(\mathbf{Vect}^{\mathrm{op}})$, if

$$(V, f, f') : (A, X) \to (A \sqcup B, [X, Y])$$

is a morphism in $\mathfrak{P}\mathbf{Fam}\,(\mathbf{Vect}^{\mathrm{op}})$, we have that $\mathtt{it}\ (V, f, f') = (U, g, g')$ where $(U, g) = \mathtt{it}\,(V, f)$ in $\mathsf{PSet}$ and, for each $u \in U$ such that there exists an $n \in \mathbb{N}$ such that $f^{(m)}(u) \in A \subseteq A \sqcup B$ for all $m < n$ and $f^{(n)}(u) \in B \subseteq A \sqcup B$,

$$g'_u \coloneqq f'_u \circ f'_{f(u)} \circ \cdots \circ f'_{f^{(n)}(u)} : Y(g(u)) \to X(u).$$

Furthermore, since ${}_{cha}\mathfrak{D} = (\mathfrak{D}, \mathfrak{D}_{\omega})$ is an $\omega$-Freyd category morphism by Theorem 3.11, we get that the CHAD-derivative functor $\mathfrak{D}_{\omega}$ preserves iteration by Corollary 3.22. In other words, we have shown that we can maintain our structure-preserving principle even in iterative setting, with

$$\mathfrak{D}_{\omega}\,(\mathtt{it}\, f) = \mathtt{it}\,\mathfrak{D}_{\omega}\,(f)\,. \tag{3.32}$$

Theorem 3.24. ${}_{cha}\mathfrak{D} = (\mathfrak{D}, \mathfrak{D}_{\omega})$ *is an iterative Freyd category morphism.*

## 4 FIBRED ITERATION VIA PARAMETERIZED INITIAL ALGEBRAS

To implement iteration constructs in a dependently typed language that emulates the iterative category fragment $(\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}}), \mathtt{it})$ of our concrete model, we first establish an equational counterpart to the iteration construct. This counterpart provides us with a syntactic framework for realizing iteration within $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ by means of initial algebra semantics.

We begin by demonstrating that the underlying 2-functor of the suitable indexed category maps the iteration operator `it` in PSet to the parameterized initial algebra operator (or $\mu$-operator) in CAT. By exploiting this preservation property, we derive a purely equational characterization of iteration in $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, which in turn enables a syntactic implementation of the derivative of functions defined in terms of iteration.

This result shows that whenever the target language supports appropriate (indexed) inductive types (of substitution functors), our framework can directly implement iteration. In other words, once the iteration construct and the non-termination effect are incorporated into the cartesian types of the target language of [30], all the necessary requirements for iterative CHAD are met. Indeed, as detailed in Appendix B, our target language imposes even milder requirements.

Beyond solving the practical and theoretical problem of implementing the iterative CHAD, our observations about the target language also yield a *novel contribution* to *Programming Languages Theory*; namely, by introducing the notion of *iteration-extensive indexed category*, we provide a principled (and practical) way of incorporating the iteration construct into a dependently typed language in a coherent manner.

### 4.1 Parameterized initial algebras

There is an extensive literature on initial algebra semantics (e.g., [3, 5, 17, 21, 45]). For our specific context, we refer the reader to [30, Section 3], which provides an overview of the relevant background. However, the particular perspective we introduce here has not yet been sufficiently explored in the literature. We view *iteration* and *recursion* as instances of initial algebra semantics in suitable settings, a perspective presented in [32]. We briefly remark on this below but leave a detailed exploration for future work, *e.g.* [32].

Herein, we present a concise approach, by making the observations needed to frame our work related to CHAD. We start by recalling the basic definition of *parameterized initial algebra* in Cat.

*Definition 4.1 (The category of E-algebras).* Let $\mathcal{D}$ be a category and $E : \mathcal{D} \to \mathcal{D}$ be an endofunctor. The category of $E$-algebras, denoted by $E$-Alg, is defined as follows. The objects are pairs $(W, \zeta)$ in which $W \in \mathcal{D}$ and $\zeta : E(W) \to W$ is a morphism of $\mathcal{D}$. A morphism between $E$-algebras $(W, \zeta)$ and $(Y, \xi)$ is a morphism $g : W \to Y$ of $\mathcal{D}$ such that

$$\begin{array}{ccc} E(W) & \xrightarrow{E(g)} & E(Y) \\ \zeta \downarrow & & \downarrow \xi \\ W & \xrightarrow[g]{} & Y \end{array} \tag{4.1}$$

commutes. Assuming the existence of the initial $E$-algebra $\mathsf{in}_{(E)} : E(\mu(E)) \to \mu(E)$ (an initial object in $E$-Alg), we denote by

$$\mathtt{fold}\,(E, (Y, \xi)) : \mu(E) \to Y, \tag{4.2}$$

the unique morphism in $\mathcal{D}$ such that

$$\begin{array}{ccc} E\left(\mu(E)\right) & \xrightarrow{E(\texttt{fold}(E,(Y,\xi)))} & E(Y) \\ \Big\downarrow{\scriptstyle \mathsf{in}_{(E)}} & & \Big\downarrow{\scriptstyle \xi} \\ \mu(E) & \xrightarrow[\texttt{fold}(E,(Y,\xi))]{} & Y \end{array} \tag{4.3}$$

commutes. Whenever it is clear from the context, we denote $\texttt{fold}\,(E,(Y,\xi))$ by $\texttt{fold}\,(E,\xi)$.

Given a functor $H : \mathcal{D}' \times \mathcal{D} \to \mathcal{D}$ and an object $X$ of $\mathcal{D}'$, we denote by $H^X$ the endofunctor

$$H^X \stackrel{\text{def}}{=} H(X,-) : \mathcal{D} \to \mathcal{D}. \tag{4.4}$$

In this setting, if $\mu H^X$ exists for any object $X \in \mathcal{D}'$ then the universal properties of the initial algebras induce a functor denoted by $\mu(H,\mathcal{D}') : \mathcal{D}' \to \mathcal{D}$, called the parameterized initial algebra. In the following, we spell out how to construct parameterized initial algebras.

Proposition 4.2 ($\mu$-operator). *Let $H : \mathcal{D}' \times \mathcal{D} \to \mathcal{D}$ be a functor. Assume that, for each object $X \in \mathcal{D}'$, the functor $H^X = H(X,-)$ is such that $\mu\big(H^X,\mathcal{D}'\big)$ exists. In this setting, the association*

$$\begin{array}{rcl} \mu(H,\mathcal{D}') : \mathcal{D}' & \to & \mathcal{D} \\ X & \mapsto & \mu\Big(H^X\Big) \\ (f : X \to Y) & \mapsto & \texttt{fold}\left(H^X, \mathsf{in}_{(H^Y)} \circ H\left(f, \mu\Big(H^Y,\mathcal{D}\Big)\right)\right). \end{array}$$

*defines a functor, called the parameterized initial algebra.*

Proof. See, for instance, [30, Section 3]. □

Motivated by Proposition 4.2, given a functor $H : \mathcal{D}' \times \mathcal{D} \to \mathcal{D}$, we say that the parameterized initial algebra $\mu(H,\mathcal{D}')$ exists if $H^X$ has initial algebra for any $X \in \mathcal{D}'$.

*4.1.1 Fold for parameterized initial algebras.* It is particularly useful to notice that, given any functor $H : \mathcal{D}' \times \mathcal{D} \to \mathcal{D}$, an object $Y \in \mathcal{D}'$ and a $H^Y$-algebra $(W,\zeta)$, we want to consider the morphism $\texttt{fold}\,\big(H^Y,(W,\zeta)\big)$ In order to be concise, we introduce the following notation.

$$\texttt{fold}\,(H,Y,\zeta) \stackrel{\text{def}}{=} \texttt{fold}\,(H,Y,(W,\zeta)) \stackrel{\text{def}}{=} \texttt{fold}\left(H^Y,(W,\zeta)\right) : \mu(H,\mathcal{D}')\,(Y) \to W. \tag{4.5}$$

## 4.2 Parameterized initial algebras in terms of colimits

It is well-known that, in the presence of colimits of $\omega$-chains, we can compute the initial algebras of an endofunctor $E$ provided that $E$ preserves such colimits. We observe below how this applies for the case of parameterized initial algebras.

Lemma 4.3. *Let $\mathcal{D}$ be a a category with initial object $0$ and colimits of $\omega$-chains. If $H : \mathcal{D}' \times \mathcal{D} \to \mathcal{D}$ is a functor such that, for any $X \in \mathcal{D}'$, $H\,(X,-)$ preserves colimits of $\omega$-chains, then $\mu(H,\mathcal{D}')$ exists.*

*Furthermore, fixing $G \stackrel{\text{def}}{=} (\pi_{\mathcal{D}'}, H)$, denoting by $\bot : \mathcal{D}' \to \mathcal{D}$ the functor constantly equal to $0$ and $\underline{\iota}$ the only natural transformation*

$$\underline{\iota} : (\mathrm{id}_{\mathcal{D}'},\bot) \to \underline{H} \circ (\mathrm{id}_{\mathcal{D}'},\bot)\,,$$

*we can conclude that, if $\underline{H}$ is the colimit of the $\omega$-chain of functors*

$$(\mathrm{id}_{\mathcal{D}'},\bot) \xrightarrow{\underline{\iota}} G \circ (\mathrm{id}_{\mathcal{D}'},\bot) \xrightarrow{G(\underline{\iota})} G^2 \circ (\mathrm{id}_{\mathcal{D}'},\bot) \xrightarrow{G^2(\underline{\iota})} \cdots \xrightarrow{G^n(\underline{\iota})} G^n \circ (\mathrm{id}_{\mathcal{D}'},\bot) \xrightarrow{G^{n+1}(\underline{\iota})} \cdots \tag{4.6}$$

*in the functor category* $\mathsf{Cat}[\mathcal{D}', \mathcal{D}' \times \mathcal{D}]$, *then*

$$\mu(H, \mathcal{D}') \cong \pi_{\mathcal{D}} \circ \underline{H}. \tag{4.7}$$

Proof. Pointwise, for each $X \in \mathcal{D}'$, we have that, indeed, $\pi_{\mathcal{D}} \circ \underline{H}(X)$ is the colimit of the diagram

$$0 \xrightarrow{\iota} H^X(0) \xrightarrow{H^X(\iota)} \left(H^X\right)^2(0) \xrightarrow{(H^X)^2(\iota)} \cdots \xrightarrow{(H^X)^n(\iota)} \left(H^X\right)^n(0) \xrightarrow{(H^X)^{n+1}(\iota)} \cdots \tag{4.8}$$

which is $\mu(H^X)$ by Adámek's theorem. □

## 4.3 Initial algebras of substitution functors

Conventionally, (parameterized) initial algebras are studied for functors built from standard type formers such as products, coproducts, and exponentials, yielding semantics for various inductive types. However, in the presence of dependent types, we gain a new class of syntactically definable functors arising from term substitution in types, often called *change-of-base functors.*

To begin, observe that any endomorphism $f : A \to A$ in $C$ induces a functor

$$\mathcal{L}(f) : \mathcal{L}(A) \to \mathcal{L}(A).$$

In most cases, the initial $\mathcal{L}(f)$-algebra is trivial; namely:

Lemma 4.4. *Let* $\mathcal{L} : C^{op} \to \mathsf{Cat}$ *be an indexed category equipped with indexed initial objects. Then for any* $f : A \to A$ *in* $C$, *the initial* $\mathcal{L}(f)$*-algebra is given by*

$$(0, \mathrm{id} : 0 \to 0).$$

On the other hand, when working in dependent type theory with sum types, we typically model it by an indexed category $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ that satisfies the extensivity property with respect to coproducts (a notion and perspective introduced in [30, Definition 37]). In this setting, a broader class of substitution functors arises. Specifically, each morphism $f : A \to B \sqcup A$ induces a functor

$$\mathcal{L}(B) \times \mathcal{L}(A) \xrightarrow{\cong} \mathcal{L}(B \sqcup A) \xrightarrow{\mathcal{L}(f)} \mathcal{L}(A), \tag{4.9}$$

whose parameterized initial algebras

$$\mathcal{L}(B) \longrightarrow \mathcal{L}(A)$$

are non-trivial and play a central role in understanding iteration in the dependent typed setting.

Motivated by the concrete example given in 4.4, we analyze in more detail these parameterized initial algebras for the substitution functors of the form (4.9), and their foundational connection with iteration, in 4.6.

## 4.4 Parameterized initial algebras of substitution functors

To lay the foundation for our observations, we again start by revisiting the elementary denotational semantics of iteration in terms of sets and partially defined functions between them, as in Section 3.5. The category PSet is naturally endowed with a two-dimensional structure – that is, the structure coming from the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enrichment given by the domain order between functions. As a consequence, for any partially defined function $f : A \to B \sqcup A$ and any set $K$, we have an induced functor

$$\underline{\mathsf{PSet}\,(f, K)} \stackrel{\text{def}}{=} \mathsf{PSet}\,(f, K) \circ \varsigma : \mathsf{PSet}\,(B, K) \times \mathsf{PSet}\,(A, K) \to \mathsf{PSet}\,(A, K) \tag{4.10}$$

where $\varsigma$ is the isomorphism

$$\mathsf{PSet}\,(B, K) \times \mathsf{PSet}\,(A, K) \cong \mathsf{PSet}\,(B \sqcup A, K)\,, \qquad (t, r) \mapsto [t, r] \tag{4.11}$$

induced by the universal property of the coproduct $B \sqcup A$. We claim that

$$\mathsf{PSet}\,(\mathtt{it}\, f, K) = \mu\Big(\underline{\mathsf{PSet}\,(f,K)}, \mathsf{PSet}\,(B,K)\Big) \tag{4.12}$$

where $\mathtt{it}\, f$ is the function defined in (3.5).

The Equation (4.12) makes the indexed category

$$\mathsf{PSet}\,(-,K) : \mathsf{PSet}^{\mathrm{op}} \to CAT \tag{4.13}$$

into an *iteration-extensive indexed category*, as introduced in 4.6 below.

## 4.5 Revisiting extensivity

In 4.6, we introduce a principled notion of iteration on the Grothendieck construction for a model of our target language. To do so, we first require that this op-Grothendieck construction admits finite coproducts. Below, we recall how finite-coproduct-extensive indexed categories help us model sum types in our setting – namely, how they ensure the existence of finite coproducts in the op-Grothendieck construction.

In [30, Definition 31], we introduced the notion of a *binary-coproduct-extensive indexed category*

$$\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT},$$

which was referred to simply as an *extensive indexed category* in that work. Motivated by this concept, we further developed the notion of extensive indexed categories with respect to a class of diagrams in [31, Section 2.3]; we recall this definition below.

*Definition 4.5 (Extensive indexed categories).* Let $\mathfrak{S}$ be a class of diagrams. We say that an indexed category $\mathcal{L} : C \to \mathsf{CAT}$ is $\mathfrak{S}$-colimit-extensive if $C$ has, and $\mathcal{L}$ preserves, the limits of diagrams $\mathfrak{S} \to C^{\mathrm{op}}$ of $C^{\mathrm{op}}$. For instance:

- we say that the indexed category $\mathcal{L} : C \to \mathsf{CAT}$ is *binary-coproduct-extensive* if $C$ has binary coproducts and $\mathcal{L}$ preserves binary products of $C^{\mathrm{op}}$;
- we say that the indexed category $\mathcal{L} : C \to \mathsf{CAT}$ is *finite-coproduct-extensive* if $C$ has finite coproducts and $\mathcal{L}$ preserves binary products of $C^{\mathrm{op}}$.

*Unlike in [30], throughout this work, we use the term "extensive indexed category" to mean what is more precisely a "finite-coproduct-extensive indexed category" as defined above.*

Clearly, every finite-coproduct-extensive indexed category is a binary-coproduct-extensive indexed category. Moreover, we recall that, by [30, Lemma 33], we have the following result:

LEMMA 4.6. *Let $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ be an indexed category such that $C$ has finite coproducts. We have that $\mathcal{L}$ is binary-coproduct-extensive and $\mathcal{L}(W)$ is non-empty for some $W \in C$ if, and only if, $\mathcal{L}$ is a finite-coproduct-extensive indexed category.*

PROOF. By [30, Lemma 33], if $\mathcal{L}$ is a binary-coproduct-extensive indexed category such that $\mathcal{L}(W)$ is non-empty for some $W \in C$, then we get that $\mathcal{L}$ preserves the terminal object of $C^{\mathrm{op}}$; that is to say, $\mathcal{L}(0) \cong 1$. Therefore $\mathcal{L}$ is finite-coproduct-extensive.

Reciprocally, if $\mathcal{L}$ is a finite-coproduct-extensive indexed category, then it is binary-coproduct-extensive, and $\mathcal{L}(0) \cong 1$ is the terminal category and, hence, non-empty. □

*Remark 4.7.* In other words, Lemma 4.6 states that, given any category $C$ with finite coproducts, the only binary-coproduct-extensive indexed category that is not finite-coproduct-extensive indexed is the indexed category $\emptyset : C^{\mathrm{op}} \to \mathsf{CAT}$ constantly equal to the empty category $\emptyset$.

We observe that the notion of a Σ-bimodel for sum types, as introduced in [30, Definition 37], coincides with that of a finite-coproduct-extensive indexed category. In the following, we clarify this relationship by first recalling the definition of a Σ-bimodel for sum types.

*Definition 4.8.* Let $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ be a binary-coproduct-extensive indexed category. We say that $\mathcal{L}$ is a Σ-bimodel for sum types if $C$ has initial object $0$ and $\mathcal{L}(0)$ has terminal and initial objects.

Lemma 4.9. *Let $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ be an indexed category. $\mathcal{L}$ is a Σ-bimodel for sum types if, and only if, it is a finite-coproduct-extensive indexed category.*

Proof. If $\mathcal{L}$ is a Σ-bimodel for sum types, we get that $C$ has finite coproducts and $\mathcal{L}(0)$ is not empty (since it has a terminal object). Therefore, $\mathcal{L}$ is a finite-coproduct-extensive indexed category by Lemma 4.6 .

Reciprocally, if $\mathcal{L}$ is a finite-coproduct-extensive indexed category, then $\mathcal{L}(0) \cong 1$ has a zero object (hence terminal and initial objects) since it is the terminal category. □

The appeal of these extensive notions lies in the properties they induce in the (op-)Grothendieck constructions, as demonstrated in our previous work [30, 31]. To keep this discussion brief herein, we highlight only the following result:

Theorem 4.10. *Let $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ be a finite-coproduct-extensive indexed category. In this $\Sigma_C \mathcal{L}^{\mathrm{op}}$ has finite coproducts. More precisely:*

- *the initial object is given by $(0, 1)$, where $1$ is the only object of $\mathcal{L}(0) \cong 1$;*
- *the binary coproduct is given by*

$$(B, Y) \sqcup (A, X) \cong \left(B \sqcup A, \varsigma^{(B,A)}(Y, X)\right)$$

*where $\varsigma^{(B,A)} : \mathcal{L}(B) \times \mathcal{L}(A) \xrightarrow{\cong} \mathcal{L}(B \sqcup A)$ is the inverse of the comparison morphism.*

Proof. From the results explored in [30, 31], we get that, indeed, the initial object of $\Sigma_C \mathcal{L}^{\mathrm{op}}$ is given by the pair $(0, 1)$ since $1$ is the terminal object of $\mathcal{L}(0) \cong 1$.

The description of the binary coproduct is, for instance, given by [30, Corollary 36]. □

We now introduce iteration into the op-Grothendieck construction of an extensive indexed category. As we explain in detail below, by strengthening our notion of extensivity, we can get this construction naturally, and ensure it is fibred. Moreover, under suitable hypothesis, it is the unique fibred iteration satisfying the fixed point equation.

### 4.6 Iteration-extensive indexed categories

We introduce below the notion of *iteration-extensive indexed category* (Definition 4.11), one of the novel contributions of the present paper.

This concept aims to provide the basic principle to answer the following general question: "how one can incorporate iteration in a dependently typed language in a principled, practical and coherent manner?" We focus, however, in the question at hand: which has to do with implementing the CHAD derivative for programs involving iteration.

We show that the concrete semantics of the target language satisfies the property we call *iteration extensivity.* This property allows us to define a fibred iteration at $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ purely in equational terms through its corresponding indexed category $\mathbf{Vect}_{\perp}^{(-)} : \mathsf{PSet}^{\mathrm{op}} \to \mathsf{CAT}$ (as defined in Eq. 3.14), without relying on the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-structure. This will give us a basic principle to define the syntax of our target language, and the implementation of our program transformation.

Motivated by Eq. (4.12) and the notions of extensive indexed categories, we introduce the following notion – which is our proposed guiding principle to introduce iteration in dependently typed languages.

*Definition 4.11 (Iteration-extensivity).* An *iteration-extensive indexed category* is a triple $(C, \mathtt{it}, \mathcal{L})$ where $(C, \mathtt{it})$ is an iterative category, $\mathcal{L} : C^{op} \to \mathsf{CAT}$ is an extensive indexed category, and, for any morphism $f \in C(A, B \sqcup A)$,

$$\mathcal{L}\left(\mathtt{it}^{(A,B)} f\right) \cong \mu\left(\mathcal{L}(f) \circ \varsigma^{(A,B)}, \mathcal{L}(B)\right) \tag{4.14}$$

where $\varsigma^{(A,B)} : \mathcal{L}(A) \times \mathcal{L}(B) \xrightarrow{\cong} \mathcal{L}(A \sqcup B)$ is the inverse of the comparison morphism.

When the iterative category $(C, \mathtt{it})$ is understood from the context, we omit it from the notation and simply say that the indexed category $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ is *iteration-extensive* (or *extensive for iteration*).

As mentioned above, for any object $K \in \mathsf{PSet}$, Eq. (4.12) shows that $\mathsf{PSet}(-, K) : \mathsf{PSet}^{\mathrm{op}} \to \mathsf{CAT}$ is iteration-extensive w.r.t. the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched iteration in $\mathsf{PSet}$. More generally, by abuse of notation, given any $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-category, we can consider the indexed category

$$C(-, K) : C^{\mathrm{op}} \to \mathsf{CAT} \tag{4.15}$$

where $C(A, K)$ is the $\omega$-complete partially ordered set of morphisms $A \to K$ in $C$. In other words, $C(-, K)$ is actually obtained from composing the arrows

$$C^{\mathrm{op}} \xrightarrow{C[-,K]} \boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{CAT}, \tag{4.16}$$

where $C[-, K] : C^{\mathrm{op}} \to \boldsymbol{\omega}\mathbf{Cpo}_\perp$ is the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched representable functor, and $\boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{CAT}$ is the inclusion.

Theorem 4.12 (Iteration as a parameterized initial algebra). *Let $C$ be an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-category with finite $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts. We consider the underlying iterative category $(C, \mathtt{it})$ as defined in Theorem 3.19. For any object $K$ of $C$, the triple $(C, \mathtt{it}, C(-, K))$ is an iteration-extensive indexed category.*

*Moreover, for each pair $(A, B) \in \mathrm{obj}C \times \mathrm{obj}C$ and each morphism $f \in C(A, B \sqcup A)$,*

$$\mathtt{it}^{(A,B)} f : A \to B \tag{4.17}$$

*is the **only** morphism such that*

$$C(\mathtt{it}^{(A,B)} f, K) = \mu\left(C(f, K) \circ \varsigma^{(A,B)}, C(B, K)\right) \tag{4.18}$$

*for any object $K$ in $C$.*

Proof. It follows from Adámek's theorem on the construction of initial algebras via the colimit of $\omega$-chains. More precisely, for each $K, A, B \in C$, the enriched Yoneda embedding induces a colimit-preserving functor $C(-, K)_{B,A} : C(B \sqcup A, A) \to \mathsf{Cat}(C(B \sqcup A, K), C(A, K))$. We denote

$$\underline{C(-, K)_{B \sqcup A, A}} \stackrel{\text{def}}{=} \mathsf{Cat}(\varsigma, C(A, K)) \circ C(-, K)_{B,A} : C(B \sqcup A, A) \to \mathsf{Cat}(C(B, K) \times C(A, K), C(A, K)). \tag{4.19}$$

Given $f \in C(A, B \sqcup A)$, by denoting

$$H \stackrel{\text{def}}{=} \underline{C(f, K)}, \quad g \stackrel{\text{def}}{=} [\iota_B, f], \quad G \stackrel{\text{def}}{=} \left(\pi_{C(A,K)}, \underline{C(f, K)}\right) \tag{4.20}$$

we have that

$$\underline{C([\iota_B, f], K)} = \underline{C(g, K)} = G = \left(\pi_{C(A,K)}, \underline{C(f, K)}\right), \quad \underline{C([\mathrm{id}_B, \perp], K)} = \left(\mathrm{id}_{C(B,K)}, \perp\right) \tag{4.21}$$

This shows that the functor $\underline{C(-,K)_{B,A}}$ takes the Diagram (3.29) to

$$\left(\mathrm{id}_{C(A,K)},\bot\right)\xrightarrow{\iota} G\circ\left(\mathrm{id}_{C(A,K)},\bot\right)\xrightarrow{G(\underline{\iota})} G^2\circ\left(\mathrm{id}_{C(A,K)},\bot\right)\xrightarrow{G^2(\underline{\iota})}\cdots\xrightarrow{G^n(\underline{\iota})} G^n\circ\left(\mathrm{id}_{C(A,K)},\bot\right)\xrightarrow{G^{n+1}(\underline{\iota})}\cdots \tag{4.22}$$

Therefore, since $\underline{C(-,K)_{B,A}}$ is colimit-preserving, denoting by $\underline{H}$ is the colimit of (4.22) and by $\underline{\mathtt{f}}$ the colimit of (3.29), we have that $\underline{H}\cong H(\underline{\mathtt{f}})$, and, hence:

$$\begin{aligned}\underline{C\left(\mathtt{it}^{(A,B)}f,K\right)} &= \underline{C\left(\underline{\mathtt{f}}\circ\iota_W,K\right)} && (4.23)\\ &= \underline{C\left(\iota_A,K\right)}\circ\underline{C\left(\underline{\mathtt{f}},K\right)} && (4.24)\\ &\cong \pi_{C(A,K)}\circ\underline{H} && (4.25)\end{aligned}$$

which, by Lemma 4.3, is isomorphic to $\mu(H, C(B,K))$.

The converse of the Theorem follows from the existence of iteration in $C$ and Yoneda Lemma. □

Actually, in the case of PSet, we have even a stronger result:

THEOREM 4.13. *We consider the iterative category* $(\mathsf{PSet},\mathtt{it})$ *as defined in Remark 3.21, and the inclusion*

$$(-)_\bot : \mathsf{PSet}\to\mathsf{Cat}_{\omega\bot} \tag{4.26}$$

*defined in* (3.12). *For any object* $K$ *in* $\mathsf{Cat}_{\omega\bot}$, *the triple*

$$(\mathsf{PSet},\mathtt{it},\mathsf{Cat}_{\omega\bot}((-)_\bot,K):\mathsf{PSet}^{\mathrm{op}}\to\mathsf{CAT}) \tag{4.27}$$

*is an iteration-extensive indexed category. As a consequence, taking* $K=\mathbf{Vect}$, *the triple*

$$\left(\mathsf{PSet},\mathtt{it},\mathbf{Vect}_\bot^{(-)}=\mathsf{Cat}_{\omega\bot}((-)_\bot,\mathbf{Vect}):\mathsf{PSet}^{\mathrm{op}}\to\mathsf{CAT}\right) \tag{4.28}$$

*is an iteration-extensive indexed category.*

*More precisely, by denoting* $\varsigma^{(B,A)}$ *(or just* $\varsigma$*) the appropriate isomorphisms induced by the coproducts universal properties, for any morphism* $f:A\to B\sqcup A$ *of* PSet,

$$\mathtt{it}^{(A,B)}f : A\to B \tag{4.29}$$

*is the only morphism such that, for any object* $K$ *in* $\mathsf{Cat}_{\omega\bot}$,

$$\mathsf{Cat}_{\omega\bot}\left(\left(\mathtt{it}^{(A,B)}f\right)_\bot,K\right)\cong\mu\left(\mathsf{Cat}_{\omega\bot}((f)_\bot,K)\circ\varsigma^{(B,A)},\mathsf{Cat}_{\omega\bot}((B)_\bot,K)\right) \tag{4.30}$$

*which is a functor* $\mathsf{Cat}_{\omega\bot}((B)_\bot,K)\to\mathsf{Cat}_{\omega\bot}((A)_\bot,K)$. *As a consequence, taking* $K=\mathbf{Vect}$,

$$\mathsf{Cat}_{\omega\bot}\left(\left(\mathtt{it}^{(A,B)}f\right)_\bot,\mathbf{Vect}\right)=\mathbf{Vect}_\bot^{\left(\mathtt{it}^{(A,B)}f\right)}\cong\mu\left(\mathbf{Vect}_\bot^{(f)}\circ\varsigma^{(B,A)},\mathbf{Vect}_\bot^{B}\right)=\mu\left(\mathcal{H}_{(f)_\bot},\mathbf{Vect}_\bot^{(B)}\right), \tag{4.31}$$

*in which* $\mathcal{H}_{(f)_\bot}\overset{\mathrm{def}}{=}\mathbf{Vect}_\bot^{(f)}\circ\varsigma^{(B,A)}:\mathbf{Vect}_\bot^{(B)}\times\mathbf{Vect}_\bot^{(A)}\to\mathbf{Vect}_\bot^{(A)}$ *is defined by*

$$(Y,X)\mapsto[Y,X]\circ(f)_\bot \tag{4.32}$$

$$(\alpha:Y\to Y',\beta:X\to X')\mapsto[\alpha,\beta]\circ(f)_\bot:[Y,X]\circ(f)_\bot\to[Y',X']\circ(f)_\bot \tag{4.33}$$

*where* $[Y,X]:(B\sqcup A)_\bot(B)_\bot\sqcup(A)_\bot\to\mathbf{Vect}$ *is the functor induced by the universal property of the coproduct in* $\mathsf{Cat}_{\omega\bot}$.

Proof. For each $K \in \mathsf{Cat}_{\omega\perp}$ and each $A, B \in C$, by observing that $(-)_\perp$ is a full (locally isomorphism) and coproduct-preserving functor, we have that

$$\mathsf{Cat}_{\omega\perp}\left((-)_\perp, K\right)_{B\sqcup A, A} : \mathsf{Cat}_{\omega\perp}\left((B\sqcup A)_\perp, (A)_\perp\right) \to \mathsf{Cat}\left(\mathsf{Cat}_{\omega\perp}\left((B\sqcup A)_\perp, K\right), \mathsf{Cat}_{\omega\perp}\left((A)_\perp, K\right)\right)$$

is a colimit-preserving functor. Since $(B \sqcup A)_\perp = (B)_\perp \sqcup (A)_\perp$, the argument follows precisely the same lines as Theorem 4.12's proof. □

*Remark 4.14.* Although we do not explicitly need this fact here, the interested reader might notice that, up to isomorphism, there is a natural definition of iteration in $\mathsf{Cat}_{\omega\perp}$, which is defined by the usual colimit of $\omega$-chains. This makes $\mathsf{Cat}_{\omega\perp}$ into a (pseudo)iterative category.

Theorem 4.13 follows from Yoneda Lemma and the fact that the full inclusions $\mathsf{PSet} \xrightarrow{(-)_\perp} \boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{Cat}_{\omega\perp}$ are (up to isomorphism) iterative category morphisms between iterative categories (with the usual notions of iteration, and the weak notion in $\mathsf{Cat}_{\omega\perp}$) since these inclusions preserve coproducts and preserve colimits of $\omega$-chains of morphisms.

*4.6.1 Iteration-extensive Freyd indexed categories.* We can finally define what is an iteration-extensive Freyd indexed category. We can think of them as models of dependently typed languages with non-termination and iteration constructs.

*Definition 4.15 (Iteration-extensive Freyd indexed category).* An iteration-extensive Freyd indexed category is a 6-tuple $(\mathcal{V}, C, j, \otimes, \mathtt{it}, \mathcal{L})$ such that $(C, \mathtt{it}, \mathcal{L})$ is an iteration-extensive indexed category in which $\mathcal{L}$ has indexed finite coproducts, and $(\mathcal{V}, C, j, \otimes)$ is a distributive Freyd category.

As a consequence of Lemma 4.19 below, we see that iteration-extensive Freyd indexed categories are iterative Freyd categories. We postpone a detailed discussion of this fact to Sec. 4.11.

### 4.7 The fibred iteration induced by an iteration-extensive indexed category

An *iteration-extensive indexed category* induces an iteration on its (op-)Grothendieck construction: specifically, the op-Grothendieck construction of such an indexed category naturally becomes an iterative category when equipped with the *container iteration* as established in Def. 4.17 and Lemma 4.19 below.

In what follows, we show that, in the setting of iterations satisfying the basic fixed point equation (see Definition 3.17), the container iteration is essentially the *unique* fibred iteration for the op-Grothendieck construction of an iteration-extensive indexed category.

More precisely, given an iteration-extensive indexed category $(C, \mathtt{it}, \mathcal{L})$, the container iteration from Def.4.17 is the unique fibred iteration that satisfies the basic fixed point equation, provided that the base iterative category $(C, \mathtt{it})$ also satisfies it, that is to say, the container iteration fulfills the universal property presented in Theorem 4.20.

We start by defining what we mean by *fibred iteration.*

*Definition 4.16 (Fibred iteration).* Let $(C, \mathtt{it})$ be an iterative category, and $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ a finite-coproduct-extensive indexed category. If $\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \underline{\mathtt{it}}\right)$ is an iterative category such that the associated op-fibration

$$P_{\mathcal{L}} : \Sigma_C \mathcal{L}^{\mathrm{op}} \to C, \tag{4.34}$$

defined by $P_{\mathcal{L}}\left(f, f'\right) = f$, yields to an iterative category morphism

$$\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \underline{\mathtt{it}}\right) \to (C, \mathtt{it}), \tag{4.35}$$

we say that $\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \underline{\mathtt{it}}\right)$ is a *fibred iterative category* over $(C, \mathtt{it})$. In the setting above, $\underline{\mathtt{it}}$ is a *fibred iteration* for the triple $(C, \mathtt{it}, \mathcal{L})$.

*Definition 4.17 (Container iteration).* Let $(C, \mathtt{it}, \mathcal{L})$ be an iteration-extensive indexed category. For each $(A, X), (B, Y) \in \Sigma_C \mathcal{L}^{\mathrm{op}}$ and each $(f, f') \in C(A, B \sqcup A)$, we define

$$\hat{\mathtt{it}}^{((A,X),(B,Y))}(f, f') \stackrel{\text{def}}{=} \left(\mathtt{it}^{(A,B)} f, \mathtt{fold}\left(\left(\mathcal{L}(f) \circ \varsigma^{(B,A)}\right), Y, (X, f')\right)\right). \tag{4.36}$$

The iteration $\hat{\mathtt{it}}$ is called the *container iteration* induced by the iteration-extensive indexed category $(C, \mathtt{it}, \mathcal{L})$.

*Remark 4.18.* It should be noted that, in the context of Def. 4.17, $f'$ indeed defines an $\mathcal{L}(f) \circ \varsigma^{(B,A)}(Y, -)$-algebra structure on $X$; namely,

$$f' : \mathcal{L}(f) \circ \varsigma^{(B,A)}(Y, X) \to X.$$

Therefore Equation (4.36) establishes a well defined iterative category.

We establish below that the container iteration $\hat{\mathtt{it}}$ for an iteration-extensive indexed category gives rise to an iterative category $\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right)$.

Lemma 4.19 (Container iterations are iterations). *Every iteration-extensive indexed category $(C, \mathtt{it}, \mathcal{L})$ gives rise to an iterative category*

$$\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right), \tag{4.37}$$

*where $\hat{\mathtt{it}}$ is the container iteration as in Definition 4.17. Moreover, the container iteration $\hat{\mathtt{it}}$ is a fibred iteration for $(C, \mathtt{it}, \mathcal{L})$, in the sense of Def. 4.16.*

Proof. The proof is only about type checking, as it amounts to showing that we indeed end up with a coarse iteration. ◻

In the setting of Lemma 4.19, if $(C, \mathtt{it})$ satisfies the basic fixed point equation, (4.37) is the unique fibred iterative category satisfying the basic fixed point equation (see Definition 3.17). More precisely:

Theorem 4.20 (Universal property of the container iteration). *Let $(C, \mathtt{it}, \mathcal{L})$ be an iteration-extensive indexed category such that $(C, \mathtt{it})$ satisfies the basic fixed point equation. The container iteration $\hat{\mathtt{it}}$ is the only fibred iteration for $(C, \mathtt{it}, \mathcal{L})$ such that $\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right)$ also satisfies the basic fixed point equation.*

*Explicitly, we have the following. If $\tilde{\mathtt{it}}$ is a fibred iteration for an iteration-extensive indexed category $(C, \mathtt{it}, \mathcal{L})$ and $\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \tilde{\mathtt{it}}\right)$ satisfies the basic fixed point equation, then $\tilde{\mathtt{it}}$ is the container iteration as in Def. 4.16, in other words, the identity functor on $\Sigma_C \mathcal{L}^{\mathrm{op}}$ induces an iterative category isomorphism*

$$\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \tilde{\mathtt{it}}\right) \to \left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right). \tag{4.38}$$

Proof. The result follows from the universal property of the (parameterized) initial algebra. More precisely, the pair

$$\left(X, f' : \mathcal{L}(f) \circ \varsigma^{(B,A)}(Y, X) \to X\right) \tag{4.39}$$

is an $\mathcal{L}(f) \circ \varsigma^{(B,A)}(Y, -)$-algebra, and, hence, $\mathtt{fold}\left(\left(\mathcal{L}(f) \circ \varsigma^{(B,A)}\right), Y, f'\right)$ is the only morphism such that Diagram (4.40) commutes.

$$\begin{array}{ccc} \mathcal{L}(f) \circ \varsigma^{(B,A)}\left(Y, \mathcal{L}\left(\mathtt{it}^{(A,B)} f\right)(Y)\right) & \xrightarrow{\mathcal{L}(f)\circ\varsigma^{(B,A)}\left(Y, \mathtt{fold}\left(\mathcal{L}(f)\circ\varsigma^{(B,A)}, Y, f'\right)\right)} & \mathcal{L}(f) \circ \varsigma^{(B,A)}(Y, X) \\ \Big\downarrow {\scriptstyle \mathsf{in}_{\left(\mathcal{L}(f)\circ\varsigma^{(B,A)}(Y,-)\right)}} & & \Big\downarrow {\scriptstyle f'} \\ \mathcal{L}\left(\mathtt{it}^{(A,B)} f\right)(Y) & \xrightarrow[\mathtt{fold}\left(\mathcal{L}(f)\circ\varsigma^{(B,A)}, Y, f'\right)]{} & X \end{array} \tag{4.40}$$

By observing that $\mathsf{in}_{\left(\mathcal{L}(f)\circ\varsigma^{(B,A)}(Y,-)\right)}$ is the identity (when the indexed category is strict), we conclude that the commutativity of Diagram (4.40) actually means that

$$[\mathrm{id}_{(B,Y)}, \hat{\mathtt{it}}^{((A,X),(B,Y))}(f, f')] \circ (f, f') = \hat{\mathtt{it}}^{((A,X),(B,Y))}(f, f'). \tag{4.41}$$

holds, and, moreover, that $\mathtt{fold}\left(\mathcal{H}^{Y}_{(f)_\perp}, f'\right)$ is indeed the only second component making (4.41) hold. ◻

*Definition 4.21.* In the context of Theorem 4.20, $\left(\Sigma_{C}\mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right)$ is the *op-Grothendieck construction of the iteration-extensive indexed category* $(C, \mathtt{it}, \mathcal{L})$.

Theorem 4.22 (Grothendieck construction of iteration-extensive Freyd indexed categories). *The op-Grothendieck construction takes each iteration-extensive Freyd indexed category* $(\mathcal{V}, C, j, \otimes, \mathtt{it}, \mathcal{L})$ *to a iterative Freyd category*

$$\left(\Sigma_{\mathcal{V}}(\mathcal{L} \circ j)^{\mathrm{op}}, \Sigma_{C}\mathcal{L}^{\mathrm{op}}, \hat{j}, \hat{\otimes}, \hat{\mathtt{it}}\right) \tag{4.42}$$

*where* $\hat{j}(A, X) \stackrel{\mathrm{def}}{=} (j(A), X)$,

$$(f : A \to B, f')\, \hat{\otimes}\, (g : C \to D, g') = (f \otimes g, \mathcal{L}(\pi_A)(f') \sqcup \mathcal{L}(\pi_C)(g')), \tag{4.43}$$

*and* $\left(\Sigma_{C}\mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right)$ *is the op-Grothendieck construction of the iteration-extensive indexed category* $(C, \mathtt{it}, \mathcal{L})$.

Proof. $\Sigma_{\mathcal{V}}(\mathcal{L} \circ j^{\mathrm{op}})^{\mathrm{op}}$ is indeed a distributive category by [30, Proposition 18], [30, Corollary 36] and [30, Theorem 39]. The fact that $\hat{\otimes}$ satisfies the conditions of Def. 3.1 follows from the construction of products in $\Sigma_{\mathcal{V}}(\mathcal{L} \circ j^{\mathrm{op}})^{\mathrm{op}}$. ◻

In Theorem 4.36, we prove that the container iteration introduced above coincides with the usual iteration in the case of the **concrete models for dependently typed languages with iteration**, or, more precisely, $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched concrete models, like the case of the *concrete semantics of our target language*. This result is foundational to our contribution, as it shows that our container iteration indeed implements the expected behavior inherited from the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enrichment of the concrete semantics – while not depending on that enrichment itself, given that the categorical semantics (and the syntax) is not $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched.

## 4.8 Concrete container iterations are container iterations

In what follows, we show that our definition of container iteration agrees with the usual notion of iteration via colimits in concrete (i.e., $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched) models. More precisely, we demonstrate that in the setting of $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed categories, the concrete iteration – defined as colimits of $\omega$-chains – satisfies the property stated in Theorem 4.20; namely, it is a fibred iteration satisfying the basic fixed point equation. Consequently, by the universal property of the container iteration, the two notions of iteration coincide.

In other words, the results of this section shows that our definition of container iteration in dependently type theories with sum types indeed is coherent with the iteration of concrete models of dependently typed languages.

*4.8.1 Key ideas of the iteration-coincidence result.* We start by observing that the op-Grothendieck construction of any $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category $\mathcal{L}$ is itself naturally $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched. Consequently, on one hand, if $\mathcal{L}$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched $\Sigma$-bimodel for sum types (see Def. 4.27), then the op-Grothendieck construction $\Sigma_C \mathcal{L}^{\mathrm{op}}$ inherits an iteration, denoted herein by $\mathtt{it}_{\,\omega}$, from the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enrichment (as established in Theorem 3.19). This gives rise to an iterative category

$$\left(\Sigma_C \mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\,\omega}\right).$$

that satisfies the basic fixed point equation. We refer to $\mathtt{it}_{\,\omega}$ as the *concrete* $\boldsymbol{\omega}\mathbf{Cpo}_\perp$*-container iteration* induced by the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched $\Sigma$-bimodel for sum types $(C, \mathcal{L})$.

On the other hand, if $\mathcal{L}$ is again an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched (coproduct-)extensive indexed category, then it provides an example of an iteration-extensive indexed category, thereby inducing a *container iteration* $\hat{\mathtt{it}}$ on its op-Grothendieck construction, as in Definition 4.17. In this setting, our main result, Theorem 4.36, shows that these two iterations coincide.

We start by establishing that the Grothendieck construction of an indexed $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched category is $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched.

*4.8.2 Basic definition: enriched indexed category.* An $\boldsymbol{\omega}\mathbf{Cpo}_\perp$*-enriched indexed category* herein is a pair $(C, \mathcal{L})$ in which $C$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched category and $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ is a 2-functor such that (1) for any object $A$ of $C$, $\mathcal{L}(A)$ has initial object and colimit of $\omega$-chains, (2) the 2-functor $\mathcal{L}$ locally preserves initial objects and colimits of $\omega$-chains of morphisms. More explicitly, this means that

- $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{Cat}$ is an indexed category;
- for any object $A$ of $C$, $\mathcal{L}(A)$ has initial object and colimits of $\omega$-chains;
- for each pair $(A, B)$ of objects in $C$, we have a functor

$$\mathcal{L}_{A,B} : C(A, B) \to \mathsf{Cat}(\mathcal{L}(B), \mathcal{L}(A)) \tag{4.44}$$

that preserves initial objects and colimits of $\omega$-chains. Explicitly, this means that, for each pair $(f : A \to B, g : A \to B)$ of morphisms in $C$ such that $f \le g$ in $C$, we have a chosen natural transformation $\mathcal{L}(f \le g) : \mathcal{L}(f) \to \mathcal{L}(g)$, such that:

(1) $\mathcal{L}(f \le f) = \mathrm{id}_{\mathcal{L}(f)}$;
(2) $\mathcal{L}(f \le g) \circ \mathcal{L}(h \le f) = \mathcal{L}(h \le g)$;
(3) $\mathcal{L}(f \le g) * \mathcal{L}(f' \le g') = \mathcal{L}(f \circ f' \le g \circ g')$, where $*$ denotes the horizontal composition of natural transformations;
(4) $\mathcal{L}(\perp : A \to B)$ is the functor $\mathcal{L}(\perp) : \mathcal{L}(B) \to (A)$ that is constantly equal to the initial object $0$ of $\mathcal{L}(A)$;
(5) for any pair $(A, B)$ of objects in $C$ and any $\omega$-chain

$$\mathcal{G} \stackrel{\text{def}}{=} (g_0 \le g_1 \le \cdots \le g_n \cdots) \tag{4.45}$$

of morphisms in $C(A, B)$, we have that

$$\mathrm{colim}\,(\mathcal{L} \circ \mathcal{G}) \cong \mathcal{L}\,(\mathrm{colim}\,(\mathcal{G})) : \mathcal{L}(B) \to \mathcal{L}(A)\,.$$

At this point, it is important to keep some examples in mind. We start by observing that:

Lemma 4.23 (Representable functor). *Let $C$ be any $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-category. For any object $K$ of $C$, the pair $(C, C(-,K))$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category, where*

$$C(-,K) : C^{\mathrm{op}} \to \mathsf{CAT} \tag{4.46}$$

*is defined by the composition of arrows*

$$C^{\mathrm{op}} \xrightarrow{C[-,K]} \boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{CAT}, \tag{4.47}$$

*in which $C[-,K] : C^{\mathrm{op}} \to \boldsymbol{\omega}\mathbf{Cpo}_\perp$ is the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched representable functor.*

Proof. It follows directly from the fact that $C$ is $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched, and the fact that the inclusion $\boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{CAT}$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category. □

Furthermore, it is clear that:

Theorem 4.24. *Given an object $K$ of $\mathsf{Cat}_{\omega\perp}$, considering the inclusions*

$$(-)_\perp : \mathsf{PSet} \to \mathsf{Cat}_{\omega\perp}, \qquad \textit{and} \qquad (-)_\perp : \boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{Cat}_{\omega\perp} \tag{4.48}$$

*the pairs*

$$\left(\boldsymbol{\omega}\mathbf{Cpo}_\perp, \mathsf{Cat}_{\omega\perp}\left((-)_\perp, K\right) : \boldsymbol{\omega}\mathbf{Cpo}_\perp^{\mathrm{op}} \to \mathsf{CAT}\right), \quad \left(\mathsf{PSet}, \mathsf{Cat}_{\omega\perp}\left((-)_\perp, K\right) : \mathsf{PSet}^{\mathrm{op}} \to \mathsf{CAT}\right) \tag{4.49}$$

*are $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed categories.*

Proof. The results from the fact that $\mathsf{Cat}_{\omega\perp}(-,K)$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category for any $K$ in $\mathsf{Cat}_{\omega\perp}$. □

*4.8.3 Enriched Grothendieck construction of an enriched indexed category.* The op-Grothendieck construction of an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category naturally inherits fibred $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-structures. We focus on the simplest one (established in Lemma 4.25), which is enough for our developments.

Lemma 4.25. *Let $(C, \mathcal{L})$ be an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category. The op-Grothendieck construction $\Sigma_C \mathcal{L}^{\mathrm{op}}$ naturally inherits an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enrichment. Concretely, for two morphisms*

$$(f, f'), (g, g') : (A, X) \to (B, Y) \tag{4.50}$$

*in $\Sigma_C \mathcal{L}^{\mathrm{op}}$ we define*

$$(f, f') \leq (g, g') \tag{4.51}$$

*if and only if $f \leq g$ in $C$, and the Diagram* (4.52) *commutes in $\mathcal{L}(A)$.*

$$\begin{array}{ccc} \mathcal{L}(f)(Y) & \xrightarrow{f'} & X \\ {\scriptstyle \mathcal{L}(f \leq g)_Y} \downarrow & \nearrow_{g'} & \\ \mathcal{L}(g)(Y) & & \end{array} \tag{4.52}$$

*With this ordering, $\Sigma_C\left(\mathcal{L}^{\mathrm{op}}\right)$ becomes an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-category, and the associated op-fibration, namely* (4.53)*, is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched functor.*

$$P_{\mathcal{L}} : \Sigma_C \mathcal{L}^{\mathrm{op}} \to C \tag{4.53}$$

Proof. In order to verify that $\Sigma_C \mathcal{L}^{\mathrm{op}}$ endowed with the order structure defined above is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-category we verify that, for each pair $((A, X), (B, Y))$ of objects in $\Sigma_C \mathcal{L}^{\mathrm{op}}$, we have that:

- the morphism defined by $(\perp, \iota_0)$, where $\iota_X : \mathcal{L}(\perp)(Y) = 0 \to X$ is the unique morphism, is the initial object/bottom element of $\Sigma_C \mathcal{L}^{\mathrm{op}}((A, X), (B, Y))$;

- given an $\omega$-chain of morphisms
$$\mathcal{G} \stackrel{\text{def}}{=} \left((g_0, g_0') \leq (g_1, g_1') \leq \cdots \leq (g_n, g_n') \cdots\right)$$
in $\Sigma_C \mathcal{L}^{\text{op}}((A, X), (B, Y))$, has a colimit, which is defined by $(\text{colim} g_t, g')$ where
$$g' : \mathcal{L}(\text{colim} g_t)(Y) \to X$$
is induced by the universal property of
$$\mathcal{L}(\text{colim} g_t)(Y) \cong \text{colim}(\mathcal{L}(g_t)(Y))$$
and the morphisms $(g_t')_{t \in \mathbb{N}}$;
- the composition is indeed an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-morphism.

Finally, with the above, we can also conclude that (4.53) is indeed an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-functor. □

*Definition 4.26 (Op-Grothendieck construction).* Let $(C, \mathcal{L})$ be an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category. The op-Grothendieck construction $\Sigma(C, \mathcal{L}^{\text{op}}) \stackrel{\text{def}}{=} \Sigma_C \mathcal{L}^{\text{op}}$ is, herein, considered to be endowed with the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-structure introduced in Lemma 4.25.

*4.8.4 Enriched extensivity.* We introduce the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched counterpart to the notion of (finite-coproduct-)extensive indexed category; namely:

*Definition 4.27 (Enriched extensivity).* An $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category $(C, \mathcal{L})$ is $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-*finite-coproduct-extensive* if $C$ has finite $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts, and $\mathcal{L}$ preserves the finite $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-products of $C^{\text{op}}$.

*Remark 4.28.* It is not in the scope of the present work, but it is clear that given a class of diagrams $\mathfrak{S}$, we have the notion of $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed category w.r.t. $\mathfrak{S}$; namely, the enriched version of Def. 4.5, *e.g.* [31, Section 2.3].

We should observe that our basic examples of $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched indexed categories are $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive. More precisely:

Theorem 4.29. *Let $C$ be an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-enriched category with finite $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts. For any $K \in C$, the representable 2-functor*
$$C(-, K) : C^{\text{op}} \to \mathsf{CAT}$$
*makes the pair $(C, C(-, K))$ into a $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category.*

Proof. It follows directly from the universal property of $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts. □

Theorem 4.30. *Given an object $K$ of $\mathsf{Cat}_{\omega\perp}$, considering the inclusions*
$$(-)_\perp : \mathsf{PSet} \to \mathsf{Cat}_{\omega\perp}, \qquad \textit{and} \qquad (-)_\perp : \boldsymbol{\omega}\mathbf{Cpo}_\perp \to \mathsf{Cat}_{\omega\perp} \tag{4.54}$$
*the pairs*
$$\left(\boldsymbol{\omega}\mathbf{Cpo}_\perp, \mathsf{Cat}_{\omega\perp}((-)_\perp, K) : \boldsymbol{\omega}\mathbf{Cpo}_\perp^{\text{op}} \to \mathsf{CAT}\right), \quad (\mathsf{PSet}, \mathsf{Cat}_{\omega\perp}((-)_\perp, K) : \mathsf{PSet}^{\text{op}} \to \mathsf{CAT}) \tag{4.55}$$
*are $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed categories.*

Proof. The results from the fact that $\mathsf{Cat}_{\omega\perp}(-, K)$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category, and the fact that the inclusions $(-)_\perp$ preserve (finite) $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts. □

As in the ordinary setting, the appeal of the notion of $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category is that it ensures the existence of $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts in the Grothendieck constructions; namely:

Theorem 4.31. *Let $(C, \mathcal{L})$ be an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category. In this setting, we have that $\Sigma_C \mathcal{L}^{\mathrm{op}}$ has $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts. Moreover, the associated op-fibration functor* (4.53) *is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-functor that preserves finite $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts.*

Proof. The proof that (4.56) has finite $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts is similar to the ordinary case, proved in [30, Section 6.6] or, more specifically, [30, Corollary 36] (see Theorem 4.10). □

Recall that, by Theorem 3.19, every $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched category $C$ has an underlying iterative category $(C, \mathtt{it}\,)$ provided that $C$ has finite $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts. With that, we observe that:

Theorem 4.32 (Concrete enriched container iteration). *Let $(C, \mathcal{L})$ be an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category. The $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enrichment of $\Sigma_C \mathcal{L}^{\mathrm{op}}$ yields to an iterative category*

$$\left(\Sigma_C\, \mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\,\omega}\right) \tag{4.56}$$

*with the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched iteration $\mathtt{it}_{\,\omega}$ as established in Def. 3.20. Furthermore, by Theorem 3.20, $\mathtt{it}_{\,\omega}$ satisfies the basic fixed point equation.*

Proof. Indeed, by Lemma 4.25 and Theorem 4.31, $\left(\Sigma_C\, \mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\,\omega}\right)$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-category with $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts. Hence, by Theorem 3.19, we have the underlying iterative category $\left(\Sigma_C\, \mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\,\omega}\right)$ that satisfies the basic fixed point equation. □

*Definition 4.33 (Enriched container iteration).* In the setting of Theorem 4.32, we refer to the iteration $\mathtt{it}_{\,\omega}$ as the *concrete $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-container iteration* induced by the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category $(C, \mathcal{L})$.

*4.8.5 Enriched extensive indexed categories are iteration-extensive.* The $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed categories provide a class of examples of iteration-extensive indexed categories. Consequently, every $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category $(C, \mathcal{L})$ inherits a container iteration $\hat{\mathtt{it}}$ from its underlying iteration-extensive indexed category by Lemma 4.19 (see Definition 4.17). More precisely:

Theorem 4.34 (Underlying iteration-extensive indexed category). *Every $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category $(C, \mathcal{L})$ has an underlying iteration-extensive indexed category $(C, \mathtt{it}\,, \mathcal{L})$, where $\mathtt{it}$ is the iteration inherited by the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-structure in $C$ (as in Definition 3.20).*

Proof. Let $(C, \mathcal{L})$ be an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category. Given a morphism $f \in C\,(A, B \sqcup A)$, the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched iteration is defined by

$$\mathtt{it}^{\,(A,B)} f := \underline{\mathtt{f}} \circ \iota_A. \tag{4.57}$$

where $\underline{\mathtt{f}}$ is the colimit of

$$[\mathrm{id}_B, \perp] \circ [\iota_Y, f] \leq [\mathrm{id}_B, \perp] \circ ([\iota_Y, f])^2 \leq \cdots \leq [\mathrm{id}_B, \perp] \circ ([\iota_Y, f])^n \leq \cdots \tag{4.58}$$

in $C\,(B \sqcup A, B)$.

Since $(C, \mathcal{L})$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category, we have that the image of (4.57) is given by

$$\pi_{\mathcal{L}(A)} \circ \underline{\mathtt{H}} \tag{4.59}$$

where $\underline{\mathtt{H}}$ is the colimit of

$$\left(\mathrm{id}_{\mathcal{L}(B)}, \perp\right) \xrightarrow{\underline{\iota}} G \circ \left(\mathrm{id}_{\mathcal{L}(B)}, \perp\right) \xrightarrow{G(\underline{\iota})} G^2 \circ \left(\mathrm{id}_{\mathcal{L}(B)}, \perp\right) \xrightarrow{G^2(\underline{\iota})} \cdots \xrightarrow{G^n(\underline{\iota})} G^n \circ \left(\mathrm{id}_{\mathcal{L}(B)}, \perp\right) \xrightarrow{G^{n+1}(\underline{\iota})} \cdots \tag{4.60}$$

in $\mathsf{Cat}[\mathcal{L}\,(B)\,, \mathcal{L}\,(B) \times \mathcal{L}\,(A)]$, where

$$G \overset{\mathrm{def}}{=} \left(\pi_{\mathcal{L}(B)}, \mathcal{L}(f) \circ \varsigma^{(B,A)}\right). \tag{4.61}$$

We observe that (4.59) is isomorphic to $\mu\Big(\mathcal{L}(f)\circ\varsigma^{(B,A),\mathcal{L}(B)}\Big)$ by Lemma 4.3. Hence, since $\mathcal{L}\left(\texttt{it}^{(A,B)}f\right)$ is isomorphic to (4.59), we get that

$$\mathcal{L}\left(\texttt{it}^{(A,B)}f\right)\cong\mu\Big(\mathcal{L}(f)\circ\varsigma^{(A,B)},\mathcal{L}(B)\Big). \tag{4.62}$$

This shows that $(C,\texttt{it},\mathcal{L})$, where $(C,\texttt{it})$ is the iterative category underlying the cocartesian $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-category $C$ (as established in Theorem 3.19), is an iteration-extensive indexed category. □

Since, in the setting of Def. 4.33, every concrete $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-container iteration satisfies the basic fixed point equation, we get the following result by the universal property of container iterations (Def. 4.17) induced by iteration-extensive categories (see Theorem 4.20 for the universal property).

Corollary 4.35. *Let $(C,\texttt{it},\mathcal{L})$ be the underlying iteration-extensive indexed category of an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category.*
*The container iteration $\hat{\texttt{it}}$ for the iteration-extensive indexed category $(C,\texttt{it},\mathcal{L})$ is the only fibred iteration for $(C,\texttt{it},\mathcal{L})$ such that $\left(\Sigma_C\,\mathcal{L}^{\mathrm{op}},\hat{\texttt{it}}\right)$ satisfies the basic fixed point equation.*

Proof. Indeed, since by Theorem 3.19 $(C,\texttt{it})$ satisfies the basic fixed point equation, the proof is complete by Theorem 4.20. □

Theorem 4.36. *Let $(C,\texttt{it},\mathcal{L})$ be the underlying iteration-extensive indexed category of an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category $(C,\mathcal{L})$.*
*Denoting by $\texttt{it}_\omega$ the* concrete $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-container iteration *induced by the $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category $(C,\mathcal{L})$ as in Def. 4.33, we have that the identity functor on $\Sigma_C\,\mathcal{L}^{\mathrm{op}}$ induces an iterative category isomorphism*

$$\left(\Sigma_C\,\mathcal{L}^{\mathrm{op}},\hat{\texttt{it}}\right)\to\left(\Sigma_C\,\mathcal{L}^{\mathrm{op}},\texttt{it}_\omega\right), \tag{4.63}$$

*where $\hat{\texttt{it}}$ is the container iteration for the iteration-indexed category $(C,\texttt{it},\mathcal{L})$.*

Proof. By Corollary 4.35, it is enough to verify that indeed $\texttt{it}_\omega$ is a fibred iteration for the iteration-extensive indexed category $(C,\texttt{it},\mathcal{L})$ that makes $\left(\Sigma_C\,\mathcal{L}^{\mathrm{op}},\texttt{it}_\omega\right)$ into an iterative category satisfying the basic fixed point equation.
By Theorem 4.31,

$$P_{\mathcal{L}}:\Sigma_C\mathcal{L}^{\mathrm{op}}\to C \tag{4.64}$$

is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-functor that preserves $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-coproducts. Hence, by Theorem 3.19, we get that (4.64) yields an iterative category morphism

$$\left(\Sigma_C\mathcal{L}^{\mathrm{op}},\texttt{it}_\omega\right)\to(C,\texttt{it}). \tag{4.65}$$

This proves that $\texttt{it}_\omega$ is a fibred iteration for the iteration-extensive indexed category $(C,\texttt{it},\mathcal{L})$. Since, by Theorem 4.32, $\left(\Sigma_C\,\mathcal{L}^{\mathrm{op}},\texttt{it}_\omega\right)$ satisfies the basic fixed point equation, this concludes our proof. □

*4.8.6 Enriched indexed Freyd categories.* To be consistent with our previous developments, we introduce the notion of $\omega$-Freyd indexed category.

*Definition 4.37 (Enriched indexed Freyd category).* An $\omega$-Freyd indexed category is a 5-tuple $(\mathcal{V},C,j,\otimes,\mathcal{L})$ such that $(C,\mathcal{L})$ is an $\boldsymbol{\omega}\mathbf{Cpo}_\perp$-finite-coproduct-extensive indexed category, and $(\mathcal{V},C,j,\otimes)$ is an $\omega$-Freyd category.

Corollary 4.38. *Every $\omega$-Freyd indexed category $(\mathcal{V}, C, j, \otimes, \mathcal{L})$ has an underlying iteration-extensive Freyd category $(\mathcal{V}, C, j, \otimes, \mathtt{it}, \mathcal{L})$ where $(C, \mathtt{it})$ is the iterative category underlying the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-category (with $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts) $C$.*

*Moreover, every $\omega$-indexed Freyd category $(\mathcal{V}, C, j, \otimes, \mathcal{L})$ is taken to an $\omega$-Freyd category*

$$\Sigma\left(\mathcal{V}, C, j, \otimes, \mathcal{L}\right)^{\mathrm{op}} \stackrel{\mathrm{def}}{=} \left(\Sigma_{\mathcal{V}}\left(\mathcal{L} \circ j\right)^{\mathrm{op}}, \Sigma_{C}\mathcal{L}^{\mathrm{op}}, \hat{j}, \hat{\otimes}\right) \tag{4.66}$$

*by the op-Grothendieck construction. The iterative Freyd category underlying* (4.66) *is isomorphic to the iterative Freyd category obtained by taking the op-Grothendieck construction of the underlying iteration-extensive indexed category $(\mathcal{V}, C, j, \otimes, \mathtt{it}, \mathcal{L})$ as in Theorem 4.22.*

*Finally, in particular, the underlying iterative category $\left(\Sigma_C\mathcal{L}^{\mathrm{op}}, \hat{\mathtt{it}}\right)$ satisfies the basic fixed point equation.*

Proof. The result follows from Theorem 4.34. The iteration $\hat{\mathtt{it}}$ is the unique fibred iteration: which coincides with the concrete $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-container iteration. ◻

### 4.9 Iterative indexed categories

The container iteration defined in the op-Grothendieck construction of an iteration-extensive indexed category introduced in Def. 4.17 is defined purely at the equational level – in other words, it is a notion of iteration that is independent of any non-equational structure/convergence.

As demonstrated in the previous results, the concrete $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-container iteration induced by a $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category $(C, \mathcal{L})$ coincides precisely with the container iteration in the op-Grothendieck construction induced by the underlying iteration-extensive indexed category. In Section 4.12 below, we illustrate explicitly that this is the setting of $\mathfrak{P}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$. This result lays a foundational path toward establishing a concrete categorical semantics for the target language of our automatic differentiation macro, CHAD.

We begin by presenting the syntactic categorical semantics of (a fragment) of our target language in detail below. It is important to note that we intentionally avoid imposing additional equations on the iteration construct, as introducing these equations would neither alter our framework nor affect the validity of our findings.

*Definition 4.39.* A *(coarse) iterative indexed category* is *herein* a quadruple

$$\left(C, \mathtt{it}, \mathcal{L}, \mathbf{fd}\left(-,-\right)\right) \tag{4.67}$$

where $(C, \mathtt{it})$ is an iterative category, $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ is a finite-coproduct-extensive indexed category, and, for each morphism $f \in C\left(A, B \sqcup A\right)$ and each pair $(X, Y) \in \mathcal{L}(A) \times \mathcal{L}(B)$, we have a function

$$\begin{aligned} \mathbf{fd}^{(X,Y)}\left(f,-\right): \quad \mathcal{L}(A)\left(\mathcal{L}(f) \circ \varsigma^{(B,A)}\left(Y,X\right), X\right) &\to \mathcal{L}(A)\left(\mathcal{L}\left(\mathtt{it}\, f\right)(Y), X\right) \\ g &\mapsto \mathbf{fd}\left(f, g\right), \end{aligned}$$

where $\varsigma^{(B,A)} : \mathcal{L}(B) \times \mathcal{L}(A) \xrightarrow{\cong} \mathcal{L}(B \sqcup A)$ is the inverse of the comparison morphism.

The family of functions $\mathbf{fd}\left(-,-\right)$ is called *iterative folders.*

*Remark 4.40.* Definition 4.39 could be rephrased as follows: a *(reverse) (coarse) iterative indexed category* is *herein* a quadruple (4.67) where $(C, \mathtt{it})$ is an iterative category, $\mathcal{L} : C^{\mathrm{op}} \to \mathsf{CAT}$ is a finite-coproduct-extensive indexed category, and, for each morphism $(f, f') : (A, X) \to (B, Y) \sqcup (A, X)$ in $\Sigma_C \mathcal{L}^{\mathrm{op}}$, we have a fixed morphism

$$\mathbf{fd}\left(f, f'\right) : \mathcal{L}\left(\mathtt{it}\, f\right)(Y) \to X. \tag{4.68}$$

The above shows how the iterative foldersare precisely the structure needed to define fibred iterations in the op-Grothendieck construction, as precisely established in Lemma 4.45.

Motivated by the results we have on iteration-extensive indexed categories, we observe that:

THEOREM 4.41 (ITERATIVE GROTHENDIECK CONSTRUCTION). *Let* $(\mathcal{C}, \mathtt{it}, \mathcal{L}, \mathbf{fd}(-,-))$ *be an iterative indexed category. The op-Grothendieck construction on* $\mathcal{L}$ *inherits an iteration structure, giving rise to an iterative category*

$$(\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\Sigma}) \tag{4.69}$$

*where, for each morphism*

$$(f, f') : (A, X) \to (B, Y) \sqcup (A, X) \cong \left(B \sqcup A, \varsigma^{(A,B)}(X, Y)\right)$$

*in* $\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}((A, X), (B, Y))$*, we define*

$$\mathtt{it}_{\Sigma}(f, f') = (\mathtt{it}\, f, \mathbf{fd}(f, f')) . \tag{4.70}$$

*Furthermore,* $\mathtt{it}_{\Sigma}$ *is a fibred iteration for the triple* $(\mathcal{C}, \mathtt{it}, \mathcal{L})$ *(see Def. 4.16). For short, we say that* $(\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\Sigma})$ *is the op-Grothendieck construction on the iterative indexed category* $(\mathcal{L}, \mathcal{C}, \mathtt{it}, \mathbf{fd}(-,-))$.

PROOF. Since our definition of iteration is free of equations, we only need to see that, indeed, $(\mathtt{it}\, f, \mathbf{fd}(f, f'))$ is a morphism $(A, X) \to (B, Y)$ in $\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}$, which follows directly from the definitions. □

*Definition 4.42 (Coarse container iteration).* In the setting of Theorem 4.41, the iteration $\mathtt{it}_{\Sigma}$ is called the *coarse container iteration.*

## 4.10 Iterative Freyd indexed categories

We now define an indexed Freyd category that suits our setting. More structured versions of this definition are certainly possible, but we choose the simpler formulation below to avoid overshadowing the paper's main focus. The reader is invited to take a look, for instance, at the Appendix B for more details on the categorical semantics of the target language.

*Definition 4.43 (Iterative Freyd indexed categories).* An iterative Freyd indexed category is a 7-tuple $(\mathcal{L}, \mathcal{V}, \mathcal{C}, j, \otimes, \mathtt{it}, \mathbf{fd}(-,-))$ where $(\mathcal{L}, \mathcal{C}, \mathtt{it}, \mathbf{fd}(-,-))$ is an iterative indexed category such that $\mathcal{L}$ has indexed finite coproducts, and $(\mathcal{V}, \mathcal{C}, j, \otimes)$ is a (distributive) Freyd category.

THEOREM 4.44. *Let* $(\mathcal{L}, \mathcal{V}, \mathcal{C}, j, \otimes, \mathtt{it}, \mathbf{fd}(-,-))$ *be an iterative Freyd indexed category. The op-Grothendieck constructions on* $\mathcal{L}$ *and on* $\mathcal{L} \circ j^{\mathrm{op}} : \mathcal{V}^{\mathrm{op}} \to \mathsf{CAT}$ *induce an iterative Freyd category*

$$\Sigma(\mathcal{L}, \mathcal{V}, \mathcal{C}, j, \otimes, \mathtt{it}, \mathbf{fd}(-,-)) \stackrel{\mathrm{def}}{=} \left(\Sigma_{\mathcal{V}}(\mathcal{L} \circ j^{\mathrm{op}})^{\mathrm{op}}, \Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}, j_{\Sigma}, \otimes_{\Sigma}, \mathtt{it}_{\Sigma}\right) \tag{4.71}$$

*where* $(\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}, \mathtt{it}_{\Sigma})$ *is the iterative category as defined in Eq.* (4.69)*,* $j_{\Sigma}(A, X) \stackrel{\mathrm{def}}{=} (j(A), X)$*, and*

$$(f : A \to B, f') \otimes_{\Sigma} (g : C \to D, g') = (f \otimes g, \mathcal{L}(\pi_A)(f') \sqcup \mathcal{L}(\pi_C)(g')) . \tag{4.72}$$

PROOF. $\Sigma_{\mathcal{V}}(\mathcal{L} \circ j^{\mathrm{op}})^{\mathrm{op}}$ is indeed a distributive category by [30, Proposition 18], [30, Corollary 36] and [30, Theorem 39]. The fact that $\otimes_{\Sigma}$ satisfies the conditions of Def. 3.1 follows from the the construction of products in $\Sigma_{\mathcal{V}}(\mathcal{L} \circ j^{\mathrm{op}})^{\mathrm{op}}$. □

## 4.11 Iteration-extensive indexed categories are iterative

We end this section observing that, indeed, the concrete semantics yields to iterative (Freyd) indexed categories and, hence, they fit the framework established in Section 4.9. We start by observing that every concept of indexed categories we introduced that yields iterative categories through the op-Grothendieck construction naturally provides examples of iterative indexed categories. More precisely:

Lemma 4.45. *Let* $(\mathcal{C}, \mathtt{it})$ *be an iterative category and* $\mathcal{L} : \mathcal{C}^{\mathrm{op}} \to \mathsf{CAT}$ *a finite-coproduct-extensive indexed category. There is a bijection between fibred iterations for the triple* $(\mathcal{C}, \mathtt{it}, \mathcal{L})$ *and iterative indexed categories* $(\mathcal{C}, \mathtt{it}, \mathcal{L}, \mathbf{fd}(-,-))$.

Proof. Every iterative indexed category yields to a fibred iteration for the triple $(\mathcal{C}, \mathtt{it}, \mathcal{L})$ by Theorem 4.41. Reciprocally, if $\underline{\mathtt{it}}$ is a fibred iteration for $(\mathcal{C}, \mathtt{it}, \mathcal{L})$, then we can define the following. For each morphism

$$(f, f') : (A, X) \to (B, Y) \sqcup (A, X) \cong \left(B \sqcup A, \varsigma^{(A,B)}(X, Y)\right)$$

in $\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}((A, X), (B, Y))$, by denoting $\left(g_{\mathtt{it}(f,f')}, g'_{\mathtt{it}(f,f')}\right) \stackrel{\mathrm{def}}{=} \underline{\mathtt{it}}(f, f')$, we establish

$$\mathbf{fd}(f, f') \stackrel{\mathrm{def}}{=} g'_{\mathtt{it}(f,f')}. \tag{4.73}$$

This yields to a iterative indexed category by Remark 4.40. It is clear to see that the constructions above are inverse to each other. □

Theorem 4.46. *Every iteration-extensive indexed category* $(\mathcal{L}, \mathcal{C})$ *has an underlying iterative indexed category by defining, for each morphism*

$$(f, f') : (A, X) \to (B, Y) \sqcup (A, X) \cong \left(B \sqcup A, \varsigma^{(A,B)}(X, Y)\right)$$

*in* $\Sigma_{\mathcal{C}}\mathcal{L}^{\mathrm{op}}((A, X), (B, Y))$,

$$\mathbf{fd}(f, f') \stackrel{\mathrm{def}}{=} \mathtt{fold}\left(\left(\mathcal{L}(f) \circ \varsigma^{(B,A)}\right), Y, (X, f')\right). \tag{4.74}$$

*Moreover, the container iteration* $\hat{\mathtt{it}}$ *(Def. 4.17) induced by the iteration-extensive indexed category* $(\mathcal{L}, \mathcal{C})$ *coincides with the coarse container iteration* $\mathtt{it}_{\Sigma}$ *(Def. 4.42) induced by the underlying iterative indexed category.*

Proof. The proof of the first statement is straightforward, as it only consists of verifying that, indeed, any iteration in the Grothendieck construction of an finite-coproduct-extensive category would induce a iterative indexed category.

The second statement follows directly from the respective definitions. □

Corollary 4.47. *Every* $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$*-finite-coproduct-extensive indexed category has an underlying iterative indexed category. Moreover, the* $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$*-container iterations coincide with the coarse container interations induced by the underlying iterative indexed category.*

Proof. We can easily check this fact directly, or we can conclude the result by Theorem 4.46 since every $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct-extensive indexed category has an underlying iteration-extensive indexed category, and the container iteration coincides with the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-container iteration. □

We can, then, extend the observations above to Freyd categorical versions. More precisely:

Corollary 4.48. *Every iteration-extensive Freyd indexed category* $(\mathcal{V}, \mathcal{C}, j, \otimes, \mathtt{it}, \mathcal{L})$ *has an underlying iterative Freyd indexed category* $(\mathcal{V}, \mathcal{C}, j, \otimes, \mathtt{it}, \mathcal{L}, \mathbf{fd}(-,-))$*, The induced iterative Freyd categories by the op-Grothendieck construcions (as established in Theorem 4.22 and 4.44) coincide.*

Corollary 4.49. *Every* $\omega$*-Freyd indexed category has an underlying iterative Freyd indexed category.*

*Remark 4.50 (Linear types).* We can add one aspect to the categorical semantics of the fragment of the target language we care about; namely, it is linear. More precisely:

*Definition 4.51 (Iterative linear Freyd indexed categories).* A *linear iterative Freyd indexed category* is an iterative Freyd indexed category $(\mathcal{V}, C, j, \otimes, \mathtt{it}, \mathcal{L}, \mathbf{fd}(-,-))$ where $(C, \mathtt{it}, \mathcal{L}, \mathbf{fd}(-,-))$ such that $\mathcal{L}$ has indexed biproducts.

### 4.12 The concrete semantics of our target language: $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ as an op-Grothendieck construction

We now revisit iteration in the concrete semantics of our target language drawing on the concepts introduced in this section. We start by recalling that, since $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched category with finite $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-coproducts, Theorem 3.19 establishes that it forms an underlying iterative category, denoted by $(\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}}), \mathtt{it})$. This is ultimately the structure we want to consider in (the op-Grothendieck construction of) our concrete model.

In order to frame this iteration in our established setting, we start by observing that the indexed category

$$\mathbf{Vect}_{\perp}^{(-)} : \mathsf{PSet}^{\mathrm{op}} \to \mathsf{CAT}, \tag{4.75}$$

as defined in Equation (3.14), satisfies the following:

Theorem 4.52. *The pair $\left(\mathsf{PSet}, \mathbf{Vect}_{\perp}^{(-)}\right)$ is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct extensive indexed category. Moreover, we have an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched isomorphism*

$$\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}}) \cong \Sigma\left(\mathsf{PSet}, \mathbf{Vect}_{\perp}^{(-)}\right)^{\mathrm{op}} \tag{4.76}$$

*between the op-Grothendieck construction $\Sigma\left(\mathsf{PSet}, \mathbf{Vect}_{\perp}^{(-)}\right)$, as established in Theorem 4.26, and $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, as established in Subsect. 3.2 (more precisely, Subsect. 3.2.3).*

*In particular, the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-container iteration induced by $\left(\mathsf{PSet}, \mathbf{Vect}_{\perp}^{(-)}\right)$ is the same as the iteration induced by the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enrichment in $\mathfrak{P}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$ as established in Subsect. 3.6.1.*

Proof. This is a particular case of Theorem 4.30, while the isomorphism is easily obtained by observing that Subsect. 3.2.2 indeed is an $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-enriched isomorphism. □

*Definition 4.53 (Concrete Freyd indexed category).* We can establish the $\omega$-Freyd indexed category $\mathfrak{F}\mathfrak{V}\mathsf{ect} \stackrel{\mathrm{def}}{=} \left(\mathbf{Set}, \mathsf{PSet}, j, \otimes, \mathbf{Vect}_{\perp}^{(-)}\right)$ where $(\mathbf{Set}, \mathsf{PSet}, j, \otimes)$ is the usual $\omega$-Freyd category, and $\left(\mathsf{PSet}, \mathbf{Vect}_{\perp}^{(-)}\right)$ is the $\boldsymbol{\omega}\mathbf{Cpo}_{\perp}$-finite-coproduct extensive indexed category established in Theorem 4.52.

Corollary 4.54. *The $\omega$-Freyd category $\Sigma(\mathfrak{F}\mathfrak{V}\mathsf{ect})^{\mathrm{op}}$ obtained by the op-Grothendieck construction of the $\omega$-Freyd indexed category $\left(\mathbf{Set}, \mathsf{PSet}, j, \otimes, \mathbf{Vect}_{\perp}^{(-)}\right)$ (as established in Corollary 4.38) is isomorphic to the $\omega$-Freyd category $\mathcal{F}\mathbf{Fam}(\mathbf{Vect}^{\mathrm{op}})$, as defined in Theorem 3.9 (Eq.* 3.11*).*

## 5 THE ITERATIVE CHAD CODE TRANSFORMATION AND ITS CORRECTNESS

In this section, we establish our main result concerning the correctness of iterative CHAD. By introducing iteration constructs on the top of the source language, we extend CHAD while keeping the structure-preserving principle as long as we also add iterative foldersto our target language. More precisely, CHAD is still defined as a structure preserving transformation between the source language and the op-Grothendieck construction of the target language concerning the respective indexed categories of linear types.

### 5.1 Source language

We consider a call-by-value functional programming language with variant, product types, non-termination and iteration construct. Like in the case of total CHAD (see, for instance, Subsect. 2.1), our language is constructed over ground types $\mathbf{real}^n$ for arrays of real numbers with static length $n$ ($n \in \mathbb{N}$). In fact, the source language we consider below is effectively the source language we considered for total CHAD (see Section 2 and Appendix A) with iteration constructs on the top of it, and with a larger set of primitive operations, still denoted by

$$\mathrm{Op} = \bigcup_{(k_1,k_2)\in\mathbb{N}\times\mathbb{N}} \bigcup_{(\mathbf{n},\mathbf{m})\in\mathbb{N}^{k_1}\times\mathbb{N}^{k_2}} \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}},$$

allowing them to implement partially defined functions. More precisely, if $\mathbf{n} = \left(n_1, \ldots, n_{k_1}\right)$ and $\mathbf{m} = \left(m_1, \ldots, m_{k_2}\right)$, a primitive operation $\mathrm{op} \in \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$ may, now, implement a partially defined differentiable function, that is to say, a morphism

$$[\![\mathrm{op}]\!] := \left(U_{\mathrm{op}}, [\![[\tilde{\mathrm{op}}]\!]]\right) : M \to N \tag{5.1}$$

of $\mathfrak{P}\mathcal{M}an$. Besides the examples of primitive operations we discussed in Subsect. 2.1 (which are totally defined), the reader can keep the following examples of primitive operations corresponding to partially defined functions in mind:

- the norm operation $\mathsf{norm}_n \in \mathrm{Op}^n_1$ that intends to implement the function

$$[\![\mathsf{norm}_n]\!] := \left(\mathbb{R}^n - \{0\}, [\![[\tilde{\mathsf{norm}}_n]\!]]\right) : \mathbb{R}^n \to \mathbb{R} \tag{5.2}$$

  which is not defined in 0 (the singularity) and is defined by $[\![[\tilde{\mathsf{norm}}_n]\!]](x) = ||x||$ for non-zero $n$-dimensional vectors $x$;
- the reciprocal operation $\mathsf{recpr} \in \mathrm{Op}^1_1$ that intends to implement the function

$$[\![\mathsf{recpr}]\!] := \left(\mathbb{R} - \{0\}, [\![[\tilde{\mathsf{recpr}}]\!]]\right) : \mathbb{R} \to \mathbb{R} \tag{5.3}$$

  which is not defined in 0 and is defined by $[\![[\tilde{\mathsf{recpr}}]\!]](x) = 1/x$ otherwise;
- the normalization operation $\mathsf{norma}_n \in \mathrm{Op}^n_n$ that should implement

$$[\![\mathsf{norma}_n]\!] = \left(\mathbb{R}^n - \{0\}, [\![[\tilde{\mathsf{norma}}_n]\!]]\right) : \mathbb{R}^n \to \mathbb{R}^n \tag{5.4}$$

  where $[\![[\tilde{\mathsf{norma}}_n]\!]]\,(x) = x/|x|$ for $x \in \mathbb{R}^n - \{0\}$;
- the sign function $\mathsf{sign} \in \mathrm{Op}^1_1$ that should implement

$$[\![\mathsf{sign}]\!] = \left(\mathbb{R}_- \cup \mathbb{R}_+, [\![[\tilde{\mathsf{sign}}]\!]]\right) : \mathbb{R} \to \mathbb{R}_0 \sqcup \mathbb{R}_1 \tag{5.5}$$

  where $[\![[\tilde{\mathsf{sign}}]\!]]\,(x) = \iota_{\mathbb{R}_0}(x)$ if $x \in \mathbb{R}_+$, and $[\![[\tilde{\mathsf{sign}}]\!]]\,(x) = \iota_{\mathbb{R}_1}(x)$ if $x \in \mathbb{R}_-$;
- deciders which are generic operations that have coproducts in the codomain. For instance, for each $a \in \mathbb{R}$, the operation ${}_{a}\mathsf{decid}^1_{(1,1)} \in \mathrm{Op}^1_{(1,1)}$ that implements

$$[\![{}_{a}\mathsf{decid}^1_{(1,1)}]\!] = \left(\mathbb{R}_- \cup \mathbb{R}_+, [\![[{}_{a}\tilde{\mathsf{decid}}^1_{(1,1)}]\!]]\right) : \mathbb{R}^1 \to \mathbb{R}_1 \sqcup \mathbb{R}_2 \tag{5.6}$$

  which is not defined at $a \in \mathbb{R}$, $[\![[{}_{a}\tilde{\mathsf{decid}}^1_{(1,1)}]\!]](x) = \iota_1(x) \in \mathbb{R}_1$ if $x > a$, and $[\![[{}_{a}\tilde{\mathsf{decid}}^1_{(1,1)}]\!]](x) = \iota_2(x) \in \mathbb{R}_2$ if $x < a$.

We establish, then, the source language to be a *standard call-by-value language with tuples, variant types, and iteration* constructed over our ground types $\mathbf{real}^n$ and our primitive operations in Op, with the typing rule described in Fig. 5.7 on the top of the language established in App. A

(Fig. A.2). Following the same principle of [38, 39], we do not impose further equations (*e.g.* [8, 15]) on our iteration construct because it is unnecessary to our developments herein.

$$\frac{x : \sigma \vdash t : \tau \sqcup \sigma}{x : \sigma \vdash \mathbf{iterate}(x \to t) : \tau}$$

Fig. 5.7. Typing rules for iteration, after appropriately adding $\mathbf{iterate}(x \to t)$ to our grammar of computations. See Appendix A for more details.

### 5.2 Categorical semantics of the source language and its universal property

Very much like the total setting discussed in Section 2, we can frame the primitive types and operations into a structured (poly)graph/computad, and take the iterative Freyd category $\mathbf{Syn} = \left(\mathbf{Syn}_V, \mathbf{Syn}_C, \mathbf{Syn}_J, \mathbf{Syn}_\otimes, \mathbf{Syn}_{\mathrm{it}}\right)$ freely generated on it. Following this foundational path, we end up with the following syntactic categorical semantics.

Theorem 5.1 (Universal Property of the Source Language). *The iterative Freyd category* $\mathbf{Syn} = \left(\mathbf{Syn}_V, \mathbf{Syn}_C, \mathbf{Syn}_J, \mathbf{Syn}_\otimes, \mathbf{Syn}_{\mathrm{it}}\right)$ *corresponding to our source call-by-value language has the following universal property. Given any iterative Freyd category* $(\mathcal{V}, C, j, \otimes, \mathtt{it})$*, for each pair* $(K, \mathtt{h})$ *where* $K = (K_n)_{n\in\mathbb{N}}$ *is a family of objects in* $C$ *and* $\mathtt{h} = \left(h_{\mathrm{op}}\right)_{\mathrm{op}\in\mathrm{Op}}$ *is a consistent family of morphisms in* $C$*, there is an unique iterative Freyd category morphism*

$$\left(H, \hat{H}\right) : \mathbf{Syn} \to (\mathcal{V}, C, j, \otimes, \mathtt{it}) \tag{5.8}$$

*such that* $\hat{H}(\mathbf{real}^n) = K_n$ *and* $\hat{H}(\mathrm{op}) = h_{\mathrm{op}}$ *for any* $n \in \mathbb{N}$ *and any primitive operation* $\mathrm{op} \in \mathrm{Op}$.

Making use of the universal property above and the fact that $\mathcal{F}\mathcal{M}an = (\mathcal{V}\mathcal{M}an, \mathfrak{P}\mathcal{M}an, \mathfrak{p}, \otimes)$ is an $\omega$-Freyd category (hence an iterative Freyd category), we can easily define our concrete semantics of the source language in terms of partially defined differentiable functions, morphisms of $\mathfrak{P}\mathcal{M}an$. More precisely, by the universal property of $\mathbf{Syn} = \left(\mathbf{Syn}_V, \mathbf{Syn}_C, \mathbf{Syn}_J, \mathbf{Syn}_\otimes, \mathbf{Syn}_{\mathrm{it}}\right)$, we can define the concrete semantics as an iterative Freyd category morphism

$$[\![-]\!] : \mathbf{Syn} \to \mathcal{F}\mathcal{M}an, \tag{5.9}$$

where $\mathcal{F}\mathcal{M}an$ is the $\omega$-Freyd category defined in Eq. (3.4).

Theorem 5.2 (Semantics iterative Freyd morphism). *There is only one iterative Freyd category morphism* (5.9)

$$[\![-]\!] : \mathbf{Syn} \to \mathcal{F}\mathcal{M}an \tag{5.10}$$

*such that, for each* $n \in \mathbb{N}$, $[\![\mathbf{real}^n]\!] = \mathbb{R}^n$ *and, for each* $(\mathbf{n}, \mathbf{m}) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2}$, $[\![\mathrm{op}]\!]$ *is the morphism* $\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_{k_1}} \to \mathbb{R}^{m_1} \sqcup \cdots \sqcup \mathbb{R}^{m_{k_2}}$ *of* $\mathfrak{P}\mathcal{M}an$ *that* op *intends to implement.*

*The iterative Freyd category morphism* (5.10) *gives the concrete semantics of the source language.*

Proof. Since $\mathcal{F}\mathcal{M}an = (\mathcal{V}\mathcal{M}an, \mathfrak{P}\mathcal{M}an, \mathfrak{p}, \otimes)$ is an $\omega$-Freyd category, it is an iterative Freyd category. Hence, the result follows from the universal property of $\mathbf{Syn}$. ◻

### 5.3 Target language and its categorical semantics

Our target language is defined following the same principle of Section 2.3; namely, we use a variant of the dependently typed enriched effect calculus [52, Chapter 5] that extends our call-by-value source language established in Section 5.1, and over *primitive linear operations* $\mathsf{lop} \in \mathsf{LOp}^{\mathbf{r};\mathbf{n}}_{\mathbf{m}}$. We refer the reader to Appendix B for the detailed grammar and typing rules of our language.

To serve as a practical target language for the automatic derivatives of all programs from the source language, we make the following assumption: for each $(k_1, k_2) \in \mathbb{N}$, each $(\mathbf{n}, \mathbf{m}) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2}$, and each $\mathrm{op} \in \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$, there is a linear operation

$$D\mathrm{op} \in \mathsf{LOp}^{\mathbf{n};\mathbf{m}}_{\mathbf{n}} \tag{5.11}$$

that intends to implement the (transpose) derivative of the primitive operation op.

Being a proper extension of our (call-by-value) source language established in Section 5.1, our target language, established in Appendix B, already allows for non-termination (and partially defined functions as semantics of primitive operations). In order to implement our macro, we add our novel iterative folderson the top of our language, following the principle of our categorical semantics – namely, the *iterative indexed categories* or, more precisely, the *iterative Freyd indexed categories* as introduced in Subsect. 4.9.

$$\frac{x : \sigma; \mathsf{v} : \underline{\rho} \vdash s : \underline{\tau}[{}^{\mathbf{iterate}\,(x \to t)}/_{y}] \qquad \begin{array}{c} x : \sigma \vdash t : \tau \sqcup \sigma \\ x : \sigma; \mathsf{v} : \mathbf{case}\, t\, \mathbf{of}\, \{\mathbf{in}_1\, y \to \underline{\tau} \mid \mathbf{in}_2\, x \to \underline{\rho'}\} \vdash r : \underline{\rho'} \end{array}}{x : \sigma; \mathsf{v} : \underline{\rho} \vdash \mathbf{fold}\, s\, \mathbf{with}\, \mathsf{v} \to r : \underline{\rho'}}$$

Fig. 5.12. Typing rules for iterative foldersfor the target language on top of the rules of A.2, B.3 and B.2.

Similarly to the case of total CHAD (see Subsect. 2.3), there is a freely generated iterative Freyd indexed category $\mathfrak{LSyn} = \left(\mathbf{LSyn}, \mathbf{LSyn}_V, \mathbf{LSyn}_C, \mathbf{LSyn}_j, \otimes, \mathbf{LSyn}_{\mathtt{it}}, \mathbf{fd}_{\mathfrak{S}yn}(-,-)\right)$ on (the fragment of linear types of) our target language (see Appendix B for more details on the target language). The important aspect of this categorical structure is that we can define the concrete semantics as following.

Theorem 5.3 (Syntactic categorical semantics). *There is a freely generated iterative Freyd indexed category* $\mathfrak{LSyn} = \left(\mathbf{LSyn}, \mathbf{LSyn}_V, \mathbf{LSyn}_C, \mathbf{LSyn}_j, \otimes, \mathbf{LSyn}_{\mathtt{it}}, \mathbf{fd}_{\mathfrak{S}yn}(-,-)\right)$ *on the linear fragment of the target language (see Appendix B for more details on the target language).*

Corollary 5.4. *The op-Grothendieck construction* $\Sigma\,(\mathfrak{LSyn})^{\mathrm{op}}$ *gives an iterative Freyd category that we denote by*

$$\begin{aligned}
\mathfrak{P}\mathbf{TSyn} &\overset{\text{def}}{=} \Sigma\,(\mathfrak{LSyn})^{\mathrm{op}} && (5.13)\\
&= \Sigma\left(\mathbf{LSyn}, \mathbf{LSyn}_V, \mathbf{LSyn}_C, \mathbf{LSyn}_j, \otimes, \mathbf{LSyn}_{\mathtt{it}}, \mathbf{fd}_{\mathfrak{S}yn}(-,-)\right)^{\mathrm{op}} && (5.14)\\
&= \left(\Sigma_{\mathbf{LSyn}_V}\left(\mathbf{LSyn} \circ \mathbf{LSyn}_j\right)^{\mathrm{op}}, \Sigma_{\mathbf{LSyn}_C}\mathbf{LSyn}^{\mathrm{op}}, \mathbf{LSyn}_j, \otimes_{T\Sigma}, \mathtt{it}_{\,\Sigma\mathbf{LSyn}}\right). && (5.15)
\end{aligned}$$

*The iteration* $\mathtt{it}_{\,\Sigma\mathbf{LSyn}}$ *is defined as follows: for each morphism*

$$(f, f') : (A, X) \to (B, Y) \sqcup (A, X) \cong \left(B \sqcup A, \varsigma^{(A,B)}(X, Y)\right)$$

*in* $\Sigma_{\mathbf{LSyn}_C}\mathbf{LSyn}^{\mathrm{op}}\left((A,X),(B,Y)\right)$*, we have*

$$\mathtt{it}_{\Sigma\mathbf{LSyn}}(f,f') = \left(\mathtt{it}\, f, \mathbf{fd}_{\mathfrak{S}yn}(f,f')\right). \tag{5.16}$$

Proof. Since $\mathfrak{L}\mathfrak{S}\mathbf{yn} = \left(\mathbf{LSyn}, \mathbf{LSyn}_V, \mathbf{LSyn}_C, \mathbf{LSyn}_j, \otimes, \mathbf{LSyn}_{\mathtt{it}}, \mathbf{fd}_{\mathfrak{S}yn}(-,-)\right)$ is an iterative Freyd indexed category, the result follows from Theorem 4.44 and Theorem 4.41. □

The usual forgetful functors $\mathcal{VM}an \to \mathbf{Set}$ and $\mathfrak{P}\mathcal{M}an \to \mathsf{PSet}$ induce a Freyd category morphism $\mathfrak{U} : \mathcal{FM}an \to \mathcal{FS}et$, where $\mathcal{FS}et := (\mathbf{Set}, \mathsf{PSet}, j, \otimes)$ is the canonical $\omega$-Freyd category obtained by the usual inclusion $j : \mathbf{Set} \to \mathsf{PSet}$.

For the purpose of stating Theorem 5.5, we denote by $[\![-]\!]_0 : \mathbf{Syn} \to \mathcal{FS}et$ the $\omega$-Freyd category morphism defined as the composition $[\![-]\!]_0 \overset{\mathrm{def}}{=} \mathfrak{U} \circ [\![-]\!]$, where $[\![-]\!]$ is the iterative Freyd category morphism that gives the concrete semantics of the source language, introduced in Eq. (5.9).

Theorem 5.5 (Concrete categorical semantics of the target language as an indexed functor). *Let* $\mathfrak{F}\mathfrak{V}\mathrm{ect}$ *be the iterative Freyd indexed category established in Definition 4.53. The iterative Freyd category morphism*

$$[\![-]\!]_0 : \mathbf{Syn} \to \mathcal{FS}et \tag{5.17}$$

*can be uniquely extended into a structure-preserving indexed functor*[11]

$$[\![-]\!]^{\mathfrak{I}} : \mathfrak{L}\mathfrak{S}\mathbf{yn} \to \mathfrak{F}\mathfrak{V}\mathrm{ect} \tag{5.18}$$

*with underlying natural transformation*

$$[\![-]\!]^{\mathfrak{I}} : \mathbf{LSyn} \to \mathbf{Vect}_{\perp}^{([\![-]\!]_0)} \tag{5.19}$$

*satisfying the following: for each* $k_1 \in \mathbb{N}$ *and each* $\mathbf{n} = \left(n_1, \ldots, n_{k_1}\right) \in \mathbb{N}^{k_1}$*, the component*

$$[\![-]\!]^{\mathfrak{I}}_{\mathbf{real}^{n_1}\times\cdots\times\mathbf{real}^{n_{k_1}}} : \mathbf{LSyn}\left(\mathbf{real}^{n_1} \times \cdots \times \mathbf{real}^{n_{k_1}}\right) \to \mathbf{Vect}_{\perp}^{\left([\![\mathbf{real}^{n_1}]\!]_0\times\cdots\times[\![\mathbf{real}^{n_{k_1}}]\!]_0\right)} \tag{5.20}$$

*of the natural transformation* (5.19) *at* $\mathbf{real}^{n_1} \times \cdots \times \mathbf{real}^{n_{k_1}}$ *is such that, for each primitive* $\mathrm{op} \in \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$,

$$[\![D\mathrm{op}]\!]^{\mathfrak{I}}_{\mathbf{n}} \in \mathbf{Vect}_{\perp}^{\left([\![\mathbf{real}^{n_1}]\!]_0\times\cdots\times[\![\mathbf{real}^{n_{k_1}}]\!]_0\right)}\left(\overline{\underline{\underline{[\![\mathbf{real}_{\mathbf{m}}]\!]}}} \circ [\![\mathrm{op}]\!], \overline{[\![\mathbf{real}^{\mathbf{n}}]\!]}\right),$$

*where* $[\![D\mathrm{op}]\!]^{\mathfrak{I}}_{\mathbf{n}}$ *is the family of linear transformations that* $D\mathrm{op}$ *intends to implement,*

Proof. This follows directly from the universal property of the (categorical semantics of the) syntax of our target language. We refer the reader to Appendix B for more detail on that. □

The op-Grothendieck constructions of $\mathfrak{L}\mathfrak{S}\mathbf{yn}$ and $\mathfrak{F}\mathfrak{V}\mathrm{ect}$, together with (5.18), induce an iterative Freyd category morphism between the corresponding op-Grothendieck constructions. This defines the most important aspect of the semantics of the target language to our correctness proof. More precisely:

Corollary 5.6 (Container categorical semantics of the target language). *The op-Grothendieck construction functor takes* (5.18) *to an iterative Freyd category morphism*

$$[\![-]\!] : \mathfrak{P}\mathbf{TSyn} \to \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \tag{5.21}$$

[11] See, for instance, [18] or [30, Subsect. 6.9] for indexed functors.

*where* $\mathfrak{P}\mathbf{TSyn} \stackrel{\mathrm{def}}{=} \Sigma\,(\mathfrak{L}\mathfrak{S}\mathbf{yn})^{\mathrm{op}}$ *corresponds to the category of containers over the target language, as in Corollary 5.4. In particular, for each morphism* $(\mathrm{op}, D\mathrm{op}')$ *in* $\mathfrak{P}\mathbf{TSyn}$ *where* $(\mathrm{op}, D\mathrm{op}) \in \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}} \times \mathrm{Op}^{\mathbf{n}'}_{\mathbf{m}'}$ *are primitive operations, we have that*

$$[\![ (\mathrm{op}, D\mathrm{op}') ]\!] = \left( [\![\mathrm{op}]\!]_0, [\![D\mathrm{op}']\!]^{\mathfrak{I}}_{\mathbf{n}'} \right) \tag{5.22}$$

Proof. It follows from Theorem 4.44 and Theorem 4.41. □

### 5.4 Iterative CHAD: the AD macro and its categorical semantics

We finally can extend the macro, defined in [30] for total languages and discussed in 2.4. This is still defined in the structure-preserving manner described, resulting the following definition for iteration – which basically is given by the principle of our categorical semantics introduced in Subsect. 4.9.

That is to say, after implementing the derivatives of the primitive operations following the same principle of total CHAD, we define CHAD as the only *structure-preserving transformation (iterative Freyd category morphism)* between the source language and the op-Grothendieck construction (category of contained) obtained from the target language that extends the implementation on the primitive operations. This allows for a straightforward correctness proof (presented in Theorem 5.8).

We write

- given $\Gamma; \mathsf{v} : \underline{\tau} \vdash t : \underline{\sigma}_i$, we write the $i$-th coprojection $\Gamma; \mathsf{v} : \underline{\tau} \vdash \mathbf{coproj}_i\,(t) \stackrel{\mathrm{def}}{=} \langle \underline{0}, \ldots, \underline{0}, t, \underline{0}, \ldots, \underline{0} \rangle : \underline{\sigma}_1 \times \cdots \times \underline{\sigma}_n$;
- for a list $x_1, \ldots, x_n$ of distinct identifiers, we write $\mathbf{idx}(x_i; x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} i$ for the index of the identifier $x_i$ in this list.

We define for each type $\tau$ of the source language:

- a cartesian type $\overleftarrow{\mathbb{D}}(\tau)_1$ of reverse-mode primals;
- a linear type $\overleftarrow{\mathbb{D}}(\tau)_2$ (with free term variable $p$) of reverse-mode cotangents.

We extend $\overleftarrow{\mathbb{D}}(-)$ to act on typing contexts $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ as

$$\begin{aligned} \overleftarrow{\mathbb{D}}(\Gamma)_1 &\stackrel{\mathrm{def}}{=} x_1 : \overleftarrow{\mathbb{D}}(\tau_1)_1, \ldots, x_n : \overleftarrow{\mathbb{D}}(\tau_n)_n && \text{(a cartesian typing context)} \\ \overleftarrow{\mathbb{D}}(\Gamma)_2 &\stackrel{\mathrm{def}}{=} \overleftarrow{\mathbb{D}}(\tau_1)_2[{}^{x_1}/_{p}] \times \cdots \times \overleftarrow{\mathbb{D}}(\tau_n)_2[{}^{x_n}/_{p}] && \text{(a linear type).} \end{aligned}$$

Similarly, we define for each term $t$ of the source language and a list $\overline{\Gamma}$ of identifiers that contains at least the free identifiers of $t$:

- a term $\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t)_1$ that represents the reverse-mode primal computation associated with $t$;
- a term $\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t)_2$ that represents the reverse-mode cotangent computation associated with $t$.

These code transformations are well-typed in the sense that a source language term $t$ that is typed according to $\Gamma \vdash t : \tau$ is translated into a term of the target language that is typed as follows:

$$\overleftarrow{\mathbb{D}}(\Gamma)_1 \vdash \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t) : \Sigma p : \overleftarrow{\mathbb{D}}(\tau)_1 . \overleftarrow{\mathbb{D}}(\tau)_2 \multimap \overleftarrow{\mathbb{D}}(\Gamma)_2,$$

where $\overline{\Gamma}$ is the list of identifiers that occurs in $\Gamma$ (that is, $\overline{x_1 : \tau_1, \ldots, x_n : \tau_n} \stackrel{\mathrm{def}}{=} x_1, \ldots, x_n$). We note that $t \stackrel{\beta\eta}{=} s$ implies $\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t) \stackrel{\beta\eta+}{=} \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(s)$.

We assume that have chosen suitable terms

$$\begin{aligned} &x_1 : \mathbf{real}^{n_1}, \ldots, x_{k_1} : \mathbf{real}^{n_{k_1}}; \mathsf{v} : \mathbf{case}\ \mathrm{op}(x_1, \ldots, x_{k_1})\ \mathbf{of}\ \{\mathbf{in}_1\ \_ \to \underline{\mathbf{reals}}^{m_1} \mid \cdots \mid \mathbf{in}_{k_2}\ \_ \to \underline{\mathbf{reals}}^{m_{k_2}}\} \\ &\quad \vdash D\mathrm{op}(x_1, \ldots, x_{k_1})(\mathsf{v}) : \underline{\mathbf{reals}}^{n_1} \times \cdots \times \underline{\mathbf{reals}}^{n_{k_1}} \end{aligned}$$

to represent the reverse-mode derivatives of the primitive operations $\mathrm{op} \in \mathrm{Op}^m_{n_1,\ldots,n_k}$.

We then define our reverse-mode AD code transformation as follows on types

$$\overleftarrow{\mathbb{D}}(\mathbf{real}^n)_1 \stackrel{\text{def}}{=} \mathbf{real}^n$$

$$\overleftarrow{\mathbb{D}}(\tau_1 \times \cdots \times \tau_n)_1 \stackrel{\text{def}}{=} \overleftarrow{\mathbb{D}}(\tau_1)_1 \times \cdots \times \overleftarrow{\mathbb{D}}(\tau_n)_1$$

$$\overleftarrow{\mathbb{D}}(\tau_1 \sqcup \cdots \sqcup \tau_n)_1 \stackrel{\text{def}}{=} \overleftarrow{\mathbb{D}}(\tau_1)_1 \sqcup \cdots \sqcup \overleftarrow{\mathbb{D}}(\tau_n)_1$$

$$\overleftarrow{\mathbb{D}}(\mathbf{real}^n)_2 \stackrel{\text{def}}{=} \underline{\mathbf{reals}}^n$$

$$\overleftarrow{\mathbb{D}}(\tau_1 \times \cdots \times \tau_n)_2 \stackrel{\text{def}}{=} \overleftarrow{\mathbb{D}}(\tau_1)_2[^{\mathbf{pr}_1\, p}/_{p}] \times \cdots \times \overleftarrow{\mathbb{D}}(\tau_n)_2[^{\mathbf{pr}_n\, p}/_{p}]$$

$$\overleftarrow{\mathbb{D}}(\tau_1 \sqcup \cdots \sqcup \tau_n)_2 \stackrel{\text{def}}{=} \mathbf{case}\, p\, \mathbf{of}\, \{\mathbf{in}_1\, p \to \overleftarrow{\mathbb{D}}(\tau_1)_2 \mid \cdots \mid \mathbf{in}_n\, p \to \overleftarrow{\mathbb{D}}(\tau_n)_2\}$$

and on terms

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\mathrm{op}(t_1,\ldots,t_k)) \stackrel{\text{def}}{=} \cdots\cdots \quad \begin{array}{l}\mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t_1)\, \mathbf{of}\, \langle x_1, x_1'\rangle \to \cdots \\ \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t_k)\, \mathbf{of}\, \langle x_k, x_k'\rangle \to \\ \langle \mathrm{op}(x_1,\ldots,x_k), \underline{\lambda}\mathsf{v}.\mathbf{let}\, \mathsf{v} = D\mathrm{op}(x_1,\ldots,x_k)(\mathsf{v})\, \mathbf{in} \\ \qquad x_1' \bullet \mathbf{pr}_1\, \mathsf{v} + \cdots + x_k' \bullet \mathbf{pr}_k\, \mathsf{v}\rangle\end{array}$$

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(x) \stackrel{\text{def}}{=} \cdots\cdots \quad \langle x, \underline{\lambda}\mathsf{v}.\mathbf{coproj}_{\mathbf{idx}(x;\overline{\Gamma})}(\mathsf{v})\rangle$$

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\mathbf{let}\, x = t\, \mathbf{in}\, s) \stackrel{\text{def}}{=} \cdots\cdots \quad \begin{array}{r}\mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t)\, \mathbf{of}\, \langle x, x'\rangle \to \\ \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma},x}(s)\, \mathbf{of}\, \langle y, y'\rangle \to \\ \langle y, \underline{\lambda}\mathsf{v}.\mathbf{let}\, \mathsf{v} = y' \bullet \mathsf{v}\, \mathbf{in}\, \mathbf{pr}_1\, \mathsf{v} + x' \bullet (\mathbf{pr}_2\, \mathsf{v})\rangle\end{array}$$

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\langle t_1,\ldots,t_n\rangle) \stackrel{\text{def}}{=} \cdots\cdots \quad \begin{array}{l}\mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t_1)\, \mathbf{of}\, \langle x_1, x_1'\rangle \to \cdots \\ \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t_n)\, \mathbf{of}\, \langle x_n, x_n'\rangle \to \\ \langle\langle x_1, \ldots x_n\rangle, \underline{\lambda}\mathsf{v}.x_1' \bullet (\mathbf{pr}_1\, \mathsf{v}) + \cdots \\ \qquad + x_n' \bullet (\mathbf{pr}_n\, \mathsf{v})\rangle\end{array}$$

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\mathbf{case}\, t\, \mathbf{of}\, \langle x_1,\ldots,x_n\rangle \to s) \stackrel{\text{def}}{=} \cdots\cdots \quad \begin{array}{r}\mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t)\, \mathbf{of}\, \langle x, x'\rangle \to \\ \mathbf{case}\, x\, \mathbf{of}\, \langle x_1,\ldots,x_n\rangle \to \\ \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma},x_1,\ldots,x_n}(s)\, \mathbf{of}\, \langle y, y'\rangle \to \\ \langle y, \underline{\lambda}\mathsf{v}.\mathbf{let}\, \mathsf{v} = y' \bullet \mathsf{v}\, \mathbf{in}\, \mathbf{pr}_1\, \mathsf{v} + x'(\mathbf{pr}_2\, \mathsf{v})\rangle\end{array}$$

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\mathbf{in}_i\, t) \stackrel{\text{def}}{=} \cdots\cdots \quad \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t)\, \mathbf{of}\, \langle x, x'\rangle \to \langle \mathbf{in}_i\, x, x'\rangle$$

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\mathbf{case}\, t\, \mathbf{of}\, \{\mathbf{in}_1\, x_1 \to s_1 \mid \cdots \mid \mathbf{in}_n\, x_n \to s_n\}) \stackrel{\text{def}}{=} \cdots\cdots\cdots \begin{array}{l} \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(t)\, \mathbf{of}\, \langle y, y' \rangle \to \\ \mathbf{case}\, y\, \mathbf{of}\, \{\mathbf{in}_1\, x_1 \to \\ \quad \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}, x_1}(s_1)\, \mathbf{of}\, \langle z_1, z_1' \rangle \to \\ \quad \langle z_1, \underline{\lambda} \mathsf{v}.\mathbf{let}\, \mathsf{v} = z_1' \bullet \mathsf{v}\, \mathbf{in}\, \mathbf{pr}_1\, \mathsf{v} + \\ \qquad (\mathbf{let}\, y = \mathbf{in}_1\, x_1\, \mathbf{in}\, y') \bullet (\mathbf{pr}_2\, \mathsf{v}) \rangle \\ \qquad \mid \cdots \mid \\ \qquad \mathbf{in}_n\, x_n \to \\ \quad \mathbf{case}\, \overleftarrow{\mathbb{D}}^{\overline{\Gamma}, x_n}(s_n)\, \mathbf{of}\, \langle z_n, z_n' \rangle \to \\ \quad \langle z_n, \underline{\lambda} \mathsf{v}.\mathbf{let}\, \mathsf{v} = z_n' \bullet \mathsf{v}\, \mathbf{in}\, \mathbf{pr}_1\, \mathsf{v} + \\ \qquad (\mathbf{let}\, y = \mathbf{in}_n\, x_n\, \mathbf{in}\, y') \bullet (\mathbf{pr}_2\, \mathsf{v}) \rangle \} \end{array}$$

$$\overleftarrow{\mathbb{D}}(\mathbf{iterate}(x \to t)) \stackrel{\text{def}}{=} \cdots\cdots\cdots\cdots \quad \mathbf{let}\, f = \overleftarrow{\mathbb{D}}^{x}(t)\, \mathbf{in}$$
$$\langle \mathbf{iterate}(x \to \mathbf{pr}_1\, f(x)), \underline{\lambda} \mathsf{v}.\mathbf{fold}\, \mathsf{v}\, \mathbf{with}\, \mathsf{v} \to (\mathbf{pr}_2\, f(x)) \bullet \mathsf{v} \rangle.$$

Similarly, we can derive the following derivative rule for iteration with context:

$$\overleftarrow{\mathbb{D}}^{\overline{\Gamma}}(\mathbf{iterate}_{\Gamma}(x \to t)) \stackrel{\text{def}}{=} \cdots\cdots \quad \mathbf{let}\, f = \overleftarrow{\mathbb{D}}^{\overline{\Gamma}, x}(t)\, \mathbf{in}$$
$$\langle \mathbf{iterate}_{\overline{\Gamma}}(x \to \mathbf{pr}_1\, f(x)), \underline{\lambda} \mathsf{v}.\mathbf{pr}_2\, (\mathbf{fold}\, \mathsf{v}\, \mathbf{with}\, \mathsf{v} \to (\mathbf{pr}_2\, f(x)) \bullet \mathsf{v}) \rangle.$$

Theorem 5.7. *There is only one iterative Freyd category morphism*

$$\overleftarrow{\mathbb{D}} = \left(\overleftarrow{\mathbb{D}}^{v}, \overleftarrow{\mathbb{D}}^{c}\right) : \mathbf{Syn} \to \mathfrak{P}\mathbf{TSyn} \tag{5.23}$$

*such that for each* $n \in \mathbb{N}$,

$$[\![ \overleftarrow{\mathbb{D}}(\mathbf{real}^n) ]\!] = \left(\mathbb{R}^n, \overline{\mathbb{R}^n}\right), \tag{5.24}$$

*and, for each* $\mathrm{op} \in \mathrm{Op}$, $\overleftarrow{\mathbb{D}}(\mathrm{op})$ *implements the CHAD-derivative of the primitive operation* $\mathrm{op} \in \mathrm{Op}$ *of the source language, that is to say,*

$$\overleftarrow{\mathbb{D}}^{c}(\mathrm{op}) = (\mathrm{op}, D\mathrm{op}) \qquad \textit{or, in other words,} \qquad [\![ \overleftarrow{\mathbb{D}}(\mathrm{op}) ]\!] = {}_{cha}\mathfrak{D}[\![\mathrm{op}]\!], \tag{5.25}$$

*where*

$${}_{cha}\mathfrak{D} := (\mathfrak{D}, \mathfrak{D}_{\omega}) : \mathcal{F}\mathcal{M}an \to \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \tag{5.26}$$

*is the CHAD-derivative* $\omega$*-Freyd category morphism defined in* (3.17)*. The iterative Freyd category morphism* $\overleftarrow{\mathbb{D}}$ *corresponds to the CHAD AD macro* $\overleftarrow{\mathbb{D}}()$ *defined above.*

Proof. Since $\mathfrak{P}\mathbf{TSyn}$ is an iterative Freyd category, the uniqueness of $\overleftarrow{\mathbb{D}}$ as an iterative Freyd category morphism follows from the universal property of **Syn**, established in Theorem 5.1, while the definition of the macro above follows the structure-preserving principle (making into a iterative Freyd category morphism satisfying the same conditions). □

## 5.5 Correctness of the iterative CHAD

Finally, we are in a position to state and prove the correctness of CHAD in the total language setting described above. Recall that our specification is given by the $\omega$-Freyd category morphism

$${}_{cha}\mathfrak{D} := (\mathfrak{D}, \mathfrak{D}_{\omega}) : \mathcal{F}\mathcal{M}an \to \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \tag{5.27}$$

which we refer to as the *CHAD-derivative*, as established in (3.17). That is, the CHAD-derivative of the semantics of a program—defined by Theorem 5.2 in the source language—should coincide with the semantics, as given in Corollary 5.6, of the macro $\underleftarrow{\mathbb{D}}(-)$ applied to that program.

Theorem 5.8. *Let $\underleftarrow{\mathbb{D}}$ denote the macro iterative Freyd category morphism defined in Theorem 5.7. Let $[\![-]\!]$ be the iterative Freyd category morphism giving the semantics of the source language, as introduced in Theorem 5.2. Finally, let $[\![-]\!]$ denote the iterative Freyd category morphism interpreting the target language, as described in Corollary 5.6.*

*The diagram of iterative Freyd category morphisms below commutes.*

$$\begin{array}{ccc} \mathbf{Syn} & \xrightarrow{\underleftarrow{\mathbb{D}}} & F\mathbf{LSyn} \\ {\scriptstyle [\![-]\!]}\downarrow & & \downarrow{\scriptstyle [\![-]\!]} \\ \mathcal{F}\mathcal{M}an & \xrightarrow[{}_{cha}\mathfrak{D}]{} & \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right) \end{array} \tag{5.28}$$

Proof. By the universal property of the iterative Freyd category $\mathbf{Syn}$ established in Theorem 5.1, since $\mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ is an iterative Freyd category, we have that there is only one iterative Freyd category morphism

$$\mathtt{C} = \left(\mathfrak{C}, \hat{\mathfrak{C}}\right) : \mathbf{Syn} \to \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right), \tag{5.29}$$

such that

c1) for each $n \in \mathbb{N}$, $\hat{\mathfrak{C}}\left(\mathbf{real}^{n}\right) = \left(\mathbb{R}^{n}, \overline{\mathbb{R}^{n}}\right)$;

c2) $\hat{\mathfrak{C}}\left(\mathrm{op}\right) = \mathfrak{D}_{\omega}[\![\mathrm{op}]\!] = [\![\hat{\underleftarrow{\mathbb{D}}}(\mathrm{op})]\!]$ for each $\mathrm{op} \in \mathrm{Op}$.

Since both $[\![\underleftarrow{\mathbb{D}}(-)]\!]$ and ${}_{cha}\mathfrak{D}\circ[\![-]\!]$ are iterative Freyd category morphisms $\mathbf{Syn} \to \mathcal{F}\mathbf{Fam}\left(\mathbf{Vect}^{\mathrm{op}}\right)$ such that c1) and c2) hold, we get that $[\![\underleftarrow{\mathbb{D}}(-)]\!] = \mathfrak{C} = {}_{cha}\mathfrak{D}\circ[\![-]\!]$. That is to say, Diagram (5.8) commutes. □

Corollary 5.9. *For any well-typed program $x : \tau \vdash t : \sigma$, we have that $[\![\underleftarrow{\mathbb{D}}(t)]\!] = \mathfrak{D}_{\omega}[\![t]\!]$.*

Proof. It follows from the fact that $[\![\hat{\underleftarrow{\mathbb{D}}}(t)]\!] = \mathfrak{D}_{\omega}[\![t]\!]$ for any morphism $t$ of $\mathbf{Syn}_{C}$. □

## ACKNOWLEDGMENTS

This research was supported by the NWO Veni grant VI.Veni.202.124 and the ERC project FoRECAST.

The first author gratefully acknowledges support from the Fields Institute for Research in Mathematical Sciences through a Fields Research Fellowship in 2023, as well as ongoing support from the Centre for Mathematics of the University of Coimbra (CMUC) — UIDB/00324/2020, funded by the Portuguese Government through FCT/MCTES. The author also expresses sincere gratitude to Henrique Bursztyn and the Instituto Nacional de Matemática Pura e Aplicada (IMPA) for their generous hospitality.

The second author acknowledges support from Google DeepMind.

## REFERENCES

[1] M. Abadi and G. Plotkin. A simple differentiable programming language. In *Proc. POPL 2020.* ACM, 2020.

[2] M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *International Conference on Foundations of Software Science and Computation Structures*, pages 23–38. Springer, 2003.

[3] J. Adamek and V. Koubek. Least fixed point of a functor. *J. Comput. Syst. Sci.*, 19:163–178, 1979.

[4] J. Adámek and J. Rosický. How nice are free completions of categories? *Topology Appl.*, 273:24, 2020. Id/No 106972.

[5] M. Barr and C. Wells. Toposes, triples and theories. *Repr. Theory Appl. Categ.*, 2005(12):1–288, 2005.

[6] T. Beck and H. Fischer. The if-problem in automatic differentiation. *Journal of Computational and Applied Mathematics*, 50(1-3):119–131, 1994.
[7] R. Blackwell, G. M. Kelly, and A. J. Power. Two-dimensional monad theory. *J. Pure Appl. Algebra*, 59(1):1–41, 1989.
[8] S. Bloom and Z. Ésik. *Iteration Theories - The Equational Logic of Iterative Processes.* EATCS Monographs on Theoretical Computer Science. Springer, 1993.
[9] A. Burroni. Higher-dimensional word problems with applications to equational logic. *Theor. Comput. Sci.*, 115(1):43–62, 1993.
[10] A. Carboni, S. Lack, and R. F. C. Walters. Introduction to extensive and distributive categories. *J. Pure Appl. Algebra*, 84(2):145–158, 1993.
[11] E. Dubuc. *Kan extensions in enriched category theory.* Lecture Notes in Mathematics, Vol. 145. Springer-Verlag, Berlin-New York, 1970.
[12] C. Elliott. The simple essence of automatic differentiation. *Proc. ACM Program. Lang.*, 2(ICFP):70:1–70:29, 2018.
[13] M. P. Fiore. *Axiomatic domain theory in categories of partial maps*, volume 14. Cambridge University Press, 2004.
[14] M. Fiore and G. Plotkin. An axiomatisation of computationally adequate domain theoretic models of fpc. In *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 92–102. IEEE, 1994.
[15] S. Goncharov, C. Rauch, and L. Schröder. Unguarded recursion on coinductive resumptions. *Electronic Notes in Theoretical Computer Science*, 319:183–198, 2015.
[16] M. Huot, A. K. Lew, V. K. Mansinghka, and S. Staton. $\omega$PAP Spaces: Reasoning Denotationally About Higher-Order, Recursive Probabilistic and Differentiable Programs. *arXiv e-prints*, page arXiv:2302.10636, February 2023.
[17] P. Johann and N. Ghani. Initial algebra semantics is enough! In S. R. D. Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2007.
[18] P. Johnstone. *Sketches of an elephant: A topos theory compendium*, volume 2. Oxford University Press, 2002.
[19] G. M. Kelly. The basic concepts of enriched category theory. Seminarber. Fachber. Math., Fernuniv. Hagen 9, 245 p. (1981)., 1981.
[20] S. Lack. Non-canonical isomorphisms. *J. Pure Appl. Algebra*, 216(3):593–597, 2012.
[21] J. Lambek. A fixpoint theorem for complete categories. *Math. Z.*, 103:151–161, 1968.
[22] J. M. Lee. *Introduction to smooth manifolds*, volume 218 of *Grad. Texts Math.* New York, NY: Springer, 2nd revised ed edition, 2013.
[23] P. B. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Information and computation*, 185(2):182–210, 2003.
[24] P. Levy. *Call-by-push-value: A Functional/imperative Synthesis*, volume 2. Springer Science & Business Media, 2012.
[25] F. Lucatelli Nunes. On biadjoint triangles. *Theory Appl. Categ.*, 31:217–256, 2016.
[26] F. Lucatelli Nunes. Pseudomonads and Descent, PhD Thesis (Chapter 1). Preprint, arXiv:1802.01767 [math.CT], 2018.
[27] F. Lucatelli Nunes, G. Plotkin, and M. Vákár. Predicate transformers via chad: a case study of iteration. *To Appear on ArXiv*, 2025.
[28] F. Lucatelli Nunes and R. Prezado. Generalized multicategories: change-of-base, embedding, and descent. 2023.
[29] F. Lucatelli Nunes and L. Sousa. On lax epimorphisms and the associated factorization. *J. Pure Appl. Algebra*, 226(12):27, 2022. Id/No 107126.
[30] F. Lucatelli Nunes and M. Vákár. CHAD for expressive total languages. *Math. Struct. Comput. Sci.*, 33(4-5):311–426, 2023.
[31] F. Lucatelli Nunes and M. Vákár. Monoidal closure of grothendieck constructions via $\Sigma$-tractable monoidal structures and dialectica formulas. 2024.
[32] F. Lucatelli Nunes and M. Vákár. Parameterized initial algebras in a two-category: Inductive types, iteration and recursion. *To Appear on ArXiv*, 2025.
[33] F. Lucatelli Nunes. Freely generated $n$-categories, coinserters and presentations of low dimensional categories. *arXiv e-prints*, page arXiv:1704.04474, April 2017.
[34] F. Lucatelli Nunes. On lifting of biadjoints and lax algebras. *Categ. Gen. Algebr. Struct. Appl.*, 9(1):29–58, 2018.
[35] F. Lucatelli Nunes. Pseudo-Kan extensions and descent theory. *Theory Appl. Categ.*, 33:390–444, 2018.
[36] F. Lucatelli Nunes. Pseudoalgebras and non-canonical isomorphisms. *Appl. Categ. Struct.*, 27(1):55–63, 2019.
[37] F. Lucatelli Nunes, R. Prezado, and M. Vákár. Free extensivity via distributivity, 2025.
[38] F. Lucatelli Nunes and M. Vákár. Logical Relations for Partial Features and Automatic Differentiation Correctness. *arXiv e-prints*, page arXiv:2210.08530, October 2022.
[39] F. Lucatelli Nunes and M. Vákár. Automatic differentiation for ML-family languages: correctness via logical relations. *Math. Struct. Comput. Sci.*, 34(8):747–806, 2024.
[40] F. Lucatelli Nunes and M. Vákár. Free Doubly-Infinitary Distributive Categories are Cartesian Closed. 2024.
[41] F. Métayer. Strict $\omega$-categories are monadic over polygraphs. *Theory Appl. Categ.*, 31:799–806, 2016.

[42] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989.

[43] G. D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5:223–255, 1978.

[44] R. Prezado. On effective descent $\mathcal{V}$-functors and familial descent morphisms. *J. Pure Appl. Algebra*, 228(5):21, 2024. Id/No 107597.

[45] L. Santocanale. $\mu$-bicomplete categories and parity games. *Theor. Inform. Appl.*, 36(2):195–227, 2002.

[46] A. Simpson and G. Plotkin. Complete axioms for categorical fixed-point operators. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 30–41. IEEE Computer Society, 2000.

[47] T. Smeding and M. Vákár. Efficient dual-numbers reverse AD via well-known program transformations. *Proc. ACM Program. Lang.*, 7(POPL):1573–1600, 2023.

[48] T. Smeding and M. Vákár. Efficient CHAD. *Proc. ACM Program. Lang.*, 8(POPL):1060–1088, 2024.

[49] S. Staton. Freyd categories are enriched lawvere theories. In J. Power and C. Wingfield, editors, *Proceedings of the Workshop on Algebra, Coalgebra and Topology, WACT 2013, Bath, UK, March 1, 2013*, volume 303 of *Electronic Notes in Theoretical Computer Science*, pages 197–206. Elsevier, 2013.

[50] R. Street. Limits indexed by category-valued 2-functors. *J. Pure Appl. Algebra*, 8:149–181, 1976.

[51] M. Vákár. Reverse AD at higher types: Pure, principled and denotationally correct. 12648:607–634, 2021.

[52] M. Vákár. In search of effectful dependent types. *arXiv preprint arXiv:1706.07997*, 2017.

[53] M. Vákár. Denotational correctness of forward-mode automatic differentiation for iteration and recursion. *arXiv preprint arXiv:2007.05282*, 2020.

[54] M. Vákár and T. Smeding. CHAD: combinatory homomorphic automatic differentiation. *ACM Trans. Program. Lang. Syst.*, 44(3):20:1–20:49, 2022.

[55] D. Verity. Enriched categories, internal categories and change of base. *Repr. Theory Appl. Categ.*, 2011(20):266, 2011.

## A SOURCE LANGUAGE AS A STANDARD CALL-BY-VALUE LANGUAGE

In this appendix, we present a basic call-by-value language that serves as the foundation for our source language. Depending on the semantics, we may allow non-termination (resulting in a partial language) – with the evaluation strategy being call-by-value – or disallow it (resulting in a total language where evaluation strategy does not matter). Since these definitions are quite standard, we include them here for completeness.

Our language is built upon ground types $\mathbf{real}^n$ ($n \in \mathbb{N}$), with sets of primitive operations $\mathrm{op} \in \mathrm{Op}^{\mathbf{n}}_{\mathbf{m}}$, for each $(k_1, k_2) \in \mathbb{N} \times \mathbb{N}$ and each $(\mathbf{n}, \mathbf{m}) = \left((n_1, \ldots, n_{k_1}), (m_1, \ldots, m_{k_2})\right) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2}$. The types $\tau, \sigma, \rho$, and computations $t, s, r$ of our language are as follows.
We use the following syntactic sugar for the $i$-th projection out of a tuple:

| | | | | | |
|---|---|---|---|---|---|
| $\tau, \sigma, \rho$ | $::=$ | types | $\mid$ | $\mathbf{1} \mid \tau_1 \times \cdots \times \tau_n$ | products |
| | $\mid\ \mathbf{real}^n$ | real arrays | $\mid$ | $\mathbf{0} \mid \tau_1 \sqcup \cdots \sqcup \tau_n$ | sums |
| $t, s, r$ | $::=$ | computations | | | |
| | $\mid\ \mathrm{op}(t_1, \ldots, t_n)$ | operation | $\mid$ | $\mathbf{case}\, r\, \mathbf{of} \left\{ \begin{array}{l} \mathbf{in}_1\, x_1 \to t_1 \\ \mid \cdots \\ \mid \mathbf{in}_n\, x_n \to t_n \end{array} \right\}$ | sum match |
| | $\mid\ x, y, z$ | variables | | | |
| | $\mid\ \mathbf{let}\, x = t\, \mathbf{in}\, s$ | sequencing | $\mid$ | $\langle t_1, \ldots, t_n \rangle$ | tuples |
| | $\mid\ \mathbf{in}_1\, t \mid \cdots \mid \mathbf{in}_n\, t$ | sum inclusions | $\mid$ | $\mathbf{case}\, s\, \mathbf{of}\, \langle x_1, \ldots, x_n \rangle \to t$ | prod. match |
| | | | $\mid$ | $\mathbf{iterate}(x \to t)$ | iteration |

Fig. A.1. A grammar for the types and terms of the source language.

$$\mathbf{pr}_i\, t \stackrel{\text{def}}{=} \mathbf{case}\, t\, \mathbf{of}\, \langle x_1, \ldots, x_n \rangle \to x_i.$$

The computations of our source language are typed according to the rules of Fig. A.2. We consider the standard call-by-value $\beta\eta$-equational theory of [42] for our language, which we list in Fig. A.4. To present this equational theory, we distinguish a subset of computations that we call *(complex) values* $v, w, u$ to consist of those computations that do not involve any iteration constructs or primitive operations $\mathrm{op} \in \mathrm{Op}$ (seeing that we are working with partial operations[12]). Effectively, the (complex) values comprise the total fragment of our language.

Observe that a more general iteration construct with typing rule

$$\frac{\Gamma, x : \sigma \vdash t : \sigma \sqcup \tau}{\Gamma, x : \sigma \vdash \mathbf{iterate}_\Gamma(x \to t) : \tau}$$

is derivable. Indeed, for $\Gamma = x_1 : \rho_1, \ldots, y_n : \rho_n$, we define $\mathbf{iterate}_\Gamma(x \to t)$ as

$\mathbf{let}\, z = \langle y_1, \ldots, y_n, x \rangle\, \mathbf{in}$
$\mathbf{iterate}(z \to \mathbf{case}\, z\, \mathbf{of}\, \langle \langle y_1, \ldots, y_n, x \rangle \rangle \to \mathbf{case}\, t\, \mathbf{of}\, \{\mathbf{in}_1\, u \to \mathbf{in}_1\, \langle y_1, \ldots, y_n, u \rangle \mid \mathbf{in}_2\, v \to \mathbf{in}_2\, v\})$

Following the same viewpoint as in [39, 53], we do not impose additional equations on the iteration construct, unlike in [8, 15]. This leads us to what we refer to as *free iteration.* We have chosen to work at this level of generality because no equations are needed to establish the correctness proof of CHAD.

We can think of this syntax as the freely generated iterative Freyd category with $\mathbf{Syn} = \left(\mathbf{Syn}_V, \mathbf{Syn}_C, \mathbf{Syn}_J, \mathbf{Syn}_\otimes, \mathbf{Syn}_{\mathrm{it}}\right)$ on our total and partial operations op. Concretely,

- $\mathbf{Syn}_V$ and $\mathbf{Syn}_C$ both have types $\tau, \sigma, \rho, \ldots$ as objects;

[12] In case we also included total operations, we could allow these in complex values.

$$\frac{\{\Gamma \vdash t_i : \mathbf{real}^{n_j}\}_{j=1}^{k_1} \quad \Big((k_1, k_2) \in \mathbb{N},\ \big((n_j), (m_i)\big) \in \mathbb{N}^{k_1} \times \mathbb{N}^{k_2},\ \mathrm{op} \in \mathrm{Op}^{(n_j)}_{(m_i)}\Big)}{\Gamma \vdash \mathrm{op}(t_1, \ldots, t_{k_1}) : \mathbf{real}^{m_1} \sqcup \cdots \sqcup \mathbf{real}^{m_{k_2}}}$$

$$\frac{((x : \tau) \in \Gamma)}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash t : \sigma \quad \Gamma, x : \sigma \vdash s : \tau}{\Gamma \vdash \mathbf{let}\, x =\ t\, \mathbf{in}\, s : \tau} \qquad \frac{\Gamma \vdash t : \tau_i}{\Gamma \vdash \mathbf{in}_i\, t : \tau_1 \sqcup \cdots \sqcup \tau_n} (i = 1, \ldots, n)$$

$$\frac{\Gamma \vdash r : \sigma_1 \sqcup \cdots \sqcup \sigma_n \quad \Gamma, x_1 : \sigma_1 \vdash t_1 : \tau \quad \cdots \quad \Gamma, x_n : \sigma_n \vdash t_n : \tau}{\Gamma \vdash \mathbf{case}\, r\, \mathbf{of}\, \{\mathbf{in}_1\, x_1 \to t_1 \mid \cdots \mid \mathbf{in}_n\, x_n \to t_n\} : \tau}$$

$$\frac{\Gamma \vdash t_1 : \tau_1 \quad \cdots \quad \Gamma \vdash t_n : \tau_n}{\Gamma \vdash \langle t_1, \ldots, t_n \rangle : \tau_1 \times \cdots \times \tau_n} \qquad \frac{\Gamma \vdash r : \tau_1 \times \cdots \times \tau_n \quad \Gamma, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash t : \sigma}{\Gamma \vdash \mathbf{case}\, r\, \mathbf{of}\, \langle x_1, \ldots, x_n \rangle \to t : \sigma}$$

Fig. A.2. Typing rules for our standard call-by-value language constructed over the ground types $\mathbf{real}^n$ and the primitive operations in Op.

$$\frac{x : \sigma \vdash t : \tau \sqcup \sigma}{x : \sigma \vdash \mathbf{iterate}(x \to t) : \tau}$$

Fig. A.3. Typing rules for iteration on the top of our standard call-by-value language.

$$\mathbf{let}\, x =\ v\, \mathbf{in}\, t = t[{}^{v}\!/\!_{x}] \qquad \mathbf{let}\, y =\ (\mathbf{let}\, x =\ t\, \mathbf{in}\, s)\, \mathbf{in}\, r = \mathbf{let}\, x =\ t\, \mathbf{in}\, (\mathbf{let}\, y =\ s\, \mathbf{in}\, r)$$

$$\mathbf{case}\, \mathbf{in}_i\, s\, \mathbf{of}\, \{\mathbf{in}_1\, x_1 \to t_1 \mid \cdots \mid \mathbf{in}_n\, x_n \to t_n\} = t_i[{}^{s}\!/\!_{x_i}] \qquad t[{}^{s}\!/\!_{z}] \overset{\#x_1,\ldots,x_n}{=} \mathbf{case}\, s\, \mathbf{of} \left\{ \begin{array}{c} \mathbf{in}_1\, x_1 \to t[{}^{\mathbf{in}_1\, x_1}\!/\!_{z}] \mid \cdots \\ \mid \mathbf{in}_n\, x_n \to t[{}^{\mathbf{in}_n\, x_n}\!/\!_{z}] \end{array} \right\}$$

$$\mathbf{case}\, \langle v_1, \ldots, v_n \rangle\, \mathbf{of}\, \langle x_1, \ldots, x_n \rangle \to t = t[{}^{v_1}\!/\!_{x_1}, \ldots, {}^{v_n}\!/\!_{x_n}] \qquad t[{}^{v}\!/\!_{z}] \overset{\#x_1,\ldots,x_n}{=} \mathbf{case}\, v\, \mathbf{of}\, \langle x_1, \ldots, x_n \rangle \to t[{}^{\langle x_1,\ldots,x_n \rangle}\!/\!_{z}]$$

$$\mathbf{let}\, x_1 =\ t_1\, \mathbf{in}\, \cdots \mathbf{let}\, x_n =\ t_n\, \mathbf{in}\, \langle x_1, \ldots, x_n \rangle \overset{\#x_1,\ldots,x_n}{=} \langle t_1, \ldots, t_n \rangle$$

$$\mathbf{let}\, x =\ t\, \mathbf{in}\, \mathbf{case}\, x\, \mathbf{of}\, \langle s_1, \ldots, s_n \rangle \to r \overset{\#x}{=} \mathbf{case}\, t\, \mathbf{of}\, \langle s_1, \ldots, s_n \rangle \to r$$

$$\mathbf{let}\, x_1 =\ t_1\, \mathbf{in}\, \cdots \mathbf{let}\, x_n =\ t_n\, \mathbf{in}\, \mathrm{op}(x_1, \ldots, x_n) \overset{\#x_1,\ldots,x_n}{=} \mathrm{op}(t_1, \ldots, t_n)$$

Fig. A.4. Basic $\beta\eta$-equational theory for our language. We write $\overset{\#x_1,\ldots,x_n}{=}$ to indicate that the variables are fresh in the left hand side. In the top right rule, $x$ may not be free in $r$. Equations hold on pairs of computations of the same type. Note that we do not include equations for iteration, as they are not needed for our development.

- morphisms $\tau \to \sigma$ are (complex) values $x : \tau \vdash v : \sigma$ (in the case of $\mathbf{Syn}_V$) or computations $x : \tau \vdash t : \sigma$ (in the case of $\mathbf{Syn}_C$) modulo $\alpha$-renaming of bound variables and $\beta\eta$-equivalence;
- identities are the equivalence class $[x : \tau \vdash x : \tau]$;
- composition of $[x : \tau \vdash t : \sigma]$ and $[y : \sigma \vdash s : \rho]$ is given by $[x : \tau \vdash \mathbf{let}\, y =\ t\, \mathbf{in}\, s : \sigma]$;
- $\mathbf{Syn}_J$ is the inclusion of complex values in the larger set of computations;
- $\mathbf{Syn}_\otimes([x : \tau \vdash v : \tau'], [y : \sigma \vdash t : \sigma']) = [z : \tau \times \sigma \vdash \mathbf{case}\, z\, \mathbf{of}\, \langle x, y \rangle \to \langle t, s \rangle : \tau' \times \sigma']$;
- $\mathbf{Syn}_{\mathrm{it}}([x : \sigma \vdash t : \tau \sqcup \sigma]) = [\mathbf{iterate}(x \to t)]$.

**Syn** has the following universal property:

Given any iterative Freyd category $(\mathcal{V}, C, j, \otimes, \mathtt{it})$, for each pair $(K, \mathtt{h})$ where $K = (K_n)_{n\in\mathbb{N}}$ is a family of objects in $C$ and $\mathtt{h} = \left(h_{\mathrm{op}}\right)_{\mathrm{op}\in\mathrm{Op}}$ is a consistent family of morphisms in $C$, there is an unique iterative Freyd category morphism

$$\left(H, \hat{H}\right) : \mathbf{Syn} \to (\mathcal{V}, C, j, \otimes, \mathtt{it})$$

such that $\hat{H}(\mathbf{real}^n) = K_n$ and $\hat{H}(\mathrm{op}) = h_{\mathrm{op}}$ for any $n \in \mathbb{N}$ and any primitive operation $\mathrm{op} \in \mathrm{Op}$.

## B LINEAR $\lambda$-CALCULUS AS THE TARGET LANGUAGE

We describe the target language for our AD transformation. Its terms and types are presented in Figure B.1. We shade out the the constructors **roll** $t$ for inductive types, as we only need the corresponding eliminators.

| | | | |
|---|---|---|---|
| $\underline{\tau}, \underline{\sigma}, \underline{\rho}$ | $::=$ | | linear types |
| | $\mid$ | $\underline{\mathbf{reals}}^n$ | real array |
| | $\mid$ | $\underline{\mathbf{1}}$ | linear unit type |
| | $\mid$ | $\underline{\tau}_1 \times \cdots \times \underline{\tau}_n$ | linear product |
| | $\mid$ | $\mathbf{case}\, t\, \mathbf{of}\, \{\mathbf{in}_1\, x_1 \to \underline{\tau}_1 \mid \cdots \mid \mathbf{in}_n\, x_n \to \underline{\tau}_n\}$ | case distinction |
| $\tau, \sigma, \rho$ | $::=$ | | Cartesian types |
| | | $\ldots$ | as in Fig. A.1 |
| | $\mid$ | $\underline{\tau} \multimap \underline{\sigma}$ | linear function |
| | $\mid$ | $\Sigma x : \tau.\sigma$ | dependent pair |
| $t, s, r$ | $::=$ | | computations |
| | | $\ldots$ | as in Fig. A.1 |
| | $\mid$ | $\mathsf{lop}(t_1, \ldots, t_k; s)$ | linear operation |
| | $\mid$ | $\mathsf{v}$ | linear identifier |
| | $\mid$ | $\mathbf{let}\, \mathsf{v} =\, t\, \mathbf{in}\, s$ | linear let-binding |
| | $\mid$ | $\mathbf{pr}_i\, t$ | projection |
| | $\mid$ | $\underline{\lambda}\mathsf{v}.t \mid t \bullet s$ | abstraction/application |
| | $\mid$ | $\underline{0} \mid t + s$ | monoid structure |
| | $\mid$ | $\mathbf{roll}\, t$ | inductive type introduction |
| | $\mid$ | $\mathbf{fold}\, t\, \mathbf{with}\, \mathsf{v} \to s$ | inductive type elimination |

Fig. B.1. A grammar for the types and terms of the target language, extending that of Fig. A.1.

These terms are typed according to the rules of Figures B.2 and B.3. We think of Cartesian types as denoting (families of) sets and of linear types as denoting (families of) commutative monoids.

Fig. A.4 and B.5 display the equational theory we consider for the terms and types, which we call $(\alpha)\beta\eta$+-equivalence.

We find it helpful to think of the target language from a categorical point of view as a Freyd indexed category[13] $\mathbf{LSyn} : \mathbf{CSyn}^{op} \to \mathrm{Cat}$, concretely:

- similar to the construction of **Syn** in Appendix A, we have an iterative Freyd category $\mathbf{CSyn} = \left(\mathbf{CSyn}_V, \mathbf{CSyn}_C, \mathbf{CSyn}_J, \mathbf{CSyn}_\otimes, \mathbf{CSyn}_{\mathtt{it}}\right)$; $\mathbf{CSyn}_V$ and $\mathbf{CSyn}_C$ have (closed) Cartesian types as

[13] By this, we mean an iterative Freyd category internal to the 2-category of (strict) indexed categories.

$$\frac{}{\Gamma;\mathsf{v}:\underline{\tau}\vdash \mathsf{v}:\underline{\tau}} \qquad \frac{\Gamma;\mathsf{v}:\underline{\tau}\vdash t:\underline{\sigma} \quad \Gamma;\mathsf{v}:\underline{\sigma}\vdash s:\underline{\rho}}{\Gamma;\mathsf{v}:\underline{\tau}\vdash \mathbf{let}\,\mathsf{v} = \ t\,\mathbf{in}\,s:\underline{\rho}} \qquad \frac{\Gamma\vdash t:\tau \quad \Gamma,x:\tau;\mathsf{v}:\underline{\sigma}\vdash s:\underline{\rho}}{\Gamma;\mathsf{v}:\underline{\sigma}[{}^{t}/_{x}]\vdash \mathbf{let}\,x = \ t\,\mathbf{in}\,s:\underline{\rho}[{}^{t}/_{x}]}$$

$$\frac{\{\Gamma\vdash t_i:\mathbf{real}^{n_i}\}_{i=1}^{k} \quad \Gamma;\mathsf{v}:\underline{\tau}\vdash s:\mathbf{LDom}(\mathsf{lop}) \quad (\mathsf{lop}\in\mathsf{LOp}^{m_1,\ldots,m_r}_{n_1,\ldots,n_k;n'_1,\ldots,n'_l})}{\Gamma;\mathsf{v}:\underline{\tau}\vdash \mathsf{lop}(t_1,\ldots,t_k;s):\mathbf{CDom}(\mathsf{lop})}$$

$$\frac{\Gamma;\mathsf{v}:\underline{\tau}\vdash t_1:\underline{\tau}_1 \quad \cdots \quad \Gamma;\mathsf{v}:\underline{\tau}\vdash t_n:\underline{\tau}_n}{\Gamma;\mathsf{v}:\underline{\tau}\vdash\langle t_1,\ldots,t_n\rangle:\underline{\tau}_1\times\cdots\times\underline{\tau}_n} \qquad \frac{\Gamma;\mathsf{v}:\underline{\tau}\vdash t:\underline{\sigma}_1\times\cdots\times\underline{\sigma}_n}{\Gamma;\mathsf{v}:\underline{\tau}\vdash \mathbf{pr}_i\,t:\underline{\sigma}_i}(i=1,\ldots,n)$$

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

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

$$\frac{\Gamma\vdash t:\tau_1\sqcup\cdots\sqcup\tau_n \quad \left\{x_i:\tau_i;\mathsf{v}:\underline{\sigma}[{}^{\mathbf{in}_i\,x_i}/_{x}]\vdash r_i:\underline{\rho}[{}^{\mathbf{in}_i\,x_i}/_{x}]\right\}_{1\le i\le n}}{\Gamma;\mathsf{v}:\underline{\sigma}[{}^{t}/_{x}]\vdash \mathbf{case}\,t\,\mathbf{of}\,\{\mathbf{in}_1\,x_1\to r_1\mid\cdots\mid\mathbf{in}_n\,x_n\to r_n\}:\underline{\rho}[{}^{t}/_{x}]}$$

$$\frac{x:\sigma\vdash t:\tau\sqcup\sigma \quad \Gamma,x:\sigma;\mathsf{v}:\underline{\rho}\vdash s:\mathbf{case}\,t\,\mathbf{of}\,\{\mathbf{in}_1\,y\to\underline{\tau}\mid\mathbf{in}_2\,x\to\underline{\tau}[{}^{\mathrm{iterate}\,(x\to t)}/_{y}]\}}{\Gamma,x:\sigma;\mathsf{v}:\underline{\rho}\vdash\mathbf{roll}\,s:\underline{\tau}[{}^{\mathrm{iterate}\,(x\to t)}/_{y}]}$$

Fig. B.2. Typing rules for the AD target language that we consider on top of the rules of Fig. A.2 and B.3.

$$\frac{\Gamma\vdash t:\tau \quad \Gamma,x:\tau\vdash s:\rho}{\Gamma\vdash\mathbf{let}\,x = \ t\,\mathbf{in}\,s:\rho[{}^{t}/_{x}]}$$

$$\frac{\Gamma\vdash t:\tau \quad \Gamma\vdash s:\sigma[{}^{t}/_{x}]}{\Gamma\vdash\langle t,s\rangle:\Sigma x:\tau.\sigma} \qquad \frac{\Gamma\vdash r:\Sigma x:\tau.\sigma \quad \Gamma,x:\tau,y:\sigma\vdash t:\rho[{}^{\langle x,y\rangle}/_{z}]}{\Gamma\vdash\mathbf{case}\,r\,\mathbf{of}\,\langle x,y\rangle\to t:\rho[{}^{r}/_{z}]}$$

Fig. B.3. Typing rules for the AD target language that we consider on top of the rules of Fig. A.2 and B.2. The new dependently typed rule for let-bindings now replaces our previous simply typed rule.

$$\frac{\begin{array}{c} x:\sigma\vdash t:\tau\sqcup\sigma \\ x:\sigma;\mathsf{v}:\underline{\rho}\vdash s:\underline{\tau}[{}^{\mathbf{iterate}\,(x\to t)}/_{y}] \quad x:\sigma;\mathsf{v}:\mathbf{case}\,t\,\mathbf{of}\,\{\mathbf{in}_1\,y\to\underline{\tau}\mid\mathbf{in}_2\,x\to\underline{\rho'}\}\vdash r:\underline{\rho'}\end{array}}{x:\sigma;\mathsf{v}:\underline{\rho}\vdash\mathbf{fold}\,s\,\mathbf{with}\,\mathsf{v}\to r:\underline{\rho'}}$$

Fig. B.4. Typing rules for iterative foldersfor the target language on top of the rules of A.2, B.3 and B.2.

$$\mathbf{let}\,\mathsf{v} = t\,\mathbf{in}\,s = s[{}^{t}/_{\mathsf{v}}]$$

$$(\underline{\lambda}\mathsf{v}.t) \bullet s = t[{}^{s}/_{\mathsf{v}}] \qquad t = \underline{\lambda}\mathsf{v}.t \bullet \mathsf{v}$$

$$t + \underline{0} = t \qquad \underline{0} + t = t \qquad (t+s)+r = t+(s+r) \qquad t+s = s+t$$

$$(\Gamma;\mathsf{v}:\underline{\tau} \vdash t : \underline{\sigma}) \text{ implies } t[{}^{\underline{0}}/_{\mathsf{v}}] = \underline{0} \qquad (\Gamma;\mathsf{v}:\underline{\tau} \vdash t : \underline{\sigma}) \text{ implies } t[{}^{s+r}/_{\mathsf{v}}] = t[{}^{s}/_{\mathsf{v}}] + t[{}^{r}/_{\mathsf{v}}]$$

$$\mathbf{case}\,\mathbf{in}_i\,s\,\mathbf{of}\,\{\mathbf{in}_1\,x_1 \to \underline{\tau}_1 \mid \cdots \mid \mathbf{in}_n\,x_n \to \underline{\tau}_n\} = \underline{\tau}_i[{}^{s}/_{x_i}]$$

$$\underline{\tau}[{}^{s}/_{y}] \overset{\#x_1,\ldots,x_n}{=} \mathbf{case}\,s\,\mathbf{of}\,\{\mathbf{in}_1\,x_1 \to \underline{\tau}[{}^{\mathbf{in}_1\,x_1}/_{y}] \mid \cdots \mid \mathbf{in}_n\,x_n \to \underline{\tau}[{}^{\mathbf{in}_n\,x_n}/_{y}]\}$$

$$\mathbf{pr}_i\,\langle t_1,\ldots,t_n\rangle = t_i \qquad t = \langle \mathbf{pr}_1\,t,\ldots,\mathbf{pr}_n\,t\rangle$$

Fig. B.5. Equational rules for the idealised, linear AD language, which we use on top of the rules of Fig. A.4. In addition to standard $\beta\eta$-rules for $\multimap$-types, we add rules making $(\underline{0},+)$ into a commutative monoid on the terms of each linear type as well as rules which say that terms of linear types are homomorphisms in their linear Equations hold on pairs of terms of the same type/types of the same kind. As usual, we only distinguish terms up to $\alpha$-renaming of bound variables. Note that we do not include equations for inductive types, just as we do not for iteration.

objects and $\beta\eta+$-equivalence classes of values and computations as morphisms, respectively; we can observe that **Syn** is a full Freyd subcategory of **CSyn**;

- a (strict) indexed category $\mathbf{LSyn}_C : \mathbf{CSyn}_C^{op} \to \mathrm{Cat}$ (from which we can also define $\mathbf{LSyn}_V : \mathbf{CSyn}_V^{op} \to \mathrm{Cat}$ as $\mathbf{LSyn}_C \circ \mathbf{CSyn}_J^{op}$) where the objects of $\mathbf{LSyn}_C(\tau)$ objects are $(\alpha)\beta\eta+$-equivalence classes $[\underline{\sigma}]$ of linear (dependent) types $x:\tau \vdash \underline{\sigma}$ and morphisms $[t] : [\underline{\sigma}] \to [\underline{\rho}]$ are $(\alpha)\beta\eta+$-equivalence classes of computations $x:\tau;\mathsf{v}:\underline{\sigma} \vdash t : \underline{\rho}$; identities in $\mathbf{LSyn}_C(\tau)$ are given by the equivalence class of $x:\tau;\mathsf{v}:\underline{\sigma}\vdash \mathsf{v}:\underline{\sigma}$ and composition of $[t] : [\underline{\sigma}_1] \to [\underline{\sigma}_2]$ and $[s] : [\underline{\sigma}_2] \to [\underline{\sigma}_3]$ is given by the equivalence class of $x:\underline{\tau};\mathsf{v}:\underline{\sigma}_1 \vdash \mathbf{let}\,\mathsf{v} = t\,\mathbf{in}\,s : \underline{\sigma}_3$; finally, given $[t] : [\underline{\sigma}] \to [\underline{\rho}]$ in $\mathbf{LSyn}_C(\tau)$ represented by $x:\tau;\mathsf{v}:\underline{\sigma}\vdash t:\underline{\rho}$ and $[s] : [\tau'] \to [\tau]$ in $\mathbf{CSyn}_C$, we define the change of base $\mathbf{LSyn}_C([s])([t])$ as the morphism $[\underline{\sigma}[{}^{s}/_{x}]] \to [\underline{\rho}[{}^{s}/_{x}]]$ in $\mathbf{LSyn}_C(\tau')$ given by the equivalence class of $x':\tau';\mathsf{v}:\underline{\sigma}[{}^{s}/_{x}] \vdash \mathbf{let}\,x = s\,\mathbf{in}\,t : \underline{\rho}[{}^{s}/_{x}]$;
- we observe (†) that $\mathbf{LSyn}_C$ has finite indexed biproducts (hence is enriched over commutative monoids), that $\mathbf{LSyn}_C$ is extensive[14] in the sense that $\mathbf{LSyn}_C([\tau_1 \sqcup \cdots \sqcup \tau_n]) \cong \mathbf{LSyn}_C(\tau_1) \times \cdots \times \mathbf{LSyn}_C(\tau_n)$ and that $\mathbf{LSyn}_C$ is *extensive for iteration* in the sense that, given $x:\sigma\vdash t:\tau\sqcup\sigma$, we have that $\mathbf{LSyn}_C(\mathbf{iterate}(x \to t))$ gives a parameterized weak initial algebra of the functor $\mathbf{LSyn}_C(t) : \mathbf{LSyn}_C(\tau)\times\mathbf{LSyn}_C(\sigma) \cong \mathbf{LSyn}_C(\tau\sqcup\sigma) \to \mathbf{LSyn}(\sigma)$ (using our **fold with** $\to$-constructs);
- similarly, we have a (strict) indexed category $\mathbf{CSyn}'_C : \mathbf{CSyn}_C^{op} \to \mathrm{Cat}$ (from which we can also define $\mathbf{CSyn}'_V : \mathbf{CSyn}_V^{op} \to \mathrm{Cat}$ as $\mathbf{CSyn}'_C \circ \mathbf{CSyn}_J^{op}$) where the objects of $\mathbf{CSyn}'_C(\tau)$ objects are $(\alpha)\beta\eta+$-equivalence classes $[\sigma]$ of Cartesian (dependent) types $x:\tau\vdash\sigma$ and morphisms $[t] : [\sigma] \to [\rho]$ are $(\alpha)\beta\eta+$-equivalence classes of computations $x:\tau, y:\sigma \vdash t:\rho$; identities in $\mathbf{CSyn}'_C(\tau)$ are given by the equivalence class of $x:\tau,y:\sigma\vdash y:\sigma$ and composition of $[t] : [\sigma_1] \to [\sigma_2]$ and $[s] : [\sigma_2] \to [\sigma_3]$ is given by the equivalence

[14]See [30] or [31] for more details on extensive indexed categories.

class of $x : \tau, y : \sigma_1 \vdash \mathbf{let}\, y = t\, \mathbf{in}\, s : \sigma_3$; finally, given $[t] : [\sigma] \to [\rho]$ in $\mathbf{CSyn}'_C(\tau)$ represented by $x : \tau, y : \sigma \vdash t : \rho$ and $[s] : [\tau'] \to [\tau]$ in $\mathbf{CSyn}_C$, we define the change of base $\mathbf{CSyn}'_C([s])([t])$ as the morphism $[\sigma[{}^{s}/_{x}]] \to [\rho[{}^{s}/_{x}]]$ in $\mathbf{CSyn}'_C(\tau')$ given by the equivalence class of $x' : \tau'; \mathsf{v} : \sigma[{}^{s}/_{x}] \vdash \mathbf{let}\, x = s\, \mathbf{in}\, t : \rho[{}^{s}/_{x}]$;
- we observe (‡) that $\mathbf{CSyn}'_C$ forms a model of dependent type theory (in the sense that it satisfies full, faithful, democratic comprehension – see [31, 52]) with strong $\Sigma$-types (in the sense that the display maps/dependent projections are closed under composition) and that $\mathbf{LSyn}_C$ is enriched over $\mathbf{CSyn}'_C$ in the sense that $\mathbf{LSyn}_C(\tau)([\underline{\sigma}], [\underline{\rho}]) \cong \mathbf{CSyn}'_C(\tau)(\mathbf{1}, [\underline{\sigma} \multimap \underline{\rho}])$.

In fact, the indexed categories $\mathbf{LSyn}_C$ and $\mathbf{CSyn}'_C$ happen to be the free indexed categories that satisfy (†) and (‡). Crucially, we now have the following when we take the Grothendieck construction of the indexed categories.

Lemma B.1. $\big(\Sigma_{\mathbf{CSyn}_V}\mathbf{LSyn}_V, \Sigma_{\mathbf{CSyn}_C}\mathbf{LSyn}_C, \Sigma_{\mathbf{CSyn}_V}\mathbf{LSyn}_V \to \Sigma_{\mathbf{CSyn}_C}\mathbf{LSyn}_C, (\times, \times),$ $(\mathbf{iterate}(\to), \mathbf{fold\ \ with}\ \ \to)\big)$ *forms an iterative Freyd category.*