Pré-Publicações do Departamento de Matemática
Universidade de Coimbra
Preprint Number 22–32

AUTOMATIC DIFFERENTIATION FOR ML-FAMILY LANGUAGES: CORRECTNESS VIA LOGICAL RELATIONS

FERNANDO LUCATELLI NUNES AND MATTHIJS VÁKÁR

Abstract: We give a simple, direct and reusable logical relations technique for languages with recursive features and partially defined differentiable functions. We do so by working out the case of Automatic Differentiation (AD) correctness: namely, we present a proof of the dual numbers style AD macro correctness for realistic functional languages in the ML-family. We also show how this macro provides us with correct forward- and *reverse-mode* AD.

The starting point was to interpret a functional programming language in a suitable freely generated categorical structure. In this setting, by the universal property of the syntactic categorical structure, the dual numbers AD macro and the basic $\boldsymbol{\omega}$**Cpo**-semantics arise as structure preserving functors. The proof follows, then, by a novel logical relations argument.

The key to much of our contribution is a powerful monadic logical relations technique for term recursion and recursive types. It provides us with a semantic correctness proof based on a simple approach for denotational semantics, making use only of the very basic concrete model of ω-cpos.

Keywords: Iteration, Term Recursion, Recursive Types, Logical relations, Automatic differentiation, Programming Languages, Functional Programming, Denotational Semantics.
Math. Subject Classification (2020): 68N15, 68N18, 68Q55, 68W30, 18D20, 18A25, 18C15.

Contents

Received October 21, 2022.
This project has received funding via NWO Veni grant number VI.Veni.202.124 as well as the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 895827. This research was supported through the programme "Oberwolfach Leibniz Fellows" by the Mathematisches Forschungsinstitut Oberwolfach in 2022. It was also partially supported by the CMUC, Centre for Mathematics of the University of Coimbra - UIDB/00324/2020, funded by the Portuguese Government through FCT/MCTES.

Introduction

AD and the PL community Automatic differentiation (AD) is a popular technique for computing derivatives of functions implemented by a piece of code, particularly when efficiency, scaling to high dimensions and numerical stability are important. It has been studied in the scientific computing community for many decades and has been heavily used in machine learning for the last decade. In the last years, the programming languages (PL) community has turned towards studying AD from a new perspective. Much progress has been made towards giving a formulation of (forward and) reverse mode AD that:

(1) is simple and purely functional;
(2) scales to the expressive ML-family functional languages that are popular in practice;
(3) admits a simple correctness proof that shows AD computes the derivative;
(4) provably has the correct asymptotic complexity and is performant in practice;
(5) is parallelism preserving.

Our contributions In this paper, we present a simple solution to problems (1)-(3), *our first major contribution.*

We give a proof of the correctness of the reverse and forward mode dual numbers style Automatic Differentiation (AD) in a semantically unified way, making use only of the very simple concrete denotational model of ω-cpos.

A key challenge in achieving the correctness proofs of this paper is to have sufficiently strong categorical logical relations techniques for reasoning about partially defined differentiable functions, and recursive types. To that end, we develop a novel monadic logical relations construction making no use of sheaf-theoretical methods as well as a novel general logical relations technique for recursive types, *our second major contribution.*

We refer to the companion paper [37] for a performant implementation of the dual numbers reverse-mode AD technique proved correct in the present paper. It shows that it efficiently differentiates most of Haskell98, contributing towards point (4). We are currently pursuing parallelism preservation (point (5)) for this AD technique and we plan to present it in future work.

In our work, we ensure to keep all constructions sufficiently simple such that they can easily be generalized to more advanced AD algorithms such as CHAD [44, 45, 26], which is one of our key motivations for this work.

Why care and why is this difficult? Given the central role that AD plays in modern scientific computing and machine learning, the ideal of differential programming has been emerging [29, 34]: compilers for general purpose programming languages should provide built-in support for automatic differentiation of any programs written in the language. Such general purpose programming languages tend to include many language features, however, which we then need to be able to differentiate. What a correct and efficient notion of derivative is of such features might not be so straightforward as they often go beyond what is studied in traditional calculus. In this paper we focus on the challenge posed, in particular, by partial language features: partial primitive operations, lazy conditionals on real numbers, iteration, recursion and recursive types.

Partial primitive operations are certainly key. Indeed, even the basic operations of division and logarithm are examples. (Lazy) conditionals on real numbers are useful in practice for pasting together various existing smooth functions, as basic example being the ReLU function

$$ReLU(x) \overset{\text{def}}{=} \mathbf{if}\, x\, \mathbf{then}\, 0\, \mathbf{else}\, x \ = \mathbf{case}\,(\mathbf{sign}\ x)\,\mathbf{of}\,\{\mathbf{inl}\ _{-} \to 0 \mid \mathbf{inr}\ _{-} \to x\},$$

which is a key component of many neural networks. They are also frequently used in probabilistic programming to paste together density functions of different distributions [4]. People have long studied the subtle issue of how one should algorithmically differentiate such functions with "kinks" under the name of *the if-problem in automatic differentiation* [3]. Our solution is the one also employed by [1]: to treat the functions as semantically undefined at their kinks (at $x = 0$ in the case of $ReLU(x)$). This is justified given how coarse the semantic treatment of floating point numbers as real numbers is already. Our semantics based on partial functions defined on real numbers is sufficient to prove many high-level correctness properties. However, like any semantics based on real numbers, it fails to capture many of the low-level subtleties introduced by the floating point implementation. Our key insight that we use to prove correctness of AD of partial programs is to construct a suitable lifting of the partiality monad to a variant of [19]'s category of $\mathbb{R}^k$-indexed logical relations used to relate programs to their derivatives. This particular monad lifting for derivatives of partial functions can be seen as our solution to the if-problem in AD.

Similarly, iteration constructs, or while-loops, are necessary for implementing iterative algorithms with dynamic stopping criteria. Such algorithms are frequently used in programs that AD is applied to. For example, AD is applied to iterative differential equation solvers to perform Bayesian inference in SIR models. This technique played a key role in modelling the Covid19-pandemic [14]. For similar reasons, AD through iterative differential equation solvers is important for probabilistic modelling of pharmacokinetics [42]. Other common use-cases of iterative algorithms that need to be AD'ed are eigen-decompositions and algebraic equation solvers, such as those employed in Stan [7]. Finally, iteration gives a convenient way of achieving numerically stable approximations to complex functions (such as the Conway-Maxwell-Poisson density function [17]). The idea is to construct, using iteration, a Taylor approximation that terminates once the next term in the series causes floating-point underflow. Indeed, for a function whose i-th terms in the Taylor expansion can be represented by a program

$$i : \mathbf{int}, x : \mathbf{real} \vdash t(i, x) : \mathbf{real},$$

we would define the underflow-truncated Taylor series by

$$\mathbf{iterate}\left(\begin{array}{l}\mathbf{case}\, x\, \mathbf{of}\, \langle x_1, x_2\rangle \to \mathbf{let}\, y = \; t(x_1, x_2)\, \mathbf{in} \\ \mathbf{case}\; -c < y < c\, \mathbf{of}\, \{\mathbf{inl}\, \_ \to \mathbf{inr}\, x_2 \mid \mathbf{inr}\, \_ \to \mathbf{inl}\, \langle x_1 + 1, x_2 + y\rangle\})\end{array}\right)\mathbf{from}\, x = \langle 0, 0\rangle,$$

where c is a cut-off for floating-point underflow.

Next, recursive neural networks [41] are often mentioned as a use case of AD applied to recursive programs. While basic Child-Sum Tree-LSTMs can also be implemented with primitive recursion (a fold) over an inductively defined tree (which can be defined as a recursive type), there are other related models such as Top-Down-Tree-LSTMs that require an iterative or general recursive approach [47]. In fact, [20] has shown that a recursive approach is preferable as it better exposes the available parallelism in the model. In Appendix D, we show some Haskell code for the recursive neural network of [39], to give an idea of how iteration and recursive types (in the form of inductive types of labelled trees) naturally arise in a functional implementation of such neural net architectures. We imagine that many more applications of AD applied to recursive programs with naturally emerge as the technique made available to machine learning researchers and engineers. Finally, we speculate that coinductive types like streams of real numbers, which can be encoded using recursive types as $\mu\alpha.\mathbf{1} \to (\mathbf{real} * \alpha)$, provide a useful API for on-line machine learning applications [36], where data is processed in real time as it becomes available. Recursion and more notably recursive types introduce one final challenge into the correctness proof of AD of such expressive functional programs: the required logical relations arguments are notoriously technical, limiting the audience of any work using them and frustrating application to more complicated AD algorithms like CHAD. To mend this problem, we introduce a novel, simple but powerful logical relations technique for open semantic logical relations for recursive types.

1. Key ideas

In this paper, we consider how to perform forward and reverse mode numbers automatic differentiation on a functional language with expressive partial features, by using a dual numbers technique.

Language We consider an idealised functional language with product types $\tau \times \sigma$, sum types $\tau \sqcup \sigma$, function types $\tau \to \sigma$ generated by

- a primitive *type* **real** *of real numbers* (in practice, implemented as floating point numbers);
- *constants* $\vdash \underline{c} : \mathbf{real}$ for $c \in \mathbb{R}$;

- sets $(\mathrm{Op}_n)_{n\in\mathbb{N}}$ of n-ary *primitive operations* op, for which we include computations
  $x_1 : \mathbf{real}, \ldots, x_n : \mathbf{real} \vdash \mathrm{op}(x_1, \ldots, x_n) : \mathbf{real}$; we think of these as implementing partial functions $\mathbb{R}^n \rightharpoonup \mathbb{R}$ with open domain of definition, on which they are differentiable; for example, we can include mathematical operations $\log, \exp \in \mathrm{Op}_1$ and $(+), (*), (/) \in \mathrm{Op}_2$;
- a construct $x : \mathbf{real} \vdash \mathbf{sign}\,(x) : \mathbf{1} \sqcup \mathbf{1}$ that computes the *sign of a real number* and is undefined at $\underline{0}$; we can use it to define a lazy conditional on real numbers $\mathbf{if}\, r\, \mathbf{then}\, t\, \mathbf{else}\, s \overset{\mathrm{def}}{=} \mathbf{case}\, \mathbf{sign}\, r\, \mathbf{of}\, \{\_ \to t \mid \_ \to r\}$ of the kind that is often used in AD libraries like Stan [7].

Next, we include two more standard mechanisms for defining partial functions:

- *(purely functional) iteration*: given a computation $\Gamma, x : \tau \vdash t : \tau \sqcup \sigma$ to iterate and a starting value $\Gamma \vdash s : \tau$, we have a computation $\mathbf{iterate}\, t\, \mathbf{from}\, x = s : \sigma$ which repeatedly calls t, starting from the value of s until the result lies in σ;
- *recursion*: given a computation $\Gamma, x : \tau \to \sigma \vdash t : \tau \to \sigma$, we have a program $\Gamma \vdash \mu x.t : \tau \to \sigma$ that recursively computes to

$$\mathbf{let}\, x = \mu x.t\, \mathbf{in}\, t.$$

Dual numbers forward AD code transform Let us assume that we have programs $\partial_i \mathrm{op}(x_1, \ldots, x_n)$ that compute the i-th partial derivative of each n-ary primitive operation op. For example, we can define $\partial_1(*)(x_1, x_2) = x_2$ and $\partial_2(*)(x_1, x_2) = x_1$. Then, we can define a very straightforward forward mode AD code transformation $\mathcal{D}$ by replacing all primitive types **real** by a pair $\mathcal{D}(\mathbf{real}) \overset{\mathrm{def}}{=} \mathbf{real} \times \mathbf{real}$ of reals and by replacing all constants $\underline{c}$, n-ary primitive operations op and sign function **sign** in the program as[a]

$$\begin{aligned}
\mathcal{D}(\underline{c}) &\overset{\mathrm{def}}{=} \langle \underline{c}, \underline{0} \rangle\\
\mathcal{D}(\mathrm{op}(r_1, \ldots, r_n)) &\overset{\mathrm{def}}{=} \mathbf{case}\, \mathcal{D}(r_1)\, \mathbf{of}\, \langle x_1, x_1' \rangle \to \ldots \to \mathbf{case}\, \mathcal{D}(r_n)\, \mathbf{of}\, \langle x_n, x_n' \rangle \to\\
&\qquad \langle \mathrm{op}(x_1, \ldots, x_n), x_1' * \partial_1 \mathrm{op}(x_1, \ldots, x_n) + \ldots + x_n' * \partial_n \mathrm{op}(x_1, \ldots, x_n) \rangle\\
\mathcal{D}(\mathbf{sign}\, r) &\overset{\mathrm{def}}{=} \mathbf{sign}\,(\mathbf{fst}\, \mathcal{D}(r)).
\end{aligned}$$

[a]Actually, while our definition for $\mathcal{D}(\mathbf{sign}\, r)$ given here is correct, there exist more efficient implementation techniques, as we discuss in Appx. B.

We extend $\mathcal{D}$ to all other types and programs in the unique homomorphic (structure preserving way), by using structural recursion. So, for example, $\mathcal{D}(\tau \to \sigma) \overset{\text{def}}{=} \mathcal{D}(\tau) \to \mathcal{D}(\sigma)$, $\mathcal{D}(x) \overset{\text{def}}{=} x$, $\mathcal{D}(\mathbf{let}\, x = t\, \mathbf{in}\, s) = \mathbf{let}\, x = \mathcal{D}(t)\, \mathbf{in}\, \mathcal{D}(s)$ and $\mathcal{D}(t\, s) = \mathcal{D}(t)\, \mathcal{D}(s)$. We like to think of $\mathcal{D}$ as a structure preserving functor $\mathcal{D} : \mathbf{Syn} \to \mathbf{Syn}$ on the syntax.

Semantics To formulate correctness of the AD transformation $\mathcal{D}$, we need to assign a formal denotational semantics $[\![-]\!]$ to our language. We use the standard interpretation of types τ as ω-cpos $[\![\tau]\!]$ (partially ordered sets with suprema of countable chains) and programs $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash t : \sigma$ as monotone ω-continuous partial functions $[\![t]\!] : [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \rightharpoonup [\![\sigma]\!]$. We interpret **real** as the flat ω-cpo $[\![\mathbf{real}]\!] \overset{\text{def}}{=} \mathbb{R}$ of real numbers, $\underline{c}$ as the constant $[\![\underline{c}]\!] \overset{\text{def}}{=} c \in \mathbb{R}$, op as the partial differentiable function $[\![\mathrm{op}(x_1, \ldots, x_n)]\!] : \mathbb{R}^n \rightharpoonup \mathbb{R}$ that it is intended to implement and **sign** as the partial function $[\![\mathbf{sign}\,(x)]\!] : \mathbb{R} \rightharpoonup \mathbf{1} \sqcup \mathbf{1}$ that sends $r < 0$ to the left copy of $\mathbf{1}$, $r > 0$ to the right copy and is undefined for $r = 0$. Having fixed these definitions, the rest of the semantics is entirely compositional and standard. In particular, we interpret iteration and recursion using Kleene's Fixpoint Theorem. We think of this semantics as a structure preserving functor $[\![-]\!] : \mathbf{Syn} \to \boldsymbol{\omega}\mathbf{Cpo}$ from the syntax to the category of ω-cpos and monotone ω-continuous functions.

Correctness statement Having defined a semantics, we can phrase what it means for $\mathcal{D}$ to be correct. We prove the following, showing that $\mathcal{D}(t)$ implements the usual calculus derivative $D[\![t]\!]$ of $[\![t]\!]$.

Theorem 1.1 (Forward AD Correctness, Theorem 7.1 with $k = 1$ in main text). *For any program $x : \tau \vdash t : \sigma$ for $\tau = \mathbf{real}^k, \sigma = \mathbf{real}^l$ (where we write $\mathbf{real}^n$ for the type $\mathbf{real} \times \cdots \times \mathbf{real}$ of length n tuples of reals), we have that*

$$\begin{aligned}&[\![\mathcal{D}(t)]\!]((x_1, v_1), \ldots, (x_k, v_k)) = \\ &\Big(\pi_1([\![t]\!](x_1, \ldots, x_k)), \pi_l(D[\![t]\!]((x_1, \ldots, x_k), (v_1, \ldots, v_k))), \ldots, \\ &\quad \pi_l([\![t]\!](x_1, \ldots, x_k)), \pi_l(D[\![t]\!]((x_1, \ldots, x_k), (v_1, \ldots, v_k)))\Big)\end{aligned}$$

for any $(x_1,\ldots,x_k)$ *in the domain of definition of* $[\![t]\!]$ *and any tangent vector* $(v_1,\ldots,v_k)$ *to* $[\![\tau]\!]$ *at* x.

In fact, we also establish the theorem above for general types τ and σ not containing function types, but its phrasing requires slight bookkeeping that might distract from the simplicity of the theorem. Importantly, the program t might use higher-order functions, iteration, recursion, etc..

A proof via logical relations The proof of the correctness theorem follows a logical relations argument that we found using categorical methods, but which can be phrased entirely in elementary terms. Let us fix some $n \in \mathbb{N}$. We define for all types τ of our language, by induction, relations $T^n_\tau \subseteq (\mathbb{R}^n \to [\![\tau]\!]) \times ((\mathbb{R}^n \times \mathbb{R}^n) \to [\![\mathcal{D}(\tau)]\!])$ and $P^n_\tau \subseteq (\mathbb{R}^n \rightharpoonup [\![\tau]\!]) \times ((\mathbb{R}^n \times \mathbb{R}^n) \rightharpoonup [\![\mathcal{D}(\tau)]\!])$ that relate a (partial) n-curve to its derivative n-curve:

$$T^n_{\mathbf{real}} \overset{\text{def}}{=} \{(\gamma,\gamma') \mid \gamma \text{ is differentiable and } \gamma' = (x,v) \mapsto (\gamma(x), D\gamma(x,v))\}$$

$$T^n_{\tau\times\sigma} \overset{\text{def}}{=} \{(x \mapsto (\gamma_1(x),\gamma_2(x)), (x,v) \mapsto (\gamma_1'(x,v),\gamma_2'(x,v))) \mid (\gamma_1,\gamma_1') \in T^n_\tau \text{ and } (\gamma_2,\gamma_2') \in T^n_\sigma\}$$

$$T^n_{\tau\sqcup\sigma} \overset{\text{def}}{=} \{(\iota_1\circ\gamma_1, \iota_1\circ\gamma_1') \mid (\gamma_1,\gamma_1') \in T^n_\tau\} \cup \{(\iota_2\circ\gamma_2, \iota_2\circ\gamma_2') \mid (\gamma_2,\gamma_2') \in T^n_\sigma\}$$

$$T^n_{\tau\to\sigma} \overset{\text{def}}{=} \{(\gamma,\gamma') \mid \forall(\delta,\delta') \in T^n_\tau.(x \mapsto \gamma(x)(\delta(x)), (x,v) \mapsto \gamma'(x,v)(\delta'(x,v))) \in P^n_\sigma\}$$

$$P^n_\tau \overset{\text{def}}{=} \Big\{(\gamma,\gamma') \mid \gamma^{-1}([\tau]) \times \mathbb{R}^n = \gamma'^{-1}([\mathcal{D}(\tau)]) \text{ is open and for all differentiable}$$
$$\delta : \mathbb{R}^n \to \gamma^{-1}([\tau]) \text{ we have } (\gamma\circ\delta, (x,v) \mapsto (\gamma(\delta(x)), \gamma'(D\delta(x,v)))) \in T^n_\tau\Big\}.$$

We then prove the following "fundamental lemma", using induction on the typing derivation of t:

If $x_1 : \tau_1,\ldots,x_n : \tau_n \vdash t : \sigma$ and, for $1 \le i \le n$, $(f_i, f_i') \in T^n_{\tau_i}$, then
$(x \mapsto [\![t]\!](f_1(x),\ldots,f_n(x)), (x,v) \mapsto [\![\mathcal{D}(t)]\!](f_1'(x,v),\ldots,f_n'(x,v))) \in P^n_\sigma$.

For example, we use that, by assumption, $[\![\partial_i \mathrm{op}(x_1,\ldots,x_n)]\!]$ equals the i-th partial derivative of $[\![\mathrm{op}(x_1,\ldots,x_n)]\!]$ combined with the chain-rule, to show that primitive operations op respect the logical relations.

As $T^k_{\mathbf{real}^k}$ contains, in particular, (1.1) our theorem follows.

$$(\mathrm{id}, ((x_1,\ldots,x_k),(v_1,\ldots,v_k)) \mapsto ((x_1,v_1),\ldots,(x_n,v_k))) \tag{1.1}$$

Extending to recursive types via a novel categorical logical relations technique Next, we extend our language with ML-style polymorphism and recursive types. That is, we allow the formation of types τ with free type variables

α and we include a type variable binder $\mu\alpha.\tau$, which binds α in τ. We extend our AD transformation homomorphically on terms and types. For example, on types, we define

$$\mathcal{D}(\alpha) \stackrel{\text{def}}{=} \alpha \qquad\qquad \mathcal{D}(\mu\alpha.\tau) \stackrel{\text{def}}{=} \mu\alpha.\mathcal{D}(\tau).$$

A type τ with n free type variables gets interpreted in our ω-cpo-semantics as an n-ary mixed-variance endofunctor $[\![\tau]\!]$ on the category of ω-cpos and partial morphisms that restricts to that of ω-cpos and total morphisms. Programs with types that have free variables get interpreted as (extra)natural transformations. As the category of ω-cpos and partial morphisms has the structure to interpret recursive types of that of ω-cpos and total morphisms, we have a canonical *minimal invariant*

$$\text{roll} : [\![\tau]\!](\mu[\![\tau]\!], \mu[\![\tau]\!]) \xrightarrow{\cong} \mu[\![\tau]\!]$$

for the mixed-variance endofunctors $[\![\tau]\!]$ on $\boldsymbol{\omega}\mathbf{Cpo}$ that types τ denote [23]. We interpret $[\![\mu\alpha.\tau]\!] \stackrel{\text{def}}{=} \mu[\![\tau]\!]$.

To extend the correctness proof to this larger language, we would like to define the logical relation

$$T^n_{\mu\alpha.\tau} \stackrel{\text{def}}{=} \left\{(\text{roll} \circ \gamma, \text{roll} \circ \gamma') \mid (\gamma, \gamma') \in T^n_{\tau[{}^{\mu\alpha.\tau}/_{\alpha}]}\right\}.$$

That is, we would like to be able to *define relations using type recursion.* If we can do so, then extending the proof of the fundamental lemma is straightforward. We can then establish the correctness theorem also for τ and σ that involve recursive types.

The traditional method is to follow the technical recipes of [33]. Instead, we develop a powerful new logical relations technique for recursive types, which we believe to be more conceptually clear and easier to use in situations like ours. To be precise, we prove a general result saying that under mild conditions, that we can interpret recursive types in the category of logical relations over a category that models recursive types itself. For simplicity, we state an important special case that we need for our application here.

Given any right adjoint $\boldsymbol{\omega}\mathbf{Cpo}$-enriched functor $G : \boldsymbol{\omega}\mathbf{Cpo}^n \to \boldsymbol{\omega}\mathbf{Cpo}$, consider the category $\mathbf{SScone}$ of logical relations, which has objects (X, P), where $X \in \boldsymbol{\omega}\mathbf{Cpo}^n$ and P is a chain-closed subset of GX, and morphisms $(X, P) \to (X', P')$ are $\boldsymbol{\omega}\mathbf{Cpo}^n$-morphisms $f : X \to X'$ such that $y \in P$ implies $Gf(y) \in P'$.

Theorem 1.2 (Logical relations for recursive types, special case of theorem 8.13 in main text)**.** *Let T be a strong monad on* **SScone** *that lifts the usual partiality monad $(-)_\perp$ on $\boldsymbol{\omega}\mathbf{Cpo}^n$ along the projection functor* $\mathbf{SScone} \to \boldsymbol{\omega}\mathbf{Cpo}^n$*. We assume that T takes the initial object to the terminal one, and the square in $\boldsymbol{\omega}\mathbf{Cpo}$ induced by each component of the unit of T is a pullback. Then,* $\mathbf{SScone} \hookrightarrow \mathbf{SScone}_T$ *is a model for recursive types.*

In particular, we can define the relations $T_{\mu\alpha.\tau}$ using type recursion, as desired.

Dual numbers reverse AD Similarly to dual numbers forward AD $\mathcal{D}$, we can define a reverse AD code transformation $\overleftarrow{\mathcal{D}}$: we define

$$\overleftarrow{\mathcal{D}}(\mathbf{real}) \overset{\text{def}}{=} \mathbf{real} \times \mathbf{vect}$$

and

$$\begin{aligned}
\overleftarrow{\mathcal{D}}(\underline{c}) &\overset{\text{def}}{=} \langle \underline{c}, 0^{\mathbf{v}} \rangle \\
\overleftarrow{\mathcal{D}}(\mathrm{op}(t_1, \ldots, t_n)) &\overset{\text{def}}{=} \mathbf{case}\, \overleftarrow{\mathcal{D}}(t_1)\, \mathbf{of}\, \langle x_1, x_1' \rangle \to \ldots \mathbf{case}\, \overleftarrow{\mathcal{D}}(t_n)\, \mathbf{of}\, \langle x_n, x_n' \rangle \to \\
&\qquad \langle \mathrm{op}(x_1, \ldots, x_n), x_1' *^{\mathbf{v}} \partial_1 \mathrm{op}(x_1, \ldots, x_n) +^{\mathbf{v}} \ldots +^{\mathbf{v}} x_n' *^{\mathbf{v}} \partial_n \mathrm{op}(x_1, \ldots, x_n) \rangle \\
\overleftarrow{\mathcal{D}}(\mathbf{sign}\, t) &\overset{\text{def}}{=} \mathbf{sign}\, (\mathbf{fst}\, \overleftarrow{\mathcal{D}}(t)).
\end{aligned}$$

and extend homomorphically to all other type and term formers, as we did before. In fact, this algorithm is exactly the same as dual numbers forward AD in code with the only differences being that

(1) the type **real** of real numbers for tangents has been replaced with a new type **vect**, which we think of as representing (dynamically sized) cotangent vectors to the global input of the program;
(2) the zero $\underline{0}$ and addition $(+)$ of type **real** have been replaced by the zero $0^{\mathbf{v}}$ and addition $(+^{\mathbf{v}})$ of cotangents of type **vect**;
(3) the multiplication $(*) : \mathbf{real} \times \mathbf{real} \to \mathbf{real}$ has been replaced by the operation $(*^{\mathbf{v}}) : \mathbf{vect} \times \mathbf{real} \to \mathbf{vect}$: $(v *^{\mathbf{v}} r)$ is the rescaling of a cotangent v by the scalar r.

We write $\overline{e}_i$ for program representing the i-th canonical basis vector e_i of type **vect** and we write

$$\mathrm{Wrap}_s(x) \overset{\text{def}}{=} \mathbf{case}\, x\, \mathbf{of}\, \langle x_1, \ldots, x_s \rangle \to \langle \langle x_1, \overline{e}_1 \rangle, \ldots, \langle x_s, \overline{e}_s \rangle \rangle. \qquad (1.2)$$

We define $[\![\mathbf{vect}]\!] \overset{\text{def}}{=} \mathbb{R}^\infty \overset{\text{def}}{=} \sum_{k=0}^{\infty} \mathbb{R}^k$ as the infinite (vector space) coproduct of k-dimensional real vector spaces. That is, we interpret **vect** as the type of dynamically sized real vectors[b]. We show that $\overleftarrow{\mathcal{D}}(t)$ implements the transposed derivative $D[\![t]\!]^t$ of $[\![t]\!]$ in the following sense.

Theorem 1.3 (Reverse AD Correctness, Theorem 7.1 with $k = \infty$ in main text)**.** *For any program $x : \tau \vdash t : \sigma$ for $\tau = \mathbf{real}^s, \sigma = \mathbf{real}^l$,*

$$[\![\mathbf{let}\, x = \mathrm{Wrap}_k(x)\, \mathbf{in}\, \overleftarrow{\mathcal{D}}(t)]\!](x_1, \dots, x_s) =$$
$$\Big((\pi_1([\![t]\!](x_1, \dots, x_s)), D[\![t]\!]^t((x_1, \dots, x_s), e_1)), \dots, (\pi_l([\![t]\!](x_1, \dots, x_s)), D[\![t]\!]^t((x_1, \dots, x_s), e_l))\Big)$$

for any $(x_1, \dots, x_s)$ in the domain of definition of $[\![t]\!]$.

We prove this theorem again using a similar logical relations argument, defining $T^n_\tau \subseteq (\mathbb{R}^n \to [\![\tau]\!]) \times ((\mathbb{R}^n \times (\mathbb{R}^\infty)^n) \to [\![\overleftarrow{\mathcal{D}}(\tau)]\!])$ and $P^n_\tau \subseteq (\mathbb{R}^n \rightharpoonup [\![\tau]\!]) \times (\mathbb{R}^n \times (\mathbb{R}^\infty)^n) \rightharpoonup [\![\overleftarrow{\mathcal{D}}(\tau)]\!])$ as before for all types τ of language, setting

$$T^n_{\mathbf{real}} \stackrel{\text{def}}{=} \{(\gamma, \gamma') \mid \gamma \text{ is differentiable and } \gamma' = (x, L) \mapsto (\gamma(x), L(D\gamma^t(x, e_1)))\}$$
$$T^n_{\tau \times \sigma} \stackrel{\text{def}}{=} \{(x \mapsto (\gamma_1(x), \gamma_2(x)), (x, L) \mapsto (\gamma'_1(x, L), \gamma'_2(x, L))) \mid (\gamma_1, \gamma'_1) \in T^n_\tau \text{ and } (\gamma_2, \gamma'_2) \in T^n_\sigma\}$$
$$T^n_{\tau \sqcup \sigma} \stackrel{\text{def}}{=} \{(\iota_1 \circ \gamma_1, \iota_1 \circ \gamma'_1) \mid (\gamma_1, \gamma'_1) \in T^n_\tau\} \cup \{(\iota_2 \circ \gamma_2, \iota_2 \circ \gamma'_2) \mid (\gamma_2, \gamma'_2) \in T^n_\sigma\}$$
$$T^n_{\tau \to \sigma} \stackrel{\text{def}}{=} \{(\gamma, \gamma') \mid \forall (\delta, \delta') \in T^n_\tau . (x \mapsto \gamma(x)(\delta(x)), (x, L) \mapsto \gamma'(x, L)(\delta'(x, L))) \in P^n_\sigma\}$$
$$T^n_{\mu\alpha.\tau} \stackrel{\text{def}}{=} \Big\{(\mathrm{roll} \circ \gamma, \mathrm{roll} \circ \gamma') \mid (\gamma, \gamma') \in T^n_{\tau[{}^{\mu\alpha.\tau}/_{\alpha}]}\Big\}$$
$$P^n_\tau \stackrel{\text{def}}{=} \Big\{(\gamma, \gamma') \mid \gamma^{-1}([\![\tau]\!]) \times (\mathbb{R}^\infty)^n = \gamma'^{-1}([\![\overleftarrow{\mathcal{D}}(\tau)]\!]) \text{ is open and for all differentiable}$$
$$\delta : \mathbb{R}^n \to \gamma^{-1}([\![\tau]\!]) \text{ we have } (\gamma \circ \delta, (x, L) \mapsto \gamma'(\delta(x), L \circ D\delta^t(x, -))) \in T^n_\tau\Big\},$$

where we consider $(\mathbb{R}^\infty)^n$ as a type of linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^\infty$. We then prove the following "fundamental lemma", using induction on the typing derivation of t:

> If $x_1 : \tau_1, \dots, x_n : \tau_n \vdash t : \sigma$ and, for $1 \le i \le n$, $(f_i, f'_i) \in T^n_{\tau_i}$, then
> $(x \mapsto [\![t]\!](f_1(x), \dots, f_n(x)), (x, L) \mapsto [\![\overleftarrow{\mathcal{D}}(t)]\!](f'_1(x, L), \dots, f'_n(x, L))) \in P^n_\sigma$.

As $T^s_{\mathbf{real}^s}$ contains, in particular,

$$(\mathrm{id}, ((x_1, \dots, x_s), (L_1, \dots, L_s)) \mapsto ((x_1, L_1 e_1), \dots, (x_s, L_s e_s))),$$

our theorem follows.

[b]Note that, in practice, [37] actually implements **vect** as a type of ASTs of simple expressions computing a dynamically sized vector. This allows us to first build up the expression during execution of the program (the forward pass) and to only evaluate this cotangent expression later (in a reverse pass) making clever use of a distributivity law of addition and multiplication (also known as the linear factoring rule in [6]) to achieve the correct computational complexity of reverse AD.

Extending to arrays AD tends to be applied to programs that manipulate large arrays of reals. Seeing that such arrays are denotationally equivalent to lists $\mu\alpha.\mathbf{1} \sqcup \alpha \times \mathbf{real}$, while only the computational complexity of operations differs, our correctness result also applies to functional languages with arrays. We thus differentiate array types $\tau[]$ with elements of type τ in the obvious structure preserving way, e.g.

$$\mathcal{D}(\tau[]) \overset{\text{def}}{=} \mathcal{D}(\tau)[] \qquad \mathcal{D}(\mathbf{generate}) \overset{\text{def}}{=} \mathbf{generate} \tag{1.3}$$

$$\mathcal{D}(\mathbf{map}) \overset{\text{def}}{=} \mathbf{map} \qquad \mathcal{D}(\mathbf{foldr}) \overset{\text{def}}{=} \mathbf{foldr} \tag{1.4}$$

and similarly for dual numbers reverse AD.

2. Categorical models for CBV languages

The aim of this section is to establish a class of models for call-by-value (CBV) languages, and, then, add free recursion and iteration. We assume some familiarity with basic category theory (see, for instance, [11]). Whenever we talk about *strict preservation of some structure* (like products, coproducts or exponentials), we are assuming that we have chosen structures (chosen products, coproducts or exponentials) and the preservation is on the nose, that is to say, the canonical comparison is the identity.

Given a cartesian closed category $\mathcal{V}$, we can see it as a $\mathcal{V}$-enriched category w.r.t. the cartesian structure. Recall that a strong *monad* $\mathcal{T}$ on a cartesian closed category $\mathcal{V}$ is the same as a $\mathcal{V}$-monad on $\mathcal{V}$. More precisely, it is a triple

$$\mathcal{T} = \left(T : \mathcal{V} \to \mathcal{V}, \mathrm{m} : T^2 \to T, \eta : \mathrm{id}_{\mathcal{V}} \to T\right), \tag{2.1}$$

where T is a $\mathcal{V}$-endofunctor and m, η are $\mathcal{V}$-natural transformations, satisfying the usual associativity and identity equations, that is to say, $\mathrm{m} \cdot (\mathrm{m}T) = \mathrm{m} \cdot (T\mathrm{m})$ and $\mathrm{m} \cdot (\eta T) = \mathrm{id}_T = \mathrm{m} \cdot (T\eta)$.[c]

Let $\mathcal{T} = (T, \mathrm{m}, \eta)$ and $\mathcal{T}' = (T', \mathrm{m}', \eta')$ be monads on $\mathcal{V}$ and $\mathcal{V}'$ respectively. Recall that an oplax morphism (or a monad op-functor) between $\mathcal{T}$ and $\mathcal{T}'$ is a pair

$$(H : \mathcal{V} \to \mathcal{V}', \phi : HT \to T'H), \tag{2.2}$$

[c]See [11, pag. 60] for the classical enriched case. For the general case of monads in 2-categories, see [40, pag. 150] or, for instance, [27, Section 3].

where H is a functor and ϕ is a natural transformation, such that

$$\phi \cdot (H\eta) = (\eta' H) \qquad \text{and} \qquad (\mathrm{m}' H) \cdot (T'\phi) \cdot (\phi T) = \phi \cdot (H\mathrm{m}) . \qquad (2.3)$$

By the universal property of Kleisli categories, denoting by $J : \mathcal{V} \to \mathcal{C}$ and $J : \mathcal{V}' \to \mathcal{C}'$ the universal Kleisli functors, the oplax morphims (2.2) correspond bijectively with pairs of functors $\left(H : \mathcal{V} \to \mathcal{V}', \overline{H} : \mathcal{C} \to \mathcal{C}'\right)$ such that the diagram (2.5) commutes.

Definition 2.1 (*CBV* pair)**.** A *CBV* pair is a pair $(\mathcal{V}, \mathcal{T})$ where $\mathcal{V}$ is bicartesian closed category and $\mathcal{T}$ is a $\mathcal{V}$-monad on $\mathcal{V}$. We further require that $\mathcal{V}$ has chosen finite products, coproducts and exponentials.

A *CBV pair morphism* between the *CBV* pairs $(\mathcal{V}, \mathcal{T})$ and $(\mathcal{V}', \mathcal{T}')$ is a strictly bicartesian closed functor H such that (H, id) defines a monad op-functor (2.2). This defines a category of *CBV* pairs and *CBV* pair morphisms, denoted herein by $\mathfrak{C}_\mathrm{p}$.

Remark 2.2. If $(\mathcal{V}, \mathcal{T})$ is a *CBV* pair, since $\mathcal{T}$ is $\mathcal{V}$-enriched, we get a $\mathcal{V}$-enriched Kleisli category $\mathcal{C}$. We denote by

$$\mathcal{C}\left[-,-\right] = \left(- \Rightarrow^k -\right) : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{V} \qquad (2.4)$$

the $\mathcal{V}$-enriched hom functor. It should be noted that, if we denote by $(X \Rightarrow Y) = \mathcal{V}\left[X, Y\right]$ the exponential in $\mathcal{V}$, we have that

$$\mathcal{C}\left[X, Y\right] = \left(X \Rightarrow^k Y\right) = (X \Rightarrow TY)$$

which is the so called *Kleisli exponential* and corresponds to the function types for our language.

Denoting by $\mathcal{C}$ and $\mathcal{C}'$ the respective Kleisli categories, each morphism

$$(H, \phi) : (\mathcal{V}, \mathcal{T}) \to (\mathcal{V}', \mathcal{T}')$$

of *CBV* pairs gives rise to a commutative square

$$\begin{tikzcd} \mathcal{C} \arrow[r, "\overline{H}"] & \mathcal{C}' \\ \mathcal{V} \arrow[u, "J"] \arrow[r, "H"'] & \mathcal{V}' \arrow[u, "J'"'] \end{tikzcd} \qquad (2.5)$$

where J and J' are, respectively, the universal Kleisli functors of $\mathcal{T}$ and $\mathcal{T}'$. In this case, $\overline{H}$ strictly preserves Kleisli exponentials, finite coproducts and the action of $\mathcal{V}$ on $\mathcal{C}$. That is to say, $\left(H, \overline{H}\right)$ strictly preserves the distributive closed Freyd-categorical structure[d].

2.1. ***CBV*** **models: term recursion and iteration.** In order to interpret our language defined in Section 4, we need an additional support for term recursion and iteration. Since we do not impose further equations for the iteration or recursion constructs in our language, the following definitions establish our class of models for term recursion and iteration.

Definition 2.3 (Free Recursion and Iteration)**.** Let $(\mathcal{V}, \mathcal{T})$ be a *CBV* pair and $\mathcal{C}$ the corresponding $\mathcal{V}$-enriched Kleisli category.

- A *free recursion* for $(\mathcal{V}, \mathcal{T})$ is a family of morphisms
$$\mu = \left(\mu^{W,Y} : \mathcal{V}\left[\mathcal{C}\left[W,Y\right], \mathcal{C}\left[W,Y\right]\right] \longrightarrow \mathcal{C}\left[W,Y\right]\right)_{(W,Y)\in\mathcal{C}\times\mathcal{C}} \tag{2.6}$$
in $\mathcal{V}$.
- A *free iteration* for $(\mathcal{V}, \mathcal{T})$ is a family of morphisms
$$\mathsf{itt} = \left(\mathsf{itt}^{W,Y} : \mathcal{C}\left[W, W \sqcup Y\right] \longrightarrow \mathcal{C}\left[W,Y\right]\right)_{(W,Y)\in\mathcal{C}\times\mathcal{C}} \tag{2.7}$$
in $\mathcal{V}$.

Definition 2.4 (*CBV* model)**.** A *CBV model* is a quadruple $(\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$ in which $(\mathcal{V}, \mathcal{T})$ is a *CBV* pair, μ is a free recursion, and itt is a free iteration for $(\mathcal{V}, \mathcal{T})$.

A *CBV model morphism between* the *CBV* models
$$(\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt}) \text{ and } (\mathcal{V}', \mathcal{T}', \mu', \mathsf{itt}')$$
is a morphism H between the underlying *CBV* pairs such that
$$H\left(\mu^{W,Y}\right) = \mu'^{HW,HY} \text{ and } H\left(\mathsf{itt}^{W,Y}\right) = \mathsf{itt}'^{HW,HY},$$
for any $(W,Y) \in \mathcal{V} \times \mathcal{V}$. This defines a category of *CBV* models, denoted herein by $\mathfrak{C}_{\mathcal{BV}}$.

It should be noted that $\mathfrak{C}_{\mathcal{BV}}$ has finite products. Given *CBV* models $(\mathcal{V}_0, \mathcal{T}_0, \mu_0, \mathsf{itt}_0)$ and $(\mathcal{V}_1, \mathcal{T}_1, \mu_1, \mathsf{itt}_1)$, the product is given by
$$\left(\mathcal{V}_0 \times \mathcal{V}_1, \mathcal{T}_0 \times \mathcal{T}_1, (\mu_0, \mu_1), (\mathsf{itt}_0, \mathsf{itt}_1)\right) \tag{2.8}$$

[d]Although this level of generality is not needed in our work, the interested reader can find more about Freyd-categorical structures and basic aspects of the modelling of call-by-value languages in [24]

where (2.9) and (2.10) hold.

$$(\mu_0, \mu_1)^{(W,W'),(Y,Y')} = \left(\mu_0^{W,Y}, \mu_1^{W',Y'}\right) \tag{2.9}$$

$$(\mathsf{itt}_0, \mathsf{itt}_1)^{(W,W'),(Y,Y')} = \left(\mathsf{itt}_0^{W,Y}, \mathsf{itt}_1^{W',Y'}\right) \tag{2.10}$$

3. Concrete models

The aim of this section is to establish a class of concrete *CBV* pairs and models. We denote by $\boldsymbol{\omega}\mathbf{Cpo}$ the usual category of ω-cpos. The objects of $\boldsymbol{\omega}\mathbf{Cpo}$ are the partially ordered sets with colimits of ω-chains, while the morphisms are functors preserving these colimits. An ω-cpo is called *pointed* if it has a least element, denoted herein by $\perp$. We say that $f \in \boldsymbol{\omega}\mathbf{Cpo}(W, Y)$ is a pointed $\boldsymbol{\omega}\mathbf{Cpo}$-morphism if W is pointed and f preserves the least element.

It is well known that $\boldsymbol{\omega}\mathbf{Cpo}$ is bicartesian closed. We consider $\boldsymbol{\omega}\mathbf{Cpo}$-enriched categories w.r.t. the cartesian structure. Henceforth, if $\mathcal{V}$ is an $\boldsymbol{\omega}\mathbf{Cpo}$-enriched category and W, Y are objects of $\mathcal{V}$, we denote by $\mathcal{V}(W, Y)$ the $\boldsymbol{\omega}\mathbf{Cpo}$-enriched hom, that is to say, the ω-cpo of morphisms between W and Y.

An $\boldsymbol{\omega}\mathbf{Cpo}$-category $\mathcal{V}$ is $\boldsymbol{\omega}\mathbf{Cpo}$-*cartesian closed* if $\mathcal{V}$ has finite $\boldsymbol{\omega}\mathbf{Cpo}$-products and, moreover, for each object $Z \in \mathcal{V}$, the $\boldsymbol{\omega}\mathbf{Cpo}$-functor $(Z \times -) : \mathcal{V} \to \mathcal{V}$ has a right $\boldsymbol{\omega}\mathbf{Cpo}$-adjoint $\mathcal{V}[Z, -]$, called, herein, the $\boldsymbol{\omega}\mathbf{Cpo}$-exponential of Z. An $\boldsymbol{\omega}\mathbf{Cpo}$-functor $H : \mathcal{V} \to \mathcal{V}'$ is *strictly* $\boldsymbol{\omega}\mathbf{Cpo}$-*cartesian closed* if it strictly preserves the $\boldsymbol{\omega}\mathbf{Cpo}$-products and the induced comparison $H \circ \mathcal{V}[-,-] \to \mathcal{V}'[H(-), H(-)]$ is the identity.

Let $\mathcal{V}$ be $\boldsymbol{\omega}\mathbf{Cpo}$-cartesian closed. For any $Z \in \mathcal{V}$, since the hom-functor $\mathcal{V}(Z, -) : \mathcal{V} \to \boldsymbol{\omega}\mathbf{Cpo}$ is cartesian, it induces the *change of enriching base 2-functors*

$$\mathfrak{G}_{\mathcal{V}(Z,-)} : \mathcal{V}\text{-}\mathsf{Cat} \to \boldsymbol{\omega}\mathbf{Cpo}\text{-}\mathsf{Cat} \tag{3.1}$$

between the 2-categories of enriched categories w.r.t. the cartesian structures. Therefore, taking $Z = \mathbb{1}$ (the terminal object of $\mathcal{V}$), we get that every $\mathcal{V}$-category ($\mathcal{V}$-functor/$\mathcal{V}$-monad) has a *suitable underlying* $\boldsymbol{\omega}\mathbf{CPO}$-category ($\boldsymbol{\omega}\mathbf{Cpo}$-functor/$\boldsymbol{\omega}\mathbf{Cpo}$-monad), given by its image by $\mathfrak{G}_{\boldsymbol{\omega}\mathbf{Cpo}} := \mathfrak{G}_{\mathcal{V}(\mathbb{1},-)}$.

Definition 3.1 (*CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair)**.** A *CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-*pair* is a *CBV* pair $(\mathcal{V}, \mathcal{T})$ in which $\mathcal{V}$ is an $\boldsymbol{\omega}\mathbf{Cpo}$-bicartesian closed category, such that $\mathcal{V}(W, TY)$ is a pointed ω-cpo for any $(W, Y) \in \mathcal{V} \times \mathcal{V}$.

A *CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-*pair morphism* between $(\mathcal{V}, \mathcal{T})$ and $(\mathcal{V}', \mathcal{T}')$ is an $\boldsymbol{\omega}\mathbf{Cpo}$-functor $H : \mathcal{V} \to \mathcal{V}'$ whose underlying functor yields a morphism between the *CBV* pairs, and such that $H : \mathcal{V}(W, TY) \to \mathcal{V}(HW, HTY)$ is a pointed $\boldsymbol{\omega}$**Cpo**-morphism for any $(W, Y) \in \mathsf{ob}\,\mathcal{V} \times \mathsf{ob}\,\mathcal{V}$. This defines a category of *CBV* $\boldsymbol{\omega}$**Cpo**-pairs, denoted herein by $\boldsymbol{\omega}\mathbf{CPO}\text{-}\mathfrak{C}_{\mathcal{BV}}$.

There is, then, an obvious forgetful functor $\mathcal{U}_{\mathrm{p}} : \boldsymbol{\omega}\mathbf{CPO}\text{-}\mathfrak{C}_{\mathcal{BV}} \to \mathfrak{C}_{\mathrm{p}}$.

3.1. Fixpoints: term recursion and iteration. Recall that, if A is pointed and $q : A \to A$ is an endomorphism in $\boldsymbol{\omega}$**Cpo**, then q has a *least fixed point* given by the colimit of the chain

$$\bot \rightarrowtail q(\bot) \rightarrowtail \cdots \rightarrowtail q^n(\bot) \rightarrowtail \cdots \tag{3.2}$$

by Kleene's Fixpoint Theorem. Given such an endomorphism, we denote by $\mathrm{lfp}(q)$ its least fixed point.

Henceforth, let $(\mathcal{V}, \mathcal{T})$ be a *CBV* $\boldsymbol{\omega}$**Cpo**-pair, and $J : \mathcal{V} \to \mathcal{C}$ the corresponding $\mathcal{V}$-enriched universal Kleisli functor. We denote by $-\otimes- : \mathcal{V}\times\mathcal{C} \to \mathcal{C}$ the $\mathcal{V}$-tensor product in $\mathcal{C}$, also called the $\mathcal{V}$-copower, which, in this case, correspond to the usual action of $\mathcal{V}$ on $\mathcal{C}$.

By hypothesis, for any $W, Y, Z \in \mathcal{V}$, the ω-cpo $\mathcal{V}(Z, \mathcal{C}[W, Y]) \cong \mathcal{C}(Z \otimes W, Y)$ is pointed. Therefore we can define

$$\begin{aligned}
\overline{\mu}_Z^{W,Y} :\ & \mathcal{V}(Z \times \mathcal{C}[W, Y], \mathcal{C}[W, Y]) && \to \mathcal{V}(Z, \mathcal{C}[W, Y]) && (3.3)\\
& f && \mapsto \mathrm{lfp}(h \mapsto f \circ (Z \times h) \circ \partial_Z) &&\\
\overline{\mathsf{it}}_Z^{W,Y} :\ & \mathcal{C}(Z \otimes W, W \sqcup Y) && \to \mathcal{C}(Z \otimes W, Y) && (3.4)\\
& f && \mapsto \mathrm{lfp}(h \mapsto \langle h, J(\pi_Y)\rangle \circ (Z \otimes f) \circ (\mathrm{diag}_Z \otimes \mathrm{id}_W)) &&
\end{aligned}$$

where $\partial_Z = (\mathrm{id}_Z, \mathrm{id}_Z) : Z \to Z \times Z$ is the diagonal morphism, and $\mathrm{diag}_Z = J(\mathrm{id}_Z, \mathrm{id}_Z) : Z \to Z \otimes Z$. Since the morphisms above are $\boldsymbol{\omega}$**Cpo**-natural in $Z \in \mathcal{V}$, they give rise to the families of morphisms

$$\mu_\omega = \left(\mu_\omega^{W,Y}\right)_{(W,Y)\in\mathcal{C}\times\mathcal{C}} \overset{\mathrm{def}}{=} \left(\overline{\mu}^{W,Y}_{\mathcal{V}[\mathcal{C}[W,Y],\mathcal{C}[W,Y]]}\left(\mathsf{eval}_{\mathcal{C}[W,Y],\mathcal{C}[W,Y]}\right)\right)_{(W,Y)\in\mathcal{C}\times\mathcal{C}} \tag{3.5}$$

$$\mathsf{it}_\omega = \left(\mathsf{it}_\omega^{W,Y}\right)_{(W,Y)\in\mathcal{C}\times\mathcal{C}} \overset{\mathrm{def}}{=} \left(\overline{\mathsf{it}}^{W,Y}_{\mathcal{V}[W,T(W\sqcup Y)]}\left(J\left(\mathsf{eval}_{W,T(W\sqcup Y)}\right)\right)\right)_{(W,Y)\in\mathcal{C}\times\mathcal{C}} \tag{3.6}$$

by the Yoneda Lemma, where $\mathsf{eval}_{A,B} : \mathcal{V}[A, B] \times A \to B$ is the evaluation morphism given by the cartesian closed structure.

Lemma 3.2 (Underlying *CBV* model)**.** *There is a forgetful functor* $\mathcal{U}_{\mathcal{BV}} : \boldsymbol{\omega}\mathbf{CPO}\text{-}\mathfrak{C}_{\mathcal{BV}} \to \mathfrak{C}_{\mathcal{BV}}$ *defined by* $\mathcal{U}_{\mathcal{BV}}(\mathcal{V}, \mathcal{T}) = (\mathcal{V}, \mathcal{T}, \mu_\omega, \mathsf{it}_\omega)$*, taking every morphism* H *to its underlying morphism of CBV models.*

Proof: Since H is a $\boldsymbol{\omega}$**Cpo**-functor and, for any $(W,Y) \in \mathsf{ob}\,\mathcal{V} \times \mathsf{ob}\,\mathcal{V}$,

$$H : \mathcal{V}(W,TY) \to \mathcal{V}'(HW,T'HY)$$

is a pointed $\boldsymbol{\omega}$**Cpo**-morphism, we get that, indeed, H respects the free iteration and free recursion as defined in (3.5) and (3.6). ■

It should be noted that, given *CBV* $\boldsymbol{\omega}$**Cpo**-pairs $(\mathcal{V}_0,\mathcal{T}_0)$ and $(\mathcal{V}_1,\mathcal{T}_1)$,

$$(\mathcal{V}_0,\mathcal{T}_0) \times (\mathcal{V}_1,\mathcal{T}_1) = (\mathcal{V}_0 \times \mathcal{V}_1, \mathcal{T}_0 \times \mathcal{T}_1) \tag{3.7}$$

is the product in $\boldsymbol{\omega}$**CPO**-$\mathfrak{C}_{\mathcal{BV}}$. Moreover, $\mathcal{U}_{\mathcal{BV}}$ preserves finite products.

4. Automatic Differentiation for term recursion and iteration

For our purpose, we could define our macro in terms of total derivatives. However, we choose to present it in terms of partial derivatives, in order to keep our treatment as close as possible to the starting point of the efficient implementation of the reverse mode given in [37].

Following this choice of presentation, it is particularly convenient to establish our *AD* macro $\mathcal{D}$ as a *program transformation* between a *source language* and a *target language* (see 4.4). The main point of this distinction is to keep track of the difference between the types corresponding to manifolds (cartesian spaces) and the (co)tangents (vector spaces) in the target language (see [37]).

4.1. Source language with iteration and recursion.

We consider a standard (coarse-grain) call-by-value language over a ground type **real**, certain real constants $\underline{c} \in \mathrm{Op}_0$, certain primitive operations $\mathrm{op} \in \mathrm{Op}_n$ for each nonzero natural number $n \in \mathbb{N}^*$, and **sign**. We denote $\mathrm{Op} := \bigcup_{n\in\mathbb{N}} \mathrm{Op}_n$.

As it is clear from the semantics defined in 5.3, **real** intends to implement the real numbers. Moreover, for each $n \in \mathbb{N}$, the operations in Op_n intend to implement partially defined functions $\mathbb{R}^n \rightharpoonup \mathbb{R}$. Finally, **sign** intends to implement the partially defined function $\mathbb{R} \rightharpoonup \mathbb{R}$ defined in $\mathbb{R}^- \cup \mathbb{R}^+$ which takes $\mathbb{R}^-$ to -1 and $\mathbb{R}^+$ to 1.

Although it is straightforward to consider more general settings, we also add the assumption that the primitive operations implement differentiable functions (see 5.6).

We treat this operations in a schematic way as this reflects the reality of practical Automatic Differentiation libraries, which are constantly being expanded with new primitive operations.

The types τ, σ, ρ, values v, w, u, and computations t, s, r of our language are as follows.

τ, σ, ρ	$::=$	types	$\mid\ \mathbf{1} \mid \tau_1 \times \tau_2$	products
	$\mid\ \mathbf{real}$	numbers	$\mid\ \tau \to \sigma$	function
	$\mid\ \mathbf{0} \mid \tau \sqcup \sigma$	sums		
v, w, u	$::=$	values	$\mid\ \langle\,\rangle \mid \langle v, w\rangle$	tuples
	$\mid\ x, y, z$	variables	$\mid\ \lambda x.t$	abstractions
	$\mid\ \underline{c}$	constants	$\mid\ \mu x.t$	term recursion
	$\mid\ \mathbf{inl}\, v \mid \mathbf{inr}\, v$	sum inclusions		
t, s, r	$::=$	computations	$\mid\ \langle\,\rangle \mid \langle t, s\rangle$	tuples
	$\mid\ x, y, z$	variables	$\mid\ \mathbf{case}\, s\, \mathbf{of}\, \langle x, y\rangle \to t$	product match
	$\mid\ \mathbf{let}\, t =\ x\, \mathbf{in}\, s$	sequencing	$\mid\ \lambda x.t$	abstractions
	$\mid\ \underline{c}$	constant	$\mid\ t\ s$	function app.
	$\mid\ \mathrm{op}(t_1, \ldots, t_n)$	operation	$\mid\ \mathbf{iterate}\, t\, \mathbf{from}\, x = s$	iteration
	$\mid\ \mathbf{case}\, t\, \mathbf{of}\, \{\,\}$	sum match	$\mid\ \mu x.t$	term recursion
	$\mid\ \mathbf{inl}\, t \mid \mathbf{inr}\, t$	sum inclusions	$\mid\ \mathbf{sign}\, t$	sign function
	$\mid\ \mathbf{case}\, r\, \mathbf{of}\, \{ \begin{smallmatrix} \mathbf{inl}\, x \to t \\ \mid \mathbf{inr}\, y \to s \end{smallmatrix} \}$	sum match		

We use sugar $\mathbf{if}\, r\, \mathbf{then}\, t\, \mathbf{else}\, s \overset{\text{def}}{=} \mathbf{case}\, \mathbf{sign}\, r\, \mathbf{of}\, \{\_ \to t \mid \_ \to r\}$, $\mathbf{fst}\, t \overset{\text{def}}{=} \mathbf{case}\, t\, \mathbf{of}\, \langle x, \_\rangle \to x$, $\mathbf{snd}\, t \overset{\text{def}}{=} \mathbf{case}\, t\, \mathbf{of}\, \langle \_, x\rangle \to x$ and $\mathbf{let}\, \mathbf{rec}\, f(x) = t\, \mathbf{in}\, s \overset{\text{def}}{=} \mathbf{let}\, f = \mu f.\lambda x.t\, \mathbf{in}\, s$. In fact, we can consider iteration as syntactic sugar as well:
$\mathbf{iterate}\, t\, \mathbf{from}\, x = s \overset{\text{def}}{=} (\mu z.\lambda x.\mathbf{case}\, t\, \mathbf{of}\, \{\mathbf{inl}\, x' \to z\, x' \mid \mathbf{inr}\, x'' \to x''\})\, s$.

The computations are typed according to the rules of Fig. 4.1 and Fig. 4.2, *where* $\mathtt{R} \subseteq \mathbb{R}$ *is a fixed set of real numbers containing* 0. For now, the reader may ignore the kinding contexts Δ. They will serve to support our treatment of ML-style polymorphism later.

We consider the standard CBV $\beta\eta$-equational theory of [30] for our language, which we list in Fig. 4.3. We could impose further equations for the iteration construct as is done in [5, 16] as well as for the basic operations op and the sign function **sign** . However, such equations are unnecessary for our development.

4.2. Target language. We define our *target language* by extending the source language adding the following syntax, with the typing rules of Fig. 4.4.

$$
\frac{((x:\tau)\in\Gamma)}{\Delta\mid\Gamma\vdash x:\tau}
\quad
\frac{\Delta\mid\Gamma\vdash t:\sigma\quad \Delta\mid\Gamma,x:\sigma\vdash s:\tau}{\Delta\mid\Gamma\vdash \mathbf{let}\,x=\ t\,\mathbf{in}\,s:\tau}
\quad
\frac{(c\in\mathtt{R})}{\Delta\mid\Gamma\vdash \underline{c}:\mathbf{real}}
$$

$$
\frac{\{\Delta\mid\Gamma\vdash t_i:\mathbf{real}\}_{i=1}^{n}\quad(\mathrm{op}\in\mathrm{Op}_n)}{\Delta\mid\Gamma\vdash \mathrm{op}(t_1,\ldots,t_n):\mathbf{real}}
\quad
\frac{\Delta\mid\Gamma\vdash t:\mathbf{0}}{\Delta\mid\Gamma\vdash \mathbf{case}\,t\,\mathbf{of}\,\{\,\}:\tau}
\quad
\frac{\Delta\mid\Gamma\vdash t:\tau}{\Delta\mid\Gamma\vdash \mathbf{inl}\,t:\tau\sqcup\sigma}
$$

$$
\frac{\Delta\mid\Gamma\vdash t:\tau}{\Delta\mid\Gamma\vdash \mathbf{inr}\,t:\tau\sqcup\sigma}
\quad
\frac{\Delta\mid\Gamma\vdash r:\sigma\sqcup\rho\quad \Delta\mid\Gamma,x:\sigma\vdash t:\tau\quad \Delta\mid\Gamma,y:\rho\vdash s:\tau}{\Delta\mid\Gamma\vdash \mathbf{case}\,r\,\mathbf{of}\,\{\mathbf{inl}\,x\to t\ \mid \mathbf{inr}\,y\to s\}:\tau}
\quad
\frac{}{\Delta\mid\Gamma\vdash\langle\,\rangle:\mathbf{1}}
$$

$$
\frac{\Delta\mid\Gamma\vdash t:\tau\quad \Delta\mid\Gamma\vdash s:\sigma}{\Delta\mid\Gamma\vdash\langle t,s\rangle:\tau\times\sigma}
\quad
\frac{\Delta\mid\Gamma\vdash r:\sigma\times\rho\quad \Delta\mid\Gamma,x:\sigma,y:\rho\vdash t:\tau}{\Delta\mid\Gamma\vdash \mathbf{case}\,r\,\mathbf{of}\,\langle x,y\rangle\to t:\tau}
\quad
\frac{\Delta\mid\Gamma,x:\sigma\vdash t:\tau}{\Delta\mid\Gamma\vdash\lambda x.t:\sigma\to\tau}
$$

$$
\frac{\Delta\mid\Gamma\vdash t:\sigma\to\tau\quad \Delta\mid\Gamma\vdash s:\sigma}{\Delta\mid\Gamma\vdash t\,s:\tau}
\quad
\frac{\Delta\mid\Gamma\vdash r:\mathbf{real}}{\Delta\mid\Gamma\vdash \mathbf{sign}\,r:\mathbf{1}\sqcup\mathbf{1}}
$$

FIGURE 4.1. Typing rules for a basic source language with real conditionals, where $\mathtt{R}\subseteq\mathbb{R}$ is a fixed set of real numbers containing 0.

$$
\frac{\Delta\mid\Gamma,x:\sigma\vdash t:\sigma\sqcup\tau\quad \Delta\mid\Gamma\vdash r:\sigma}{\Delta\mid\Gamma\vdash \mathbf{iterate}\,t\,\mathbf{from}\,x=r:\tau}
\quad
\frac{\Delta\mid\Gamma,x:\tau\vdash t:\tau}{\Delta\mid\Gamma\vdash \mu x.t:\tau}(\tau=\sigma\to\rho)
$$

FIGURE 4.2. Typing rules for term recursion and iteration.

$$
\begin{array}{ll}
\mathbf{let}\,x=\ v\,\mathbf{in}\,t=t[{}^{v}\!/\!{}_{x}] & \mathbf{let}\,y=\ (\mathbf{let}\,x=\ t\,\mathbf{in}\,s)\,\mathbf{in}\,r=\mathbf{let}\,x=\ t\,\mathbf{in}\,(\mathbf{let}\,y=\ s\,\mathbf{in}\,r)\\
\mathbf{case}\,\mathbf{inl}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x\to t\ \mid\mathbf{inr}\,y\to s\}=t[{}^{v}\!/\!{}_{x}] & t[{}^{v}\!/\!{}_{z}]\overset{\#x,y}{=}\mathbf{case}\,v\,\mathbf{of}\,\{\begin{array}{c}\mathbf{inl}\,x\to t[{}^{\mathbf{inl}\,x}\!/\!{}_{z}]\\ \mid\mathbf{inr}\,y\to t[{}^{\mathbf{inr}\,y}\!/\!{}_{z}]\end{array}\}\\
\mathbf{case}\,\mathbf{inr}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x\to t\ \mid\mathbf{inr}\,y\to s\}=s[{}^{v}\!/\!{}_{y}] & \\
\mathbf{case}\,\langle v,w\rangle\,\mathbf{of}\,\langle x,y\rangle\to t=t[{}^{v}\!/\!{}_{x},{}^{w}\!/\!{}_{y}] & t[{}^{v}\!/\!{}_{z}]\overset{\#x,y}{=}\mathbf{case}\,v\,\mathbf{of}\,\langle x,y\rangle\to t[{}^{\langle x,y\rangle}\!/\!{}_{z}]\\
(\lambda x.t)\ v=t[{}^{v}\!/\!{}_{x}] & v\overset{\#x}{=}\lambda x.v\,x
\end{array}
$$

FIGURE 4.3. 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.

τ,σ,ρ	$::=$	types	$\mid$	$\mathbf{vect}$	(co)tangent
	$\mid\ \ldots$	as before			
v,w,u	$::=$	values	$\mid$	$\overline{0}$	zero
	$\mid\ \overline{e}_i$	i-th canonical element	$\mid$	$t+s$	addition of vectors
	$\mid\ \ldots$	as before	$\mid$	$t*s$	scalar multiplication
			$\mid$	$\mathfrak{h}_i t$	proj. handler
			$\mid$	$\overline{0}$	zero
t,s,r	$::=$	computations	$\mid$	$t+s$	addition of vectors
	$\mid\ \ldots$	as before	$\mid$	$t*s$	scalar multiplication
	$\mid\ \overline{e}_i$	canonical element	$\mid$	$\mathfrak{h}_i t$	proj. handler

$$\frac{(i \in \mathbb{N}^*)}{\Delta \mid \Gamma \vdash \overline{e}_i : \mathbf{vect}} \qquad \frac{}{\Delta \mid \Gamma \vdash \overline{0} : \mathbf{vect}} \qquad \frac{\Delta \mid \Gamma \vdash t : \mathbf{vect} \quad \Delta \mid \Gamma \vdash s : \mathbf{vect}}{\Delta \mid \Gamma \vdash t + s : \mathbf{vect}}$$

$$\frac{\Delta \mid \Gamma \vdash t : \mathbf{vect} \quad \Delta \mid \Gamma \vdash s : \mathbf{real}}{\Delta \mid \Gamma \vdash t * s : \mathbf{vect}} \qquad \frac{(i \in \mathbb{N}^*) \quad \Delta \mid \Gamma \vdash t : \mathbf{vect}}{\Delta \mid \Gamma \vdash \mathfrak{h}_i t : \mathbf{real}^i}$$

FIGURE 4.4. Extra typing rules for the target language with iteration and recursion, where we denote $\mathbb{N}^* := \mathbb{N} - \{0\}$, $\mathbf{real}^1 := \mathbf{real}$ and $\mathbf{real}^{i+1} = \mathbf{real}^i \times \mathbf{real}$.

The operational semantics of the target language depends on the intended behavior for the *AD* macro $\mathcal{D}$ defined in 4.4. In our context, we want **vect** to implement a vector space (playing the role of the (co)tangent), with the respective operations and the usual laws between the operations such as distributivity of the scalar multiplication over the vector addition (which is particularly useful for efficient implementations [37]).

The terms $\mathfrak{h}_i t$ are irrelevant for the definition and correctness of the macro $\mathcal{D}$, but it is particularly useful to illustrate the expected types in 7.6 and 7.7. Although this perspective is negligible to our correctness statement, **vect** can be seen as a type encompassing a *linear effect* with *handlers* given by the terms $\mathfrak{h}_i t$.

We are particularly interested in the case that $(\mathbf{vect}, +, *, \overline{0})$ implements the vector space $(\mathbb{R}^k, +, *, 0)$, for some $k \in \mathbb{N} \cup \{\infty\}$,[e] where $\overline{e}_i$ implements the i-th element $e_i^k \in \mathbb{R}^k$ of the canonical basis if $k = \infty$ or if $i \leq k$, and $0 \in \mathbb{R}^k$ otherwise. In this case, $\mathfrak{h}_i t$ is supposed to implement

$$\mathfrak{p}_{k \to i} : \mathbb{R}^k \to \mathbb{R}^i, \tag{4.5}$$

which denotes the canonical projection if $i \leq k$ and the coprojection otherwise.

For short, we say that **vect** implements the vector space $\mathbb{R}^k$ to refer to the case above. It corresponds to the k-semantics for the target language defined in 5.4.

4.3. The syntactic *CBV* models.

As discussed in Appendix A, we can translate our coarse-grain languages to fine-grain call-by-value languages.

[e] $\mathbb{R}^\infty$ is the vector space freely generated by the infinite set $\{e_i : i \in \mathbb{N}^*\}$. In other words, it is the infinity coproduct of $\mathbb{R}^i$ $(i \in \mathbb{N}^*)$. In order to implement it, one can use lists/arrays and pattern matching for the vector addition.

The fine-grain languages corresponding to the source and target languages correspond to the CBV models

$$\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right) \qquad \text{and} \qquad \left(\mathbf{Syn}^{\mathbf{tr}}_V, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_\mu, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right) \tag{4.6}$$

with the following universal properties.

Theorem 4.1 (Universal Property of CBV models (4.6))**.** *Let* $(\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$ *be a* CBV *model. Assume that Fig. 4.7 and Fig. 4.8 are given consistent assignments.*

(1) There is a unique CBV *model morphism*

$$H : \left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right) \to (\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$$

respecting the assignment of Fig. 4.7.

(2) There is a unique CBV *model morphism*

$$\mathcal{H} : \left(\mathbf{Syn}^{\mathbf{tr}}_V, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_\mu, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right) \to (\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$$

that extends H *and respects the assignment of Fig. 4.8.*

For each primitive operation $\mathrm{op} \in \mathrm{Op}_n$ $(n \in \mathbb{N})$ and each constant $c \in \mathbb{R}$:

$$H(\mathbf{real}) \in \mathsf{ob}\,\mathcal{V}; \quad H(\mathbf{sign}\,) \in \mathcal{C}\left(H(\mathbf{real}), 1 \sqcup 1\right) = \mathcal{V}\left(H(\mathbf{real}), T\left(1 \sqcup 1\right)\right);$$
$$H(\underline{c}) \in \mathcal{V}\left(1, H(\mathbf{real})\right); \quad H(\mathrm{op}) \in \mathcal{C}\left(H(\mathbf{real})^n, H(\mathbf{real})\right) = \mathcal{V}\left(H(\mathbf{real})^n, TH(\mathbf{real})\right).$$

FIGURE 4.7. Assignment that gives the universal property of the source language.

$$\mathcal{H}(\mathbf{vect}) \in \mathsf{ob}\,\mathcal{V}; \ \mathcal{H}(\overline{0}) \in \mathcal{V}\left(1, \mathcal{H}(\mathbf{vect})\right);$$
$$\mathcal{H}\left(\mathfrak{h}_i\right) \in \mathcal{V}\left(\mathcal{H}\left(\mathbf{vect}\right), \mathcal{H}\left(\mathbf{real}\right)^i\right) \ \text{(for each } i \in \mathbb{N}^*\text{)};$$
$$\mathcal{H}(+) \in \mathcal{V}\left(\mathcal{H}(\mathbf{vect})^2, T\mathcal{H}(\mathbf{vect})\right); \quad \mathcal{H}(*) \in \mathcal{V}\left(\mathcal{H}(\mathbf{vect}) \times \mathcal{H}\left(\mathbf{real}\right), T\mathcal{H}\left(\mathbf{vect}\right)\right).$$

FIGURE 4.8. Assignment that gives the universal property of the target language.

4.4. Dual numbers AD for term recursion and iteration. Let us fix, for all $n \in \mathbb{N}$, for all $\mathrm{op} \in \mathrm{Op}_n$, for all $1 \leq i \leq n$, computations $x_1 : \mathbf{real}, \ldots, x_n : \mathbf{real} \vdash \partial_i \mathrm{op}(x_1, \ldots, x_n) : \mathbf{real}$, which represent the partial derivatives of op. Using these terms for representing partial derivatives, we define, in Fig. 4.9, a

$$
\begin{array}{lll}
\mathcal{D}(\mathbf{real}) \overset{\text{def}}{=} \mathbf{real} \times \mathbf{vect} & \mathcal{D}(\mathbf{0}) \overset{\text{def}}{=} \mathbf{0} & \mathcal{D}(\tau \sqcup \sigma) \overset{\text{def}}{=} \mathcal{D}(\tau) \sqcup \mathcal{D}(\sigma) \\
\mathcal{D}(\mathbf{1}) \overset{\text{def}}{=} \mathbf{1} & \mathcal{D}(\tau \to \sigma) \overset{\text{def}}{=} \mathcal{D}(\tau) \to \mathcal{D}(\sigma) & \mathcal{D}(\tau \times \sigma) \overset{\text{def}}{=} \mathcal{D}(\tau) \times \mathcal{D}(\sigma)
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{D}(x) \overset{\text{def}}{=} x & \mathcal{D}(\mathbf{let}\, x = \, t \,\mathbf{in}\, s) \overset{\text{def}}{=} \mathbf{let}\, x = \, \mathcal{D}(t) \,\mathbf{in}\, \mathcal{D}(s) \\
\mathcal{D}(\mathbf{case}\, r \,\mathbf{of}\, \{\,\}) \overset{\text{def}}{=} \mathbf{case}\, \mathcal{D}(r) \,\mathbf{of}\, \{\,\} & \\
\mathcal{D}(\mathbf{inl}\, t) \overset{\text{def}}{=} \mathbf{inl}\, \mathcal{D}(t) & \mathcal{D}(\mathbf{case}\, r \,\mathbf{of}\, \{ \begin{array}{c} \mathbf{inl}\, x \to t \\ \mid \mathbf{inr}\, y \to s \end{array} \}) \overset{\text{def}}{=} \\
\mathcal{D}(\mathbf{inr}\, t) \overset{\text{def}}{=} \mathbf{inr}\, \mathcal{D}(t) & \quad \mathbf{case}\, \mathcal{D}(r) \,\mathbf{of}\, \{ \begin{array}{c} \mathbf{inl}\, x \to \mathcal{D}(t) \\ \mid \mathbf{inr}\, y \to \mathcal{D}(s) \end{array} \} \\
\mathcal{D}(\langle\,\rangle) \overset{\text{def}}{=} \langle\,\rangle & \\
\mathcal{D}(\langle t, s \rangle) \overset{\text{def}}{=} \langle \mathcal{D}(t), \mathcal{D}(s) \rangle & \mathcal{D}(\mathbf{case}\, r \,\mathbf{of}\, \langle x, y \rangle \to t) \overset{\text{def}}{=} \mathbf{case}\, \mathcal{D}(r) \,\mathbf{of}\, \langle x, y \rangle \to \mathcal{D}(t) \\
\mathcal{D}(\lambda x.t) \overset{\text{def}}{=} \lambda x.\mathcal{D}(t) & \mathcal{D}(t\, r) \overset{\text{def}}{=} \mathcal{D}(t)\, \mathcal{D}(r) \\
\mathcal{D}(\mathbf{iterate}\, t \,\mathbf{from}\, x = r) \overset{\text{def}}{=} & \mathcal{D}(\mu x.t) \overset{\text{def}}{=} \mu x.\mathcal{D}(t) \\
\quad \mathbf{iterate}\, \mathcal{D}(t) \,\mathbf{from}\, x = \mathcal{D}(r) &
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{D}(\underline{c}) \overset{\text{def}}{=} & \langle \underline{c}, \underline{0} \rangle \\
\mathcal{D}(\mathrm{op}(r_1, \ldots, r_n)) \overset{\text{def}}{=} & \mathbf{case}\, \mathcal{D}(r_1) \,\mathbf{of}\, \langle x_1, x_1' \rangle \to \ldots \to \mathbf{case}\, \mathcal{D}(r_n) \,\mathbf{of}\, \langle x_n, x_n' \rangle \to \\
& \mathbf{let}\, y = \, \mathrm{op}(x_1, \ldots, x_n) \,\mathbf{in} \\
& \mathbf{let}\, z_1 = \, \partial_1 \mathrm{op}(x_1, \ldots, x_n) \,\mathbf{in}\, \ldots \mathbf{let}\, z_n = \, \partial_n \mathrm{op}(x_1, \ldots, x_n) \,\mathbf{in} \\
& \langle y, x_1' * z_1 + \ldots + x_n' * z_n \rangle \\
\mathcal{D}(\mathbf{sign}\, r) \overset{\text{def}}{=} & \mathbf{sign}\, (\mathbf{fst}\, \mathcal{D}(r))
\end{array}
$$

FIGURE 4.9. AD macro $\mathcal{D}(-)$ defined on types and computations. All newly introduced variables are chosen to be fresh. We provide a more efficient way of differentiating **sign** in Appx. B.

structure preserving macro $\mathcal{D}$ on the types and computations of our language for performing AD.

We extend $\mathcal{D}$ to contexts: $\mathcal{D}(\{x_1{:}\tau_1, ..., x_n{:}\tau_n\}) \overset{\text{def}}{=} \{x_1{:}\mathcal{D}(\tau_1), ..., x_n{:}\mathcal{D}(\tau_n)\}$. This turns $\mathcal{D}$ into a well-typed, functorial macro in the following sense.

Lemma 4.2 (Functorial macro)**.** *Our macro respects typing, substitution, and $\beta\eta$-equality:*

- *If* $\Gamma \vdash t : \tau$, *then* $\mathcal{D}(\Gamma) \vdash \mathcal{D}(t) : \mathcal{D}(\tau)$.
- $\mathcal{D}(\mathbf{let}\, x = \, t \,\mathbf{in}\, s) = \mathbf{let}\, x = \, \mathcal{D}(t) \,\mathbf{in}\, \mathcal{D}(s)$.
- *If* $t \overset{\beta\eta}{=} s$, *then* $\mathcal{D}(t) \overset{\beta\eta}{=} \mathcal{D}(s)$.

Our macro $\mathcal{D}$ can be seen as a class of macros, since it depends on the target language. More precisely, it depends on what **vect** implements (see 4.2).

4.5. AD transformation as a *CBV* model morphism. By the universal property of $\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}\right)$ established in Theorem 4.1, the assignment defined in Fig. 4.11 induces a unique CBV model morphism

$$\mathbb{D} : \left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}\right) \to \left(\mathbf{Syn}^{\mathbf{tr}}_V, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_{\mu}, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right). \quad (4.10)$$

$$\mathbb{D}(\mathbf{real}) \overset{\text{def}}{=} \mathbf{real} \times \mathbf{vect} \in \mathsf{ob}\,\mathbf{Syn}^{\mathbf{tr}}_V, \qquad \mathbb{D}(\underline{c}) \overset{\text{def}}{=} (\underline{c}, \underline{0}) \in \mathbf{Syn}^{\mathbf{tr}}_V\left(1, \mathbf{real} \times \mathbf{vect}\right),$$

$$\mathbb{D}(\mathrm{op}) \overset{\text{def}}{=} \lambda y_1.\lambda \ldots .\lambda y_n.\, \overrightarrow{d}\,\mathrm{op}\,(y_1, \ldots, y_n) \in \mathbf{Syn}^{\mathbf{tr}}_V\left((\mathbf{real} \times \mathbf{vect})^n, \mathbf{Syn}_{\mathcal{S}}\left(\mathbf{real} \times \mathbf{vect}\right)\right),$$

$$\mathbb{D}(\mathbf{sign}\,) \overset{\text{def}}{=} (\mathbf{sign}\, \circ \pi_1) \in \mathbf{Syn}^{\mathbf{tr}}_V\left(\mathbf{real} \times \mathbf{vect}, \mathbf{Syn}_{\mathcal{S}}\left(1 \sqcup 1\right)\right),$$

for each primitive operation $\mathrm{op} \in \mathrm{Op}_n$ ($n \in \mathbb{N}$) and each constant $c \in \mathtt{R}$, where

$$\begin{aligned}\overrightarrow{d}\,\mathrm{op}\,(y_1, \ldots, y_n) \overset{\text{def}}{=}\quad & \mathbf{case}\, y_1\, \mathbf{of}\, \langle x_1, x_1' \rangle \to \ldots \to \mathbf{case}\, y_n\, \mathbf{of}\, \langle x_n, x_n' \rangle \to \\ & \mathbf{let}\, y' = \mathrm{op}(x_1, \ldots, x_n)\, \mathbf{in} \\ & \mathbf{let}\, z_1 = \partial_1\mathrm{op}(x_1, \ldots, x_n)\, \mathbf{in} \ldots \mathbf{let}\, z_n = \partial_n\mathrm{op}(x_1, \ldots, x_n)\, \mathbf{in} \\ & \langle y', x_1' * z_1 + \ldots + x_n' * z_n \rangle.\end{aligned}$$

FIGURE 4.11. AD assignment.

The macro $\mathcal{D}$ defined in Fig. 4.9 is encompassed by (4.10).

5. Semantics for the AD transformation

We establish basic facts about the semantics of the automatic differentiation.

5.1. Basic concrete model. The most fundamental example of a CBV $\boldsymbol{\omega}\mathbf{Cpo}$-pair is given by $(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp)$ where $(-)_\perp$ is the lax idempotent monad that freely adds an initial object $\perp$ to each ω-cpo. Indeed, of course, $\boldsymbol{\omega}\mathbf{Cpo}\,(W, (Y)_\perp)$ is pointed for any pair $(W, Y) \in \mathsf{ob}\,\boldsymbol{\omega}\mathbf{Cpo} \times \mathsf{ob}\,\boldsymbol{\omega}\mathbf{Cpo}$.

We consider the product

$$(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp) \times (\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp) = (\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp),$$

where, by abuse of language, $((C, C'))_\perp = ((C)_\perp, (C')_\perp)$. By Lemma 3.2, $\mathcal{U}_{\mathcal{BV}}\,(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp)$ and

$$\mathcal{U}_{\mathcal{BV}}\,(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp) = \mathcal{U}_{\mathcal{BV}}\,(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp) \times \mathcal{U}_{\mathcal{BV}}\,(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp)$$

are CBV models.

5.2. Differentiable functions and interleaved derivatives. *Henceforth, unless stated otherwise, the cartesian spaces $\mathbb{R}^n$ and its subspaces are endowed with the respective discrete $\boldsymbol{\omega}$**Cpo**-structures.*

Definition 5.1 (Interleaving function)**.** For each $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, denoting by $\mathbb{I}_n$ the set $\{1,\ldots,n\}$, we define the isomorphism (in $\boldsymbol{\omega}$**Cpo** with the respective discrete $\boldsymbol{\omega}$**Cpo**-structures)

$$\begin{array}{rcl} \phi_{n,k}: & \mathbb{R}^n \times \left(\mathbb{R}^k\right)^n & \to \left(\mathbb{R} \times \mathbb{R}^k\right)^n \\ & ((x_j)_{j\in\mathbb{I}_n}, (y_j)_{j\in\mathbb{I}_n}) & \mapsto (x_j, y_j)_{j\in\mathbb{I}_n}. \end{array} \tag{5.1}$$

For each open subset $U \subseteq \mathbb{R}^n$, we denote by

$$\phi^U_{n,k} : U \times \left(\mathbb{R}^k\right)^n \to \phi_{n,k}\left(U \times \left(\mathbb{R}^k\right)^n\right)$$

the isomorphism obtained from restricting $\phi_{n,k}$.

In Def. 5.2, Remark 5.3 and Lemma 5.4, let $g : U \to \coprod_{j\in L} V_j$ be a map where U is an open subset of $\mathbb{R}^n$, and, for each $i \in L$, V_i is an open subset of $\mathbb{R}^{m_i}$.

Definition 5.2 (Derivative)**.** The map g is *differentiable* if, for any $i \in L$, $g^{-1}(V_i) = W_i$ is open in $\mathbb{R}^n$ and the restriction $g|_{W_i} : W_i \to V_i$ is differentiable w.r.t the submanifold structures $W_i \subseteq \mathbb{R}^n$ and $V_i \subseteq \mathbb{R}^{m_i}$. In this case, for each $k \in (\mathbb{N} \cup \{\infty\})$, we define the function:

$$\begin{array}{rcl} \mathfrak{D}^k g: & \phi_{n,k}\left(U \times \left(\mathbb{R}^k\right)^n\right) & \to \coprod_{j\in L}\left(\phi_{m_j,k}\left(V_i \times \left(\mathbb{R}^k\right)^{m_i}\right)\right) \qquad (5.2) \\ & z & \mapsto \iota_{m_j} \circ \phi^{V_j}_{m_j,k}\left(g(x), \tilde{w}\cdot g'(x)^t\right), \qquad (5.3) \\ & & \text{whenever } \phi^{-1}_{n,k}(z) = (x,w) \in W_i \times \left(\mathbb{R}^k\right)^n \end{array}$$

in which $\tilde{w}$ is the linear transformation $\mathbb{R}^n \to \mathbb{R}^k$ corresponding to the vector w, $\cdot$ is the composition of linear transformations, ι_{m_i} is the obvious *ith*-coprojection of the coproduct (in the category $\boldsymbol{\omega}$**Cpo**), and $g'(x)^t$ is the transpose of the derivative $g'(x) : \mathbb{R}^n \to \mathbb{R}^{m_i}$ of $g|_{W_i} : W_i \to V_i$ at $x \in U$.

Remark 5.3. It should be noted that, in Def. 5.2, W_i might be empty for some $i \in L$. In this case, $g|_{W_i} : W_i \to V_i$ is trivially differentiable. Analogously, U might be empty. In this case, the function g *is differentiable and $\mathfrak{D}^k g$ is the unique morphism with domain $\emptyset$ and codomain as in* (5.2).

Lemma 5.4. *Let $\dot{g}$ be a function with domain as in (5.2). The map g is differentiable and $\dot{g} = \mathfrak{D}^k g$ if, and only if, $g \circ \alpha$ is differentiable and $\dot{g} \circ \mathfrak{D}^k \alpha = \mathfrak{D}^k (g \circ \alpha)$ for any differentiable map $\alpha : \mathbb{R}^n \to U$.*

Definition 5.5 (Differentiable partial maps)**.** Let $h : \coprod_{r \in K} \mathbb{R}^{n_r} \to \left(\coprod_{j \in L} \mathbb{R}^{m_j} \right)_{\perp}$ be a morphism in $\boldsymbol{\omega}\mathbf{Cpo}$. We say that h is differentiable if, for each $i \in K$, the component $h_i := h \circ \iota_i : \mathbb{R}^{n_i} \to \left(\coprod_{j \in L} \mathbb{R}^{m_j} \right)_{\perp}$ satisfies the following two conditions:

- $h_i^{-1} \left(\coprod_{j \in L} \mathbb{R}^{m_j} \right) = U_i$ is open in $\mathbb{R}^{n_i}$;
- the corresponding total function (5.4) is differentiable.

$$\underline{h_i} = h|_{U_i} : U_i \to \coprod_{j \in L} \mathbb{R}^{m_j} \qquad (5.4) \qquad \mathfrak{d}^k(h) : \coprod_{r \in K} \left(\mathbb{R} \times \mathbb{R}^k \right)^{n_r} \to \left(\coprod_{j \in L} \left(\mathbb{R} \times \mathbb{R}^k \right)^{m_j} \right)_{\perp} \qquad (5.5)$$

In this case, for each $k \in \mathbb{N} \cup \{\infty\}$, we define (5.5) to be the morphism induced by $\langle \mathfrak{d}^k(h_r) \rangle_{r \in K}$ where, for each $i \in K$, $\mathfrak{d}^k(h_i)$ is defined by (5.6), which is just the corresponding canonical extension of the map $\mathfrak{D}^k h_i$.

$$\begin{array}{rrl} \mathfrak{d}^k(h_i) : & \left(\mathbb{R} \times \mathbb{R}^k \right)^{n_i} & \to \left(\coprod_{j \in L} \left(\mathbb{R} \times \mathbb{R}^k \right)^{m_j} \right)_{\perp} \\ & z & \mapsto \begin{cases} \mathfrak{D}^k h_i(z), & \text{if } z \in \phi_{n_i,k} \left(U_i \times \left(\mathbb{R}^k \right)^{n_i} \right) \subseteq \left(\mathbb{R} \times \mathbb{R}^k \right)^{n_i}; \\ \perp, & \text{otherwise.} \end{cases} \end{array} \qquad (5.6)$$

5.3. The semantics for the source language. We give a concrete semantics for our language, interpreting it in the *CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair $(\boldsymbol{\omega}\mathbf{Cpo}, (-)_{\perp})$.

We denote by $\mathbb{R}$ the discrete ω-cpo of real numbers, and we define $\mathsf{sign} : \mathbb{R} \to (\mathbf{1} \sqcup \mathbf{1})_{\perp}$ by (5.8), where $\iota_1, \iota_2 : \mathbf{1} \to \mathbf{1} \sqcup \mathbf{1}$ are the two coprojections of the coproduct.

$$[\![-]\!] : \left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}} \right) \to \mathcal{U}_{\mathcal{BV}} \left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_{\perp} \right) \qquad (5.7)$$

$$\mathsf{sign}(x) = \begin{cases} \perp, & \text{if } x = 0 \\ \iota_1(*), & \text{if } x < 0 \\ \iota_2(*), & \text{if } x > 0 \end{cases} \qquad (5.8)$$

By the universal property of $\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right)$, there is only one *CBV* model morphism (5.7) consistent with the assignment of Fig. 5.9 where c is the constant that $\underline{c}$ intends to implement, and, for each $\mathrm{op} \in \mathrm{Op}_n$, f_{op} is the partial map that op intends to implement.

$$\boxed{\begin{array}{c} [\![\mathbf{real}]\!] \overset{\text{def}}{=} \mathbb{R} \in \mathsf{ob}\,\boldsymbol{\omega}\mathbf{Cpo}; \qquad [\![\underline{c}]\!] \overset{\text{def}}{=} \mathsf{c} \in \boldsymbol{\omega}\mathbf{Cpo}\left(1, \mathbb{R}\right); \\ [\![\mathrm{op}]\!] \overset{\text{def}}{=} f_{\mathrm{op}} \in \boldsymbol{\omega}\mathbf{Cpo}\left(\mathbb{R}^n, (\mathbb{R})_\perp\right); \quad [\![\mathbf{sign}\,]\!] \overset{\text{def}}{=} \mathsf{sign} \in \boldsymbol{\omega}\mathbf{Cpo}\left(\mathbb{R}, (1 \sqcup 1)_\perp\right). \end{array}}$$

FIGURE 5.9. Semantics' assignment for each primitive operation $\mathrm{op} \in \mathrm{Op}_n$ ($n \in \mathbb{N}$) and each constant $c \in \mathtt{R}$.

The *CBV* model morphism (5.7) (or, more precisely, the underlying functor of the *CBV* morphism $[\![-]\!]$) gives the semantics for the source language. Although our work holds for more general contexts, we consider the following assumption over the semantics of our language.

Assumption 5.6. For each $n \in \mathbb{N}$ and $\mathrm{op} \in \mathrm{Op}_n$, $[\![\mathrm{op}]\!] = f_{\mathrm{op}} : \mathbb{R}^n \to (\mathbb{R})_\perp$ is differentiable.

5.4. The k-semantics for the target language. For each $k \in \mathbb{N} \cup \{\infty\}$, we define the k-semantics for the target language by interpreting **vect** as the vector space $\mathbb{R}^k$. Namely, we extend the semantics $[\![-]\!]$ of the source language into a k-semantics of the target language. More precisely, by Theorem 4.1, there is a unique *CBV* model morphism (5.10) that extends $[\![-]\!]$ and is consistent with the assignment given by the vector structure (5.11) together with the projection (coprojection) $[\![\mathfrak{h}_i]\!]_k : \mathbb{R}^k \to \mathbb{R}^i$ if $i \leq k$ ($i \geq k$), for each $i \in \mathbb{N}^*$.

$$[\![-]\!]_k : \left(\mathbf{Syn}^{\mathbf{tr}}_V, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_\mu, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right) \;\to\; \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \quad (5.10)$$

$$\left([\![\mathbf{vect}]\!]_k, [\![+]\!]_k, [\![*]\!]_k, [\![\overline{0}]\!]_k\right) \;:=\; \left(\mathbb{R}^k, +, *, 0\right) \quad (5.11)$$

5.5. Prim-op-correct macro.

Definition 5.7 (Sound for primitives)**.** A macro $\mathcal{D}$ as defined in Fig. 4.9 and its corresponding *CBV* model morphism $\mathbb{D}$ as defined in (4.10) are *sound for primitives* if, for any primitive $\mathrm{op} \in \mathrm{Op}$, $[\![\mathcal{D}(\mathrm{op})]\!]_k = \mathfrak{d}^k\left([\![\mathrm{op}]\!]\right)$ for any k.

For each $j \in \mathbb{I}_n$, given a differentiable function $f : \mathbb{R}^n \to (\mathbb{R})_\perp$, we denote by $\mathfrak{d}_j\left(f\right) : \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R})_\perp$ the function defined by $\mathfrak{d}_j\left(f\right)(x_1, \ldots, x_n) =$

$\mathfrak{d}^1(f) \circ \phi_{n,1}\big((x_1,\dots,x_n), e^n_j\big)$, where e^n_j the j-th vector of the canonical basis of $\mathbb{R}^n$.

Lemma 5.8. *The macro $\mathcal{D}$ defined in Fig. 4.9 is sound for primitives provided that*

$$[\![\langle \mathrm{op}(y_1,\dots,y_n), \partial_j \mathrm{op}(y_1,\dots,y_n)\rangle]\!] = \mathfrak{d}_j\left([\![\mathrm{op}]\!]\right), \tag{5.12}$$

for any primitive operation $\mathrm{op} \in \mathrm{Op}_n$ *of the source language.*

6. Enriched scone and subscone

Given an $\boldsymbol{\omega}\mathbf{Cpo}$-functor $G : \mathcal{B} \to \mathcal{D}$, the comma $\boldsymbol{\omega}\mathbf{Cpo}$-category $\mathcal{D} \downarrow G$ of the identity along G in $\boldsymbol{\omega}\mathbf{Cpo}$-$\mathsf{Cat}$ is defined as follows.

- The objects of $\mathcal{D} \downarrow G$ are triples $(D \in \mathcal{D}, C \in \mathcal{B}, j : D \to G(C))$ in which j is a morphism of $\mathcal{D}$;
- a morphism $(D, C, j) \to (D', C', h)$ between objects of $\mathcal{D} \downarrow G$ is a pair (6.1) making (6.2) commutative in $\mathcal{D}$;

$$\alpha = (\alpha_0 : D \to D', \alpha_1 : C \to C') \tag{6.1}$$

$$\begin{tikzcd} D \arrow[r, "\alpha_0"] \arrow[d, "j"'] & D' \arrow[d, "h"] \\ G(C) \arrow[r, "G(\alpha_1)"'] & G(C') \end{tikzcd} \tag{6.2}$$

- if $\alpha = (\alpha_0 : D \to D', \alpha_1 : C \to C')\,, \beta = (\beta_0 : D \to D', \beta_1 : C \to C') : (D, C, j) \to (D', C', h)$, are two morphisms of $\mathcal{D} \downarrow G$, we have that $\alpha \leq \beta$ if $\alpha_0 \leq \beta_0$ in $\mathcal{D}$ and $\alpha_1 \leq \beta_1$ in $\mathcal{B}$.

Following the approach of [26, Section 9], we have:

Theorem 6.1. *Let $G : \mathcal{B} \to \mathcal{D}$ be a right $\boldsymbol{\omega}\mathbf{Cpo}$-adjoint functor. Assuming that $\mathcal{D}$ has finite $\boldsymbol{\omega}\mathbf{Cpo}$-products and $\mathcal{B}$ has finite $\boldsymbol{\omega}\mathbf{Cpo}$-coproducts, the $\boldsymbol{\omega}\mathbf{Cpo}$-functor*

$$\mathcal{L} : \mathcal{D} \downarrow G \to \mathcal{D} \times \mathcal{B}, \tag{6.3}$$

defined by $(D \in \mathcal{D}, C \in \mathcal{B}, j : D \to G(C)) \mapsto (D, C)$, is $\boldsymbol{\omega}\mathbf{Cpo}$-comonadic and $\boldsymbol{\omega}\mathbf{Cpo}$-monadic. This implies, in particular, that $\mathcal{L}$ creates (and strictly preserves) $\boldsymbol{\omega}\mathbf{Cpo}$-limits and colimits.

By Theorem 6.1 and the enriched adjoint triangle theorem[f], we have:

[f]See [10] for the original adjoint triangle theorem, and [25, Section 1] for the enriched version.

Corollary 6.2. *Let $G : \mathcal{B} \to \mathcal{D}$ be a right $\boldsymbol{\omega}$**Cpo**-adjoint functor between $\boldsymbol{\omega}$**Cpo**-bicartesian closed categories. In this case, $\mathcal{D} \downarrow G$ is an $\boldsymbol{\omega}$**Cpo**-bicartesian closed category. Moreover, if $\mathcal{D} \times \mathcal{B}$ is $\boldsymbol{\omega}$**Cpo**-cocomplete, so is $\mathcal{D} \downarrow G$.*

Theorem 6.1 and Corollary 6.2 are $\boldsymbol{\omega}$**Cpo**-enriched versions of the fundamental results of [26, Section 9]. The details and proofs are presented in Appx. C.

6.1. Subscone. *Henceforth, we assume that* **Sub** $(\mathcal{D} \downarrow G)$ *is a full reflective and replete $\boldsymbol{\omega}$**Cpo**-subcategory of $\mathcal{D} \downarrow G$. We denote, herein, by $\mathfrak{T}_{sub}$ the idempotent $\boldsymbol{\omega}$**Cpo**-monad induced by the $\boldsymbol{\omega}$**Cpo**-adjuntion.*

Recall that *a morphism q in $\boldsymbol{\omega}$**Cpo** is full* if its underlying functor is full. In this case, the underlying functor is also faithful and injective on objects. Moreover, a morphism j *in an $\boldsymbol{\omega}$**Cpo**-category $\mathcal{B}$ is full* if $\mathcal{B}(B, j)$ is full in $\boldsymbol{\omega}$**Cpo** for any $B \in \mathcal{B}$.

Furthermore, recall that *an $\boldsymbol{\omega}$**Cpo**-functor $H : \mathcal{W} \to \mathcal{Z}$ is locally full if, for any $(X, W) \in \mathsf{ob}\,\mathcal{W} \times \mathsf{ob}\,\mathcal{W}$, the morphism $H : \mathcal{W}(X, W) \to \mathcal{Z}(HX, HW)$ is a full $\boldsymbol{\omega}$**Cpo**-morphism.* It should be noted that the 2-functor underlying a locally full $\boldsymbol{\omega}$**Cpo**-functor is *locally fully faithful.* Moreover, since every full morphism in $\boldsymbol{\omega}$**Cpo** is injective on objects, *every locally full $\boldsymbol{\omega}$**Cpo**-functor is faithful (locally injective on objects).*

Assumption 6.3. We require that:

(Sub.1) whenever $(D \in \mathcal{D}, C \in \mathcal{B}, j) \in$ **Sub** $(\mathcal{D} \downarrow G)$, j is a full morphism in $\mathcal{B}$;
(Sub.2) $G : \mathcal{B} \to \mathcal{D}$ is a right $\boldsymbol{\omega}$**Cpo**-adjoint functor between $\boldsymbol{\omega}$**Cpo**-bicartesian closed categories;
(Sub.3) $\mathfrak{T}_{sub}$ strictly preserves $\boldsymbol{\omega}$**Cpo**-products;
(Sub.4) Diag. (6.5) commutes.

$$\begin{tikzcd}\mathbf{Sub}(\mathcal{D} \downarrow G) \arrow[r] & \mathcal{D} \downarrow G \arrow[r, "\mathcal{L}"] & \mathcal{D} \times \mathcal{B} \arrow[r, "\pi_{\mathcal{B}}"] & \mathcal{B}\end{tikzcd} \tag{6.4}$$

$$\begin{tikzcd}\mathcal{D} \downarrow G \arrow[rr, "\mathfrak{T}_{sub}"] \arrow[d, "\mathcal{L}"'] & & \mathcal{D} \downarrow G \arrow[d, "\mathcal{L}"] \\ \mathcal{D} \times \mathcal{B} \arrow[r, "\pi_{\mathcal{B}}"'] & \mathcal{B} & \mathcal{D} \times \mathcal{B} \arrow[l, "\pi_{\mathcal{B}}"]\end{tikzcd} \tag{6.5}$$

We denote by $\underline{\mathcal{L}}$: **Sub** $(\mathcal{D} \downarrow G) \to \mathcal{B}$ the $\boldsymbol{\omega}$**Cpo**-functor given by the composition (6.4) where the unlabeled arrow is the full inclusion.

Theorem 6.4. *The full inclusion* $\mathbf{Sub}\,(\mathcal{D}\downarrow G)\to\mathcal{D}\downarrow G$ *creates (and strictly preserves)* $\boldsymbol{\omega}\mathbf{Cpo}$*-limits and* $\boldsymbol{\omega}\mathbf{Cpo}$*-exponentials. Moreover, if* $\mathcal{D}\downarrow G$ *is* $\boldsymbol{\omega}\mathbf{Cpo}$*-cocomplete, so is* $\mathbf{Sub}\,(\mathcal{D}\downarrow G)$*.*

Proof: $\mathbf{Sub}\,(\mathcal{D}\downarrow G)\to\mathcal{D}\downarrow G$ is $\boldsymbol{\omega}\mathbf{Cpo}$-monadic and, hence, it creates $\boldsymbol{\omega}\mathbf{Cpo}$-limits.

By (Sub.3) of 6.3, $\mathfrak{T}_{sub}$ is commutative and, hence, $\mathbf{Sub}\,(\mathcal{D}\downarrow G)\to\mathcal{D}\downarrow G$ creates $\boldsymbol{\omega}\mathbf{Cpo}$-exponentials.

Since $\mathfrak{T}_{sub}$ is idempotent, $\mathbf{Sub}\,(\mathcal{D}\downarrow G)$ is $\boldsymbol{\omega}\mathbf{Cpo}$-cocomplete whenever $\mathcal{D}\downarrow G$ is. ■

Corollary 6.5. $\mathbf{Sub}\,(\mathcal{D}\downarrow G)$ *is an* $\boldsymbol{\omega}\mathbf{Cpo}$*-bicartesian closed category. Moreover, if* $\mathcal{D}\times\mathcal{B}$ *is* $\boldsymbol{\omega}\mathbf{Cpo}$*-cocomplete, so is* $\mathbf{Sub}\,(\mathcal{D}\downarrow G)$*.*

Proof: It follows from Theorem 6.4 and Corollary 6.2. ■

Theorem 6.6. *The* $\boldsymbol{\omega}\mathbf{Cpo}$*-functor* $\underline{\mathcal{L}}:\mathbf{Sub}\,(\mathcal{D}\downarrow G)\to\mathcal{B}$ *is strictly (bi)cartesian closed and locally full (hence, faithful). Moreover,* $\underline{\mathcal{L}}$ *strictly preserves* $\boldsymbol{\omega}\mathbf{Cpo}$*-colimits.*

Proof: The $\boldsymbol{\omega}\mathbf{Cpo}$-functors $\mathcal{L}:\mathcal{D}\downarrow G\to\mathcal{D}\times\mathcal{B}$ and $\pi_{\mathcal{B}}:\mathcal{D}\times\mathcal{B}\to\mathcal{B}$ strictly preserve $\boldsymbol{\omega}\mathbf{Cpo}$-weighted limits and colimits. Since $\mathfrak{T}_{sub}$ is idempotent and (6.5) commutes, this implies that $\underline{\mathcal{L}}$ strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-limits and colimits.

The composition $\pi_{\mathcal{B}}\circ\mathcal{L}$ has a left $\boldsymbol{\omega}\mathbf{Cpo}$-adjoint given by $C\mapsto(0,C,\iota_0)$. Since the counit of this $\boldsymbol{\omega}\mathbf{Cpo}$-adjunction is the identity and $\pi_{\mathcal{B}}\circ\mathcal{L}$ strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-products, we get that this $\boldsymbol{\omega}\mathbf{Cpo}$-adjunction strictly satisfies the *Frobenius reciprocity condition.* This implies that $\pi_{\mathcal{B}}\circ\mathcal{L}$ strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-exponentials.

Since $\mathfrak{T}_{sub}$ strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-products, we get that $\mathbf{Sub}\,(\mathcal{D}\downarrow G)\to\mathcal{D}\downarrow G$ strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-exponentials as well. Therefore, $\underline{\mathcal{L}}$ strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-exponentials.

The locally fully faithfulness (and, hence, faithfulness) of $\underline{\mathcal{L}}$ follows from (Sub.1) of 6.3. ■

Remark 6.7. Condition (Sub.1) of 6.3 ensures that our subscone indeed gives us a proof-irrelevant approach: in particular, as stressed above, it implies that $\underline{\mathcal{L}}$ is faithful. Given objects $(D,C,j),(D',C',j')$ and a morphism $f:C\to C'$ in $\mathcal{B}$, if there is $\alpha:(D,C,j)\to(D',C',j')$ satisfying $\underline{\mathcal{L}}(\alpha)=f$, then α is unique with this property. In this case, *we say that f defines a morphism $(D,C,j)\to(D',C',j')$ in* $\mathbf{Sub}\,(\mathcal{D}\downarrow G)$.

7. Correctness of Dual Numbers AD

In this section, we show that, as long as the macro $\mathcal{D}$ defined in Fig. 4.9 is sound for primitives and **vect** implements $\mathbb{R}^k$, $\mathcal{D}$ is correct according to the k-specification below. More precisely, we prove that:

Theorem 7.1. *Assume that* **vect** *implements the vector space* $\mathbb{R}^k$*, for some* $k \in \mathbb{N} \cup \{\infty\}$*. For any program* $x : \tau \vdash t : \sigma$ *where* τ, σ *are data types, we have that* $[\![t]\!]$ *is differentiable and, moreover,*

$$[\![\mathcal{D}(t)]\!]_k = \mathfrak{d}^k\left([\![t]\!]\right) \tag{7.1}$$

provided that $\mathcal{D}$ *is sound for primitives.*

In 7.7 and 7.6, we show how we can *correctly* get the derivative and the transpose derivative out of Theorem 7.1. In other words, we get forward and reverse AD out of our *correct* macro, provided that **vect** implements a suitable vector space $\mathbb{R}^k$.

7.1. Basic setting. *Henceforth, we follow the notation and definitions established in Section 5.* In particular, *unless stated otherwise, the cartesian spaces* $\mathbb{R}^n$ *and its subspaces are endowed with the discrete* $\boldsymbol{\omega}\mathbf{Cpo}$*-structure.*

For each $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, we define the $\boldsymbol{\omega}\mathbf{Cpo}$-functor (7.2). We consider the full reflective $\boldsymbol{\omega}\mathbf{Cpo}$-subcategory $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ of $\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}$ whose objects are triples (7.3) such that j is full (and, hence, injective on objects).

$$G_{n,k} \overset{\text{def}}{=} \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}\left(\left(\mathbb{R}^n, \left(\mathbb{R} \times \mathbb{R}^k\right)^n\right), (-,-)\right) : \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo} \to \boldsymbol{\omega}\mathbf{Cpo} \tag{7.2}$$

$$\left(D \in \boldsymbol{\omega}\mathbf{Cpo},\ (C, C') \in \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo},\ \left(j : D \to G_{n,k}(C, C')\right) \in \boldsymbol{\omega}\mathbf{Cpo}\right) \tag{7.3}$$

The $\boldsymbol{\omega}\mathbf{Cpo}$-functor $G_{n,k}$ together with $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ satisfies 6.3. Therefore:

Theorem 7.2. $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ *is a cocomplete* $\boldsymbol{\omega}\mathbf{Cpo}$*-cartesian closed category. Moreover, the forgetful* $\boldsymbol{\omega}\mathbf{Cpo}$*-functor* $\underline{\mathcal{L}}_{n,k} : \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right) \to \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}$ *is locally full and strictly cartesian closed. Furthermore, it strictly preserves* $\boldsymbol{\omega}\mathbf{Cpo}$*-colimits.*

Proof: It follows from Corollary 6.5 and Theorem 6.6. ■

7.2. The monad. Let $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. In order to get a categorical model of our language, we need to define a partiality monad for $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$.

We denote by $\mathfrak{O}_n$ the set of proper open non-empty subsets of the cartesian space $\mathbb{R}^n$. For each $U \in \mathfrak{O}_n$, we define

$$\begin{aligned}\mathsf{Diff}_{(U,n,k)} &\overset{\text{def}}{=} \left(\left\{\left(g : \mathbb{R}^n \to U, \mathfrak{D}^k g\right) : g \text{ is differentiable}\right\}, \left(U, \phi_{n,k}\left(U \times \left(\mathbb{R}^k\right)^n\right)\right), \text{incl.}\right) \\ &\in \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right).\end{aligned}$$

We define the $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$-monad $\mathcal{P}_{n,k}\left(-\right)_\perp$ on $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ by

$$\mathcal{P}_{n,k}\left(D, \left(C, C'\right), j\right)_\perp \overset{\text{def}}{=} \left(\underline{\mathcal{P}_{n,k}\left(D, \left(C, C'\right), j\right)_\perp}, \left(\left(C\right)_\perp, \left(C'\right)_\perp\right), \mathtt{j}_X\right) \tag{7.4}$$

where $\underline{\mathcal{P}_{n,k}\left(D, \left(C, C'\right), j\right)_\perp}$ is the union

$$\{\perp\} \sqcup G_{n,k}\left(C, C'\right) \sqcup \left(\coprod_{U \in \mathfrak{O}_n} \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)\left(\mathsf{Diff}_{(U,n,k)}, \left(D, \left(C, C'\right), j\right)\right)\right) \tag{7.5}$$

with the full $\boldsymbol{\omega}\mathbf{Cpo}$-substructure of $G_{n,k}\left(\left(C\right)_\perp, \left(C'\right)_\perp\right)$ induced by the inclusion $\mathtt{j}_X$ which is defined by the following components:

- the inclusion $\{\perp\} \to G_{n,k}\left(\left(C\right)_\perp, \left(C'\right)_\perp\right)$ of the least morphism
$$\perp : \left(\mathbb{R}^n, \left(\mathbb{R} \times \mathbb{R}^k\right)^n\right) \to \left(\left(C\right)_\perp, \left(C'\right)_\perp\right)$$
in $\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}\left(\left(\mathbb{R}^n, \left(\mathbb{R} \times \mathbb{R}^k\right)^n\right), \left(\left(C\right)_\perp, \left(C'\right)_\perp\right)\right)$;
- the inclusion of the total functions
$$G_{n,k}\left(\eta_C, \eta_{C'}\right) : G_{n,k}\left(C, C'\right) \to G_{n,k}\left(\left(C\right)_\perp, \left(C'\right)_\perp\right);$$
- for each $U \in \mathfrak{O}_n$, the injection
$$\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)\left(\mathsf{Diff}_{(U,n,k)}, \left(D, \left(C, C'\right), j\right)\right) \to G_{n,k}\left(\left(C\right)_\perp, \left(C'\right)_\perp\right)$$
defined by
$$\left(\alpha_0, \alpha_1 = \left(\beta_0 : U \to C, \beta_1 : \phi_{n,k}\left(U \times \left(\mathbb{R}^k\right)^n\right) \to C'\right)\right)$$
$$\mapsto$$
$$\left(\overline{\beta_0} : \mathbb{R}^n \to \left(C\right)_\perp, \overline{\beta_1} : \left(\mathbb{R} \times \mathbb{R}^k\right)^n \to \left(C'\right)_\perp\right),$$
where $\overline{\beta_0}$ and $\overline{\beta_1}$ are the respective corresponding canonical extensions.

For each $(C, C') \in \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}$, the component $(\mathrm{m}_C, \mathrm{m}_{C'})$ and $(\eta_C, \eta_{C'})$ of the multiplication and the unit of the monad $(-)_\perp$ on $\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}$ define morphisms

$$\begin{aligned} \overline{\mathrm{m}}_{(D,(C,C'),j)} &: \mathcal{P}_{n,k}\left(\mathcal{P}_{n,k}\left(D,\left(C,C'\right),j\right)_\perp\right)_\perp \to \mathcal{P}_{n,k}\left(D,\left(C,C'\right),j\right)_\perp \quad (7.6)\\ \overline{\eta}_{(D,(C,C'),j)} &: \left(D,\left(C,C'\right),j\right) \to \mathcal{P}_{n,k}\left(D,\left(C,C'\right),j\right)_\perp . \quad (7.7)\end{aligned}$$

in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$. Therefore, $\overline{\mathrm{m}}$ and $\overline{\eta}$ define the multiplication and the unit for $\mathcal{P}_{n,k}\left(-\right)_\perp$, completing the definition of our monad. Analogously, we lift, as morphisms of $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$, the strength of $(-)_\perp$, making $\mathcal{P}_{n,k}\left(-\right)_\perp$ into a strong monad (*i.e.* $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$-enriched monad).

In order to finish the proof that $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_\perp\right)$ is a *CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair, it is enough to see that, for any pair of objects $\left(D_0, \left(C_0, C_0'\right), j_0\right)$, $\left(D_1, \left(C_1, C_1'\right), j_1\right)$ of $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$, the least morphism $\perp : \left(C_0, C_0'\right) \to \left(\left(C_1\right)_\perp, \left(C_1'\right)_\perp\right)$, of $\boldsymbol{\omega}\mathbf{Cpo}\left(C_0, \left(C_1\right)_\perp\right) \times \boldsymbol{\omega}\mathbf{Cpo}\left(C_0', \left(C_1'\right)_\perp\right)$ defines the least morphism $\left(D_0, \left(C_0, C_0'\right), j_0\right) \to \mathcal{P}_{n,k}\left(D_1, \left(C_1, C_1'\right), j_1\right)_\perp$ in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$.

Finally, since the underlying endofunctor of the monad $\mathcal{P}_{n,k}\left(-\right)_\perp$, the multiplication and the identity are clearly lifted from $(-)_\perp$ through $\underline{\mathcal{L}}_{n,k}$ as defined above, we have:

Theorem 7.3. *For each* $(n, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$*,*

$$\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_\perp\right)$$

is a CBV $\boldsymbol{\omega}\mathbf{Cpo}$*-pair. Moreover,*

$$\underline{\mathcal{L}}_{n,k} : \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right) \to \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}$$

is a CBV $\boldsymbol{\omega}\mathbf{Cpo}$*-pair morphism between* $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_\perp\right)$ *and* $\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$*.*

Therefore, by Lemma 3.2, $\mathcal{U}_{\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right)$ is a *CBV* model morphism between the underlying *CBV* models of

$$\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_\perp\right) \text{ and } \left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right).$$

7.3. Logical relations as a *CBV* model morphism. *Henceforth, we assume that the macro* $\mathcal{D}$ *is sound for primitives (see Def. 5.5).* We establish the *CBV* model morphism (7.16). We start by establishing the logical relations' assignment.

Let $(n, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$*.* We define the object (7.8) in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$.

$$\overline{[\![\mathbf{real}]\!]}_{n,k} \overset{\text{def}}{=} \left(\left\{\left(f : \mathbb{R}^n \to \mathbb{R}, f^*\right) : f \text{ is differentiable, } f^* = \mathfrak{D}^k f\right\}, \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right), \text{incl.}\right) \quad (7.8)$$

For each $m \in \mathbb{N}$, $\mathrm{op} \in \mathrm{Op}_m$ and $c \in \mathtt{R}$, we define the morphisms (7.9), (7.10) and (7.11) in $\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}$, in which $\mathbb{D}$, $[\![-]\!]$ and $[\![-]\!]_k$ are the functors underlying the *CBV* model morphisms respectively defined in (4.10), (5.7) and (5.10).

$$\overline{[\![\mathbf{sign}]\!]}_k \overset{\text{def}}{=} \left(\mathsf{sign}, \mathfrak{d}^k(\mathsf{sign})\right) = (\mathsf{sign}, [\![\mathbb{D}(\mathbf{sign})]\!]_k) : \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right) \to \left((1 \sqcup 1)_\perp, (1 \sqcup 1)_\perp\right) \tag{7.9}$$

$$\overline{[\![\underline{c}]\!]}_k \overset{\text{def}}{=} \left(\mathsf{c}, \mathfrak{d}^k(\mathsf{c})\right) : (1, 1) \to \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right) \tag{7.10}$$

$$\overline{[\![\mathrm{op}]\!]}_k \overset{\text{def}}{=} \left([\![\mathrm{op}]\!], \mathfrak{d}^k([\![\mathrm{op}]\!])\right) : \left(\mathbb{R}^m, \left(\mathbb{R} \times \mathbb{R}^k\right)^m\right) \to \left((\mathbb{R})_\perp, \left(\mathbb{R} \times \mathbb{R}^k\right)_\perp\right) \tag{7.11}$$

By Theorem 6.4, we have that the product $\overline{[\![\mathbf{real}]\!]}^m_{n,k}$ in $\mathbf{Sub}(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k})$ is given by (7.12). Therefore, *by the chain rule for derivatives*, we have that (7.9), (7.10) and (7.11) respectively define the morphisms (7.13), (7.14), and (7.15) in $\mathbf{Sub}(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k})$, where $\overline{1} \sqcup \overline{1}$ denotes the coproduct of the terminal $\overline{1} = (1, (1,1), \mathrm{id})$ with itself.

$$\begin{aligned}&\left(\left\{\left(f_j : \mathbb{R}^n \to \mathbb{R}, f_j^*\right)_{j \in \mathbb{I}_m} : f_j^* \text{ is differentiable and } f_j^* = \mathfrak{D}^k f_j, \forall j \in \mathbb{I}_m\right\}, \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right)^m, \text{incl.}\right)\\ &\cong \left(\left\{(f : \mathbb{R}^n \to \mathbb{R}^m, f^*) : f \text{ is differentiable, } f^* = \mathfrak{D}^k f\right\}, \left(\mathbb{R}^m, \left(\mathbb{R} \times \mathbb{R}^k\right)^m\right), \text{incl.}\right).\end{aligned} \tag{7.12}$$

$$\overline{[\![\mathbf{sign}]\!]}_{n,k} : \overline{[\![\mathbf{real}]\!]}_{n,k} \to \mathcal{P}_{n,k}\left(\overline{1} \sqcup \overline{1}\right)_\perp \tag{7.13}$$

$$\overline{[\![\underline{c}]\!]}_{n,k} : \overline{1} \to \overline{[\![\mathbf{real}]\!]}_{n,k} \tag{7.14}$$

$$\overline{[\![\mathrm{op}]\!]}_{n,k} : \overline{[\![\mathbf{real}]\!]}^m_{n,k} \to \mathcal{P}_{n,k}\left(\overline{[\![\mathbf{real}]\!]}_{n,k}\right)_\perp \tag{7.15}$$

By the universal property of the *CBV* model $\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right)$, we get:

Theorem 7.4. *For each* $(n, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$*, there is only one CBV model morphism*

$$\overline{[\![-]\!]}_{n,k} : \left(\mathbf{Syn}^{\mathbf{tr}}_V, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_\mu, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right) \to \mathcal{U}_{\mathcal{BV}}\left(\mathbf{Sub}(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}), \mathcal{P}_{n,k}(-)_\perp\right) \tag{7.16}$$

that is consistent with the assignment given by (7.8)*,* (7.13)*,* (7.15)*, and* (7.14)*. Moreover, Diag.* (7.17) *commutes.*

$$\begin{tikzcd}
\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right) \arrow[r, "(\mathrm{id}, \mathbb{D})"] \arrow[d, "\overline{[\![-]\!]}_{n,k}"'] & \left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right) \times \left(\mathbf{Syn}^{\mathbf{tr}}_V, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_\mu, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right) \arrow[d, "{[\![-]\!] \times [\![-]\!]_k}"] \\
\mathcal{U}_{\mathcal{BV}}\left(\mathbf{Sub}(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}), \mathcal{P}_{n,k}(-)_\perp\right) \arrow[r, "\mathcal{U}_{\mathcal{BV}}(\underline{\mathcal{L}}_{n,k})"'] & \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)
\end{tikzcd}$$

(7.17)

Proof: Both $(\llbracket - \rrbracket \times \llbracket - \rrbracket_k) \circ (\mathrm{id} \times \mathbb{D})$ and $\mathcal{U}_{\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right) \circ \overline{\llbracket - \rrbracket}_{n,k}$ yield CBV model morphisms that are consistent with the assignment given by the object $\left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right)$ together with (7.9), (7.10) and (7.11). ■

7.4. AD Logical Relations for Data Types. As a consequence of Theorem 7.4, we establish a fundamental result on the logical relations $\overline{\llbracket - \rrbracket}_{n,k}$ for data types in our setting: namely, Theorem 7.6. We start by establishing Lemma 7.5 about our logical relations and the coproducts in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$.

Lemma 7.5. *Let $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. If $(g, \dot{g}) \in \coprod_{j \in L} \overline{\llbracket \mathbf{real} \rrbracket}^{l_j}_{n,k}$, then $g : \mathbb{R}^n \to \coprod_{j \in L} \mathbb{R}^{l_j}$ is differentiable and $\dot{g} = \mathfrak{D}^k g$.*

Proof: By Theorem 7.2, $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ has coproducts. Moreover, we can conclude that $(g, \dot{g}) \in \coprod_{j \in L} \overline{\llbracket \mathbf{real} \rrbracket}^{l_j}_{n,k}$ implies that, for some $r \in L$, we have a pair

$$\left(\underline{g} : \mathbb{R}^n \to \mathbb{R}^{l_r}, \mathfrak{D}^k g : \left(\mathbb{R} \times \mathbb{R}^k\right)^n \to \left(\mathbb{R} \times \mathbb{R}^k\right)^{l_r}\right) \tag{7.18}$$

such that $(g, \dot{g}) = \left(\iota_{\mathbb{R}^{l_r}} \circ \underline{g}, \iota_{\left(\mathbb{R} \times \mathbb{R}^k\right)^{l_r}} \circ \mathfrak{D}^k g\right)$. Following Def. 5.2, this completes our proof. ■

Theorem 7.6. *Let $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. If $(g, \dot{g}) \in \underline{\mathcal{P}_{n,k}\left(\coprod_{j \in L} \overline{\llbracket \mathbf{real} \rrbracket}^{l_j}_{n,k}\right)_{\perp}}$, then $g : \mathbb{R}^n \to \left(\coprod_{j \in L} \mathbb{R}^{l_j}\right)_{\perp}$ is differentiable and $\dot{g} = \mathfrak{d}^k(g)$.*

Proof: Indeed, by the definition of $\underline{\mathcal{P}_{n,k}\left(-\right)_{\perp}}$, we have one of the following situations.

s1. g and $\dot{g}$ are the least morphisms, that is to say, they are constantly equal to $\perp$;

s2. the pair $(g, \dot{g})$ come from a pair of total functions $\left(\underline{g}, \underline{\dot{g}}\right) \in \coprod_{j \in L} \overline{\llbracket \mathbf{real} \rrbracket}^{l_j}_{n,k}$;

s3. $g^{-1}\left(\coprod_{j\in L}\mathbb{R}^{l_j}\right) = W$ is open. Moreover, denoting by (7.19) the pair consisting of the corresponding total functions, we have that (7.20) holds for any differentiable map $\alpha : \mathbb{R}^n \to W$.

$$\left(\underline{g} : W \to \left(\coprod_{j\in L}\mathbb{R}^{l_j}\right), \underline{\dot{g}}\right) \tag{7.19}$$

$$\left(\underline{g}\circ\alpha,\ \underline{\dot{g}}\circ\mathfrak{D}^k\alpha\right) \in \coprod_{j\in L}\overline{[\![\mathbf{real}]\!]}^{l_j}_{n,k}. \tag{7.20}$$

If (**s**1.) holds, following Def. 5.5, we get that g is differentiable and $\dot{g} = \mathfrak{d}^k(g)$ by Remark 5.3.

In case of (**s**2.), we get $\underline{g}$ is differentiable and $\underline{\dot{g}} = \mathfrak{D}^k\underline{g}$ by Lemma 7.5. Hence g is differentiable and $\dot{g} = \mathfrak{d}^k(g)$.

Finally, in case of (**s**3.), by Lemma 7.5, we get that, for any differentiable $\alpha : \mathbb{R}^n \to W$, $\underline{g}\circ\alpha$ is differentiable and $\underline{\dot{g}}\circ\mathfrak{D}^k\alpha$ is well defined and equal to $\mathfrak{D}^k\left(\underline{g}\circ\alpha\right)$. By Lemma 5.4, this implies that $\underline{g}$ is differentiable and $\mathfrak{D}^k\underline{g} = \underline{\dot{g}}$. Following Def. 5.5, this completes the proof that g is differentiable and $\dot{g} = \mathfrak{d}^k(g)$. ■

Corollary 7.7. *Let $k\in\mathbb{N}\cup\{\infty\}$. If, for each $i\in\mathfrak{L}$, the morphism $(g,\dot{g})$ in $\boldsymbol{\omega}\mathbf{Cpo}\times\boldsymbol{\omega}\mathbf{Cpo}$ defines the morphism (7.21) in $\mathbf{Sub}(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{s_i,k})$, then*

$$g : \coprod_{r\in\mathfrak{L}}\mathbb{R}^{s_r} \to \left(\coprod_{j\in L}\mathbb{R}^{l_j}\right)_{\perp} \quad \textit{is differentiable and } \dot{g} = \mathfrak{d}^k(g).$$

$$\mathsf{g} : \coprod_{r\in\mathfrak{L}}\overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L}\overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_{\perp} \tag{7.21}$$

$$\iota_i : \overline{[\![\mathbf{real}]\!]}^{s_i}_{s_i,k} \to \coprod_{r\in K}\overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \tag{7.22}$$

Proof: From the hypothesis, for each $i\in\mathfrak{L}$, we conclude that the pair (7.23) defines the morphism (7.24), since $\left(\iota_{\mathbb{R}^{s_i}}, \iota_{(\mathbb{R}\times\mathbb{R}^k)^{s_i}}\right)$ defines the coprojection (7.22) in $\mathbf{Sub}(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{s_i,k})$.

$$\left(g_i \overset{\text{def}}{=} g\circ\iota_{\mathbb{R}^{s_i}},\ \dot{g}_i \overset{\text{def}}{=} \dot{g}\circ\iota_{(\mathbb{R}\times\mathbb{R}^k)^{s_i}}\right) \tag{7.23}$$

$$\mathsf{g}_i \overset{\text{def}}{=} \mathsf{g}\circ\iota_i : \overline{[\![\mathbf{real}]\!]}^{s_i}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L}\overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_{\perp} \tag{7.24}$$

Since $\mathrm{id}_{\mathbb{R}^{s_i}} : \mathbb{R}^{s_i} \to \mathbb{R}^{s_i}$ is differentiable, and $\mathfrak{D}^k(\mathrm{id}_{\mathbb{R}^{s_i}})$ is given by the identity $\left(\mathbb{R} \times \mathbb{R}^k\right)^{s_i} \to \left(\mathbb{R} \times \mathbb{R}^k\right)^{s_i}$, we conclude that

$$(g_i, \dot{g}_i) \in \underline{\mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_{\perp}} . \tag{7.25}$$

By Theorem7.6, (7.25) proves that g_i is differentiable and $\dot{g}_i = \mathfrak{d}^k\,(g_i)$. Since this result holds for any $i \in \mathfrak{L}$, we conclude that g is differentiable and $\dot{g} = \mathfrak{d}^k\,(g)$. ■

7.5. Fundamental AD correctness theorem. We prove Theorem 7.8, which completes the proof of Theorem 7.1.

Theorem 7.8. *Let* $t : \coprod_{r\in\mathfrak{L}} \mathbf{real}^{s_r} \to \mathbf{Syn}_{\mathcal{S}}\left(\coprod_{j\in L} \mathbf{real}^{l_j}\right)$ *be a morphism in* $\mathbf{Syn}_V$. *We have that* $[\![t]\!] : \coprod_{r\in\mathfrak{L}} \mathbb{R}^{s_r} \to \left(\coprod_{j\in L} \mathbb{R}^{l_j}\right)_{\perp}$ *is differentiable and, for any* $k \in (\mathbb{N} \cup \{\infty\})$, $[\![\mathbb{D}\,(t)]\!]_k = \mathfrak{d}^k\,([\![t]\!])$.

Proof: We assume that we have t as above. For each $i \in \mathfrak{L}$, the pair (7.26) is in the image of $([\![-]\!] \times [\![-]\!]_k) \circ (\mathrm{id} \times \mathbb{D}) = \mathcal{U}_{\mathcal{BV}}\left(\underline{\mathcal{L}}_{s_i,k}\right) \circ \overline{[\![-]\!]}_{s_i,k}$. This implies that (7.26) defines the morphism (7.27) in $\mathbf{Sub}\,(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{s_i,k})$. Therefore, by Corollary 7.7, we conclude that $[\![t]\!]$ is differentiable and $[\![\mathbb{D}(t)]\!]_k = \mathfrak{d}^k\,([\![t]\!])$.

$$([\![t]\!], [\![\mathbb{D}(t)]\!]_k) \quad (7.26) \qquad \overline{[\![t]\!]}_{s_i,k} : \coprod_{r\in K} \overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_{\perp} \tag{7.27}$$
■

7.6. Correctness of the dual numbers forward AD. We assume that **vect** implements the vector space $\mathbb{R}$. It is straightforward to see that we get forward mode AD out of our macro $\mathcal{D}$: namely, for a program $x : \tau \vdash t : \sigma$ (where τ and σ are data types) in the source language, we get a program $x : \mathcal{D}(\tau) \vdash \mathcal{D}(t) : \mathcal{D}(\sigma)$ in the target language, which, by Theorem 7.1, satisfies the following properties.

- $[\![t]\!] : \coprod_{r\in K}\mathbb{R}^{n_r} \to \left(\coprod_{j\in L}\mathbb{R}^{m_j}\right)_\perp$ is differentiable as in Def. 5.5;
- if $y \in \mathbb{R}^{n_i} \cap [\![t]\!]^{-1}(\mathbb{R}^{m_j}) = W_j$ for some $i \in K$ and $j \in L$, we have that, for any $w \in \mathbb{R}^{n_i}$, denoting $z := \phi_{n_i,1}(y,w)$,

$$\begin{aligned}[\![\mathcal{D}(t)]\!]_1(\phi_{n_i,1}(y,w)) &= \mathfrak{d}^1([\![t]\!])(z) = \mathfrak{D}^1[\![t]\!]|_{W_j}(z) = \phi_{m_j,1}\left([\![t]\!](y), \tilde{w}\cdot[\![t]\!]'(y)^t\right)\\ &= \phi_{l,1}([\![t]\!](y), [\![t]\!]'(y)(w)), \qquad (7.28)\end{aligned}$$

where $[\![t]\!]'(y) : \mathbb{R}^{n_i} \to \mathbb{R}^{m_j}$ is the derivative of $[\![t]\!]|_{W_j} : W_j \to \mathbb{R}^{m_j}$ at y.

7.7. Correctness of the dual numbers reverse AD. We assume that **vect** implements the vector space $\mathbb{R}^k$, for some fixed $k \in \mathbb{N}\cup\{\infty\}$. We consider the respective (co)projections $\mathfrak{p}_{k\to s}$ for each $s \in \mathbb{N}\cup\{\infty\}$, as defined in (4.5) . The following shows how our macro encompasses reverse mode AD.

For each $s \in \mathbb{N}^*$ with $s \le k$, we can define the morphism $\mathbf{wrap}_s \overset{\text{def}}{=} (\pi_j, \overline{e}_j)_{j\in\mathbb{I}_s} : \mathbf{real}^s \to (\mathbf{real}\times\mathbf{vect})^s$ in $\mathbf{Syn}^{\mathbf{tr}}_V$, which corresponds to the wrapper defined in (1.2) in the target language. We denote $\mathtt{wrap}_s \overset{\text{def}}{=} [\![\mathbf{wrap}_s]\!]_k$. By the definition of the k-semantics, it is clear that $\mathtt{wrap}_s(y) = \phi_{s,k}\left(y, e^k_1, \ldots, e^k_s\right)$.

For a program $x : \mathbf{real}^s \vdash t : \mathbf{real}^l$ (where $s, l \in \mathbb{N}^*$), we have that, for any $y \in [\![t]\!]^{-1}\left(\mathbb{R}^l\right) \subseteq \mathbb{R}^s$,

$$\begin{aligned}[\![\mathcal{D}(t)\circ\mathbf{wrap}_s]\!]_k(y) &= \mathfrak{d}^k([\![t]\!])\circ\mathtt{wrap}_s(y) = \mathfrak{D}^k[\![t]\!]\circ\mathtt{wrap}_s(y)\\ &= \mathfrak{D}^k[\![t]\!]\circ\phi_{s,k}\left(y, e^k_1,\ldots,e^k_s\right)\\ &= \phi_{l,k}\left([\![t]\!](y), \mathfrak{p}_{s\to k}[\![t]\!]'(y)^t\right)\end{aligned}$$

by Theorem 7.1. This gives the transpose derivative $\mathfrak{p}_{s\to k}[\![t]\!]'(y)^t$ as something of the type $\mathbf{vect}^l$. This should be good enough whenever $k = s$, since, in this case, $[\![\mathbf{vect}^l]\!]_k = (\mathbb{R}^s)^l$ and $\mathfrak{p}_{s\to k} = \mathfrak{p}_{k\to k} = \mathrm{id}$.

In case of $s < k$, if needed, the type can be fixed by using the handler $\mathfrak{h}_s$. More precisely, we can define the morphism

$$\mathfrak{h}_{l,s} \overset{\text{def}}{=} (\mathrm{id}, \mathfrak{h}_s)_{i\in\mathbb{I}_l} : (\mathbf{real}\times\mathbf{vect})^l \to (\mathbf{real}\times\mathbf{real}^s)^l$$

and, by the definition of k-semantics, we conclude that

$$\begin{aligned}[\![\mathfrak{h}_{l,s}\circ\mathcal{D}(t)\circ\mathbf{wrap}_s]\!]_k(y) &= [\![\mathfrak{h}_{l,s}]\!]_k\circ\phi_{l,k}\left([\![t]\!](y), \mathfrak{p}_{s\to k}[\![t]\!]'(y)^t\right)\\ &= \phi_{l,k}\left([\![t]\!](y), \mathfrak{p}_{k\to s}\circ\mathfrak{p}_{s\to k}[\![t]\!]'(y)^t\right)\\ &= \phi_{l,k}\left([\![t]\!](y), [\![t]\!]'(y)^t\right),\end{aligned}$$

since $\mathfrak{p}_{k\to s}\circ\mathfrak{p}_{s\to k} = \mathrm{id}$ whenever $s \le k$.

Again, by Theorem 7.1, it is straightforward to generalize the correctness statements above to more general data types σ. Furthermore, it should be noted that, for $k = \infty$ (representing the case of a type of dynamically sized array of cotangents), the above shows that our macro gives the reverse mode AD for any program $x : \tau \vdash t : \sigma$ for data types τ and σ. This choice of $k = \infty$ is the easiest route to take for a practical implementation of this form of dual-numbers reverse AD, as it leads to a single type of cotangent vectors that works for any program.

8. AD for recursive types and ML-polymorphism

8.1. Syntax. We extend both our source and target languages of 4.1 and 4.2 with ML-style polymorphism and type recursion in the sense of FPC [13]. That is, we extend types, values and computations for each of the two languages as

$$\begin{array}{llll|ll}
\tau, \sigma, \rho & ::= & \text{types} & & \alpha, \beta, \gamma & \text{type variables} \\
& \mid \ \ldots & \text{as before} & & \mu\alpha.\tau & \text{recursive type} \\
\\
v, w, u & ::= & \text{values} & & \mathbf{roll}\, v & \text{recursive intro} \\
& \mid \ \ldots & \text{as before} & & & \\
\\
t, s, r & ::= & \text{computations} & & \mathbf{roll}\, t & \text{recursive intro} \\
& \mid \ \ldots & \text{as before} & & \mathbf{case}\, t\, \mathbf{of}\, \mathbf{roll}\, x \to s & \text{recursive elim}
\end{array}$$

The new values and computations according to the rules in Fig. 8.1.

$$\frac{\Delta \mid \Gamma \vdash t : \sigma[{}^{\mu\alpha.\sigma}/_{\alpha}]}{\Delta \mid \Gamma \vdash \mathbf{roll}\, t : \mu\alpha.\sigma} \qquad \frac{\Delta \mid \Gamma \vdash t : \mu\alpha.\sigma \quad \Delta \mid \Gamma, x : \sigma[{}^{\mu\alpha.\sigma}/_{\alpha}] \vdash s : \tau}{\Delta \mid \Gamma \vdash \mathbf{case}\, t\, \mathbf{of}\, \mathbf{roll}\, x \to s : \tau}$$

FIGURE 8.1. Typing rules for the recursive types extension.

Here, kinding contexts Δ are lists of type variables $\alpha_1, \ldots, \alpha_n$. We consider judgements $\Delta \mid \Gamma \vdash t : \tau$, where the types in Γ and τ may contain free type variables from Δ. They should be read as specifying that t is a program of type τ, with free variables typed according to Γ, that is polymorphic in the type variables of Δ.

We use the $\beta\eta$-rules of Fig. 8.2.

Once a language has recursive types, it is already expressive enough to get term recursion and, hence, iteration. Namely, we can now consider term recursion at type $\tau = \sigma \to \rho$ as syntactic sugar. Namely, we first define

$$\boxed{\textbf{case}\,\textbf{roll}\,v\,\textbf{of}\,\textbf{roll}\,x \to t = t[{}^{v}\!/\!_{x}] \quad t[{}^{v}\!/\!_{z}] \overset{\#x}{=} \textbf{case}\,v\,\textbf{of}\,\textbf{roll}\,x \to t[{}^{\textbf{roll}\,x}\!/\!_{z}]}$$

FIGURE 8.2. The standard $\beta\eta$-equational theory for recursive types in CBV.

$\chi \overset{\mathrm{def}}{=} \mu\alpha.\,(\alpha \to \tau)$ and then:

$$\textbf{unroll}\,t \overset{\mathrm{def}}{=} \textbf{case}\,t\,\textbf{of}\,\textbf{roll}\,x \to x$$

$$\mu x : \tau.t \overset{\mathrm{def}}{=} \textbf{let}\,\mathit{body} : \chi \to \tau = \,(\lambda y : \chi.\lambda z : \sigma.\textbf{let}\,x : \tau = \,\textbf{unroll}\,y\,y\,\textbf{in}\,t\,z)\,\textbf{in}\,\mathit{body}(\textbf{roll}\,\mathit{body}). \quad (8.3)$$

The semantics of the language is, of course, expected to be consistent – meaning that term recursion should be compatible with the definition above. Alternatively, we can consider that the source language is given by the basic language with the typing rules given by Fig. 4.1 with the corresponding grammar plus the recursive types established above, while the target language is the source language plus the extension given by the grammar and typing rules defined in 4.2.

8.2. Categorical models for recursive types: ***rCBV*** **models.** Here, we establish the basic categorical model for the syntax of call-by-value languages with recursive types. *Let $(\mathcal{V},\mathcal{T})$ be a CBV pair and $J : \mathcal{V} \to \mathcal{C}$ the corresponding universal Kleisli functor.* Moreover, let $\mathsf{Cat}\,(2, \mathcal{V}\text{-}\mathsf{Cat})$ be the category of morphisms of $\mathcal{V}$-Cat.

For each $n \in \mathbb{N}$, an *n-variable $(\mathcal{V},\mathcal{T})$-parametric type* (or a $(\mathcal{V},\mathcal{T})$-parametric type of degree n) is a morphism $E : (J^{\mathrm{op}} \times J)^n \to J$ in $\mathsf{Cat}\,(2, \mathcal{V}\text{-}\mathsf{Cat})$. In other words, it consists of a pair $E = (E_{\mathcal{V}}, E_{\mathcal{C}})$ of $\mathcal{V}$-enriched functors such that (8.4) commutes. *A $(\mathcal{V},\mathcal{T})$-parametric type of degree* 0 (8.6) *can be identified with the corresponding object $\mathcal{V}$.*

$$\begin{tikzcd}
(\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^n \arrow[r, "E_{\mathcal{C}}"] & \mathcal{C} \\
(\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n \arrow[r, "E_{\mathcal{V}}"'] \arrow[u, "(J^{\mathrm{op}}\times J)^n"] & \mathcal{V} \arrow[u, "J"']
\end{tikzcd} \quad (8.4)$$

We denote by $\mathsf{Param}\,(\mathcal{V},\mathcal{T})$ the collection of all $(\mathcal{V},\mathcal{T})$-parametric types $E = (E_{\mathcal{V}}, E_{\mathcal{C}})$ of any degree $n \in \mathbb{N}$. As the terminology indicates, the objects of $\mathsf{Param}\,(\mathcal{V},\mathcal{T})$ play the role of the parametric types in our language. However, the parametric types in the actual language could be a bit more restrictive. They usually are those constructed out of the primitive type formers. Namely, in our case, tupling (finite products), cotupling (finite coproducts), exponetiation (Kleisli exponential) and type recursion.

Definition 8.1 (Free type recursion)**.** A *free decreasing degree type operator* (*fddt* operator) for $(\mathcal{V}, \mathcal{T})$ is a function (8.5) identity on parametric types of degree 0 which takes each $(n+1)$-variable $(\mathcal{V}, \mathcal{T})$-parametric type $E = (E_{\mathcal{V}}, E_{\mathcal{C}})$ to a $(\mathcal{V}, \mathcal{T})$-parametric type $\nu E = (\nu E_{\mathcal{V}}, \nu E_{\mathcal{C}})$ of degree n, provided that $n \in \mathbb{N}$.

$$\nu : \mathsf{Param}\,(\mathcal{V}, \mathcal{T}) \;\to\; \mathsf{Param}\,(\mathcal{V}, \mathcal{T}) \tag{8.5}$$

$$\begin{tikzcd}
(\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{n+1} \arrow[r, "E_{\mathcal{C}}"] & \mathcal{C} & & (\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{n} \arrow[r, "\nu E_{\mathcal{C}}"] & \mathcal{C} \\
(\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n+1} \arrow[u, "(J^{\mathrm{op}}\times J)^{n+1}"] \arrow[r, "E_{\mathcal{V}}"'] & \mathcal{V} \arrow[u, "J"'] \arrow[r, phantom, "\mapsto"] & & (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n} \arrow[u, "(J^{\mathrm{op}}\times J)^{n}"] \arrow[r, "\nu E_{\mathcal{V}}"'] & \mathcal{V} \arrow[u, "J"']
\end{tikzcd}$$

A *rolling* for (8.5) is a collection (8.7) of natural transformations such that (8.8) is invertible for any $E = (E_{\mathcal{V}}, E_{\mathcal{C}})$, that is to say, $J\left(\mathsf{roll}^{E}\right)$ is a natural isomorphism.

$$\left((\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{0} \to \mathcal{V}, (\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{0} \to \mathcal{C}\right) \tag{8.6}$$

$$\underline{\mathsf{roll}} = \left(\mathsf{roll}^{E}\right)_{E=(E_{\mathcal{V}}, E_{\mathcal{C}}) \in \mathsf{Param}(\mathcal{V}, \mathcal{T})} \tag{8.7}$$

$$\begin{tikzcd}
 & (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n} \arrow[r, "(\mathrm{id}, \nu E_{\mathcal{V}}^{\mathrm{op}}, \nu E_{\mathcal{V}})"] \arrow[dr, "\nu E_{\mathcal{V}}"'] & (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n+1} \arrow[d, "E_{\mathcal{V}}"] \arrow[dl, phantom, "\overset{\mathsf{roll}^{E}}{\Longleftarrow}"] \\
\mathcal{C} & & \mathcal{V} \arrow[ll, "J"]
\end{tikzcd} \tag{8.8}$$

A *free type recursion* for $(\mathcal{V}, \mathcal{T})$ is a pair $\underline{\nu} = (\nu, \underline{\mathsf{roll}})$ where ν is an *fddt* operator and $\underline{\mathsf{roll}}$ is a rolling for ν.

Definition 8.2 (H-compatible)**.** Let H be a CBV pair morphism between CBV pairs $(\mathcal{V}, \mathcal{T})$ and $(\mathcal{V}', \mathcal{T}')$. A pair $(E, E') \in \mathsf{Param}\,(\mathcal{V}, \mathcal{T}) \times \mathsf{Param}\,(\mathcal{V}', \mathcal{T}')$ of parametric types is *H-compatible* if they have the same degree n and the diagram (8.9) commutes. In particular, if $n = 0$, the pair (E, E') is H-compatible if $H(E_{\mathcal{V}}) = E'_{\mathcal{V}}$.

$$\begin{tikzcd}
(\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n \arrow[r, "E_{\mathcal{V}}"] \arrow[d, "(H^{\mathrm{op}}\times H)^n"'] & \mathcal{V} \arrow[d, "H"] \\
(\mathcal{V}'^{\mathrm{op}} \times \mathcal{V}')^n \arrow[r, "E'_{\mathcal{V}'}"'] & \mathcal{V}'
\end{tikzcd} \tag{8.9}$$

Definition 8.3 (*rCBV* models)**.** An *rCBV model* is a triple $(\mathcal{V}, \mathcal{T}, \underline{\nu})$ where $(\mathcal{V}, \mathcal{T})$ is a *CBV* pair and $\underline{\nu}$ is a free type recursion for $(\mathcal{V}, \mathcal{T})$.

An *rCBV model morphism* between the *rCBV* models $(\mathcal{V}, \mathcal{T}, \underline{\nu})$ and $(\mathcal{V}', \mathcal{T}', \underline{\nu}')$ consists of a *CBV* pair morphism between $(\mathcal{V}, \mathcal{T})$ and $(\mathcal{V}', \mathcal{T}')$ such that, for every H-compatible pair $(E, E') \in \mathsf{Param}(\mathcal{V}, \mathcal{T}) \times \mathsf{Param}(\mathcal{V}', \mathcal{T}')$ of n-variable parametric types, $(\nu E, \nu E')$ is H-compatible and, if $n > 0$, (8.10) holds, that is to say, $H\left(\mathsf{roll}^E\right) = \mathsf{roll}^E_{(H^{\mathrm{op}}\times H)^{n-1}}$. The *rCBV* models and *rCBV* model morphisms define a category, *denoted herein by* $\mathfrak{C}_{\mathcal{RBV}}$.

$$\begin{tikzcd}
(\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n-1} \arrow[r, "(\mathrm{id}, \nu E^{\mathrm{op}}_{\mathcal{V}}, \nu E_{\mathcal{V}})"] \arrow[dr, "\nu E_{\mathcal{V}}"'] & (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n \arrow[d, "E_{\mathcal{V}}"] \arrow[dl, phantom, "\overset{\mathsf{roll}^E}{\Longleftarrow}"] \\
\mathcal{V}' & \mathcal{V} \arrow[l, "H"]
\end{tikzcd}
=
\begin{tikzcd}
(\mathcal{V}'^{\mathrm{op}} \times \mathcal{V}')^{n-1} \arrow[r, "(\mathrm{id}, \nu E'^{\mathrm{op}}_{\mathcal{V}'}, \nu E'_{\mathcal{V}'})"] \arrow[dr, "\nu E'_{\mathcal{V}'}"'] & (\mathcal{V}'^{\mathrm{op}} \times \mathcal{V}')^n \arrow[d, "E_{\mathcal{V}'}"] \arrow[dl, phantom, "\overset{\mathsf{roll}^{E'}}{\Longleftarrow}"] \\
(\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n-1} \arrow[u, "(H^{\mathrm{op}}\times H)^{n-1}"] & \mathcal{V}'
\end{tikzcd} \tag{8.10}$$

There is, then, an obvious forgetful functor $\mathcal{U}_{\mathrm{rp}} : \mathfrak{C}_{\mathcal{RBV}} \to \mathfrak{C}_{\mathsf{p}}$.

Remark 8.4. We do not use this fact in our work, but every *rCBV* model has an underlying *CBV* model. More precisely, free term iteration can be defined out of the free term recursion, while the latter can be defined out of the free type recursion (see (8.3)). This defines a forgetful functor

$$\mathcal{R} : \mathfrak{C}_{\mathcal{RBV}} \to \mathfrak{C}_{\mathcal{BV}}. \tag{8.11}$$

8.3. The syntactic ***rCBV*** **models.** We consider the *rCBV* model generated by each syntax, that is to say, the free *rCBV* models coming from the fine-grain CBV translations of the source and target languages. This provides us with the *rCBV* models

$$\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right) \qquad \text{and} \qquad \left(\mathbf{Syn}^{\mathsf{Rtr}}_{V}, \mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}}, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right) \tag{8.12}$$

with the universal property described in Theorem 8.5.

Theorem 8.5 (Universal Property of the *rCBV* models (8.12))**.** *Let* $(\mathcal{V},\mathcal{T},\underline{\nu})$ *be an rCBV model. Assume that Fig. 4.7 and Fig. 4.8 are given consistent assignments.*

(1) There is a unique rCBV model morphism $H : \left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right) \to (\mathcal{V},\mathcal{T},\underline{\nu})$ *respecting the assignment of Fig. 4.7.*
(2) There is a unique rCBV model morphism $\mathcal{H} : \left(\mathbf{Syn}^{\mathsf{Rtr}}_{V}, \mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}}, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right) \to (\mathcal{V},\mathcal{T},\underline{\nu})$ *that extends* H *and respects the assignment of Fig. 4.8.*

Remark 8.6. By Theorem 4.1, we have (unique) *CBV* model morphisms (8.13) and (8.14) that are identity on the primitive operations and types.

$$\mathsf{s} : \left(\mathbf{Syn}_{V}, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}\right) \to \mathcal{R}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right) \quad (8.13)$$

$$\mathsf{s}^{\mathsf{t}} : \left(\mathbf{Syn}^{\mathbf{tr}}_{V}, \mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}}, \mathbf{Syn}^{\mathbf{tr}}_{\mu}, \mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right) \to \mathcal{R}\left(\mathbf{Syn}^{\mathsf{Rtr}}_{V}, \mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}}, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right) \quad (8.14)$$

Theorem 8.5 states that $H \mapsto \mathcal{R}(H) \circ \mathsf{s}$ and $\mathcal{H} \mapsto \mathcal{R}(\mathcal{H}) \circ \mathsf{s}^{\mathsf{t}}$ give the bijections (8.15) and (8.16), respectively.

$$\mathfrak{C}_{\mathcal{RBV}}\left(\left(\mathbf{Syn}^{\mathsf{R}}_{V},\mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}},\underline{\nu}_{\mathbf{Syn}}\right),(\mathcal{V},\mathcal{T},\underline{\nu})\right) \cong \mathfrak{C}_{\mathcal{BV}}\left(\left(\mathbf{Syn}_{V},\mathbf{Syn}_{\mathcal{S}},\mathbf{Syn}_{\mu},\mathbf{Syn}_{\mathsf{it}}\right),\mathcal{R}(\mathcal{V},\mathcal{T},\underline{\nu})\right) \quad (8.15)$$

$$\mathfrak{C}_{\mathcal{RBV}}\left(\left(\mathbf{Syn}^{\mathsf{Rtr}}_{V},\mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}},\underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right),(\mathcal{V},\mathcal{T},\underline{\nu})\right) \cong \mathfrak{C}_{\mathcal{BV}}\left(\left(\mathbf{Syn}^{\mathbf{tr}}_{V},\mathbf{Syn}^{\mathbf{tr}}_{\mathcal{S}},\mathbf{Syn}^{\mathbf{tr}}_{\mu},\mathbf{Syn}^{\mathbf{tr}}_{\mathsf{it}}\right),\mathcal{R}(\mathcal{V},\mathcal{T},\underline{\nu})\right) \quad (8.16)$$

8.4. Automatic differentiation for languages with recursive types. We extend our definition of AD to recursive types in Fig. 8.17. We note that our extension is compatible with our previous definitions if we view term recursion (and iteration) as syntactic sugar.

Lemma 8.7 (Type preservation)**.** *If* $\Delta \mid \Gamma \vdash t : \tau$*, then* $\Delta \mid \mathcal{D}(\Gamma) \vdash \mathcal{D}(t) : \mathcal{D}(\tau)$*.*

$$\mathcal{D}(\alpha) \stackrel{\mathrm{def}}{=} \alpha \qquad \mathcal{D}(\mu\alpha.\tau) \stackrel{\mathrm{def}}{=} \mu\alpha.\mathcal{D}(\tau)$$

$$\mathcal{D}(\mathbf{roll}\, t) \stackrel{\mathrm{def}}{=} \mathbf{roll}\, \mathcal{D}(t) \qquad \mathcal{D}(\mathbf{case}\, t\, \mathbf{of}\, \mathbf{roll}\, x \to s) \stackrel{\mathrm{def}}{=} \mathbf{case}\, \mathcal{D}(t)\, \mathbf{of}\, \mathbf{roll}\, x \to \mathcal{D}(s)$$

FIGURE 8.17. The definitions of AD on recursive types.

8.5. AD transformation as an ***rCBV*** **model morphism.** By Theorem 8.5, the assignment defined in Fig. 4.11 induces a unique *rCBV* model morphism (8.18), which *encompasses the macro* $\mathcal{D}$ *defined by Fig. 4.9 and extended in Fig. 8.17.*

$$\mathbb{D} : \left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \left(\mathbf{Syn}^{\mathsf{Rtr}}_{V}, \mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}}, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right). \quad (8.18)$$

8.6. Concrete models: *rCBV* $\boldsymbol{\omega}$**Cpo-pairs.** Although the setting of *bilimit compact expansions* is the usual reasonable basic framework for solving recursive domain equations, we do not need this level of generality. Instead, we consider a subclass of concrete models, the *rCBV* $\boldsymbol{\omega}$**Cpo**-pairs established in Def. 8.8.[g]

We are back again to the setting of $\boldsymbol{\omega}$**Cpo**-enriched categories. Recall that an *embedding-projection-pair (ep-pair)* $u : A \overset{\hookrightarrow}{\leftarrow} B$ in an $\boldsymbol{\omega}$**Cpo**-category $\mathcal{C}$ is a pair $u = (u^e, u^p)$ consisting of a $\mathcal{C}$-morphism $u^e : A \to B$, the *embedding*, and a $\mathcal{C}$-morphism $u^p : B \to A$, the *projection*, such that $u^e \circ u^p \leq \mathrm{id}$ and $u^p \circ u^e = \mathrm{id}$.

It should be noted that, when considering the underlying 2-category of the $\boldsymbol{\omega}$**Cpo**-category, an ep-pair consists of an adjunction[h] whose unit is the identity. In this context, it is also called a lari adjunction (*left adjoint right-inverse*), see [8, Sect. 1]. In particular, as in the case of any adjunction, an embedding $u^e : A \to B$ uniquely determines the associated projection $u^p : B \to A$ and vice-versa.

A zero object[i] $\mathfrak{O}$ in an $\boldsymbol{\omega}$**Cpo**-category $\mathcal{C}$ is an *ep-zero object* if, for any object A, the pair $\iota_A = (\iota^e : \mathfrak{O} \to A, \iota^p : A \to \mathfrak{O})$ consisting of the unique morphisms is an ep-pair.

Definition 8.8 (*rCBV* $\boldsymbol{\omega}$**Cpo**-pair)**.** An *rCBV* $\boldsymbol{\omega}$**Cpo**-pair is a *CBV* pair $(\mathcal{V}, \mathcal{T})$ such that, denoting by $J : \mathcal{V} \to \mathcal{C}$ the corresponding universal Kleisli $\mathcal{V}$-functor,

rω.1 $\mathcal{V}$ is a cocomplete $\boldsymbol{\omega}$**Cpo**-cartesian closed category[j];
rω.2 the unit of $\mathcal{T}$ is pointwise a full morphism (hence, J is a locally full $\boldsymbol{\omega}$**Cpo**-functor);
rω.3 $\mathcal{C}$ has an ep-zero object $\mathfrak{O} = J(\mathsf{0})$, where $\mathsf{0}$ is initial in $\mathcal{V}$;
rω.4 whenever $u : J(A) \overset{\hookrightarrow}{\leftarrow} J(B)$ is an ep-pair in $\mathcal{C}$, there is one morphism $\hat{u} : A \to B$ in $\mathcal{V}$ such that $J(\hat{u}) = u^e$.

An *rCBV* $\boldsymbol{\omega}$**Cpo**-*pair morphism* from $(\mathcal{V}, \mathcal{T})$ into $(\mathcal{V}', \mathcal{T}')$ is an $\boldsymbol{\omega}$**Cpo**-functor $H : \mathcal{V} \to \mathcal{V}'$ that strictly preserves $\boldsymbol{\omega}$**Cpo**-colimits, and whose underlying functor is a morphism between the *CBV* pairs. This defines a category of *rCBV* $\boldsymbol{\omega}$**Cpo**-pairs, denoted herein by $\boldsymbol{\omega}$**CPO**-$\mathfrak{C}_{r\mathcal{BV}}$.

[g]See [23, 4.2.2] or [43, Sect. 8] for the general setting of bilimit compact expansions.

[h]See, for instance, [21, Sect. 2] or [27, 3.10] for adjunctions in 2-categories.

[i]Recall that a *zero object* is an object that is both initial and terminal.

[j]$\mathcal{V}$ is, hence, $\boldsymbol{\omega}$**Cpo**-cocomplete as well.

Every *rCBV* $\boldsymbol{\omega}$**Cpo**-pair $(\mathcal{V}, \mathcal{T})$ has an underlying $\boldsymbol{\omega}$**Cpo**-pair, and this extends to a forgetful functor $\boldsymbol{\omega}\mathbf{CPO}\text{-}\mathfrak{C}_{r\mathcal{BV}} \to \boldsymbol{\omega}\mathbf{CPO}\text{-}\mathfrak{C}_{\mathcal{BV}}$. More importantly to our work, we have the following.

8.6.1. *rCBV* $\boldsymbol{\omega}$**Cpo**-*pairs are rCBV models.* Let $(\mathcal{V}, \mathcal{T})$ be an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair. It is clear that we have an underlying *CBV* pair which, by abuse of language, we denote by $(\mathcal{V}, \mathcal{T})$ as well. Hence, we can consider $(\mathcal{V}, \mathcal{T})$-parametric types.

Let $n \in \mathbb{N}^*$ and (8.4) be an n-variable $(\mathcal{V}, \mathcal{T})$-parametric type. For each $A \in (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n-1}$, we get an 1-variable $(\mathcal{V}, \mathcal{T})$-parametric type $E^A = \left(E^A_{\mathcal{V}}, E^A_{\mathcal{C}}\right)$ where $E^A_{\mathcal{V}}(W, Y) \overset{\text{def}}{=} E_{\mathcal{V}}(A, W, Y)$ and $E^A_{\mathcal{C}}(W', Y') \overset{\text{def}}{=} E_{\mathcal{C}}(J(A), W', Y')$. Let $\mathcal{E}^E_A$ be the diagram (8.20) in $\mathcal{C}$ given by the chain of morphisms

$$\left(a^e_n : \mathfrak{A}_n \to \mathfrak{A}_{n+1}\right)_{n \in \mathbb{N}},$$

where $(a_n)_{n\in\mathbb{N}}$ is the chain of ep-pairs inductively defined by (8.19).

$$\begin{aligned} a_0 &\overset{\text{def}}{=} \left(\iota^e : \mathfrak{D} \to E^A_{\mathcal{C}}(\mathfrak{D}, \mathfrak{D}), \iota^p : E^A_{\mathcal{C}}(\mathfrak{D}, \mathfrak{D}) \to \mathfrak{D}\right) \\ a_{n+1} &\overset{\text{def}}{=} \left(E^A_{\mathcal{C}}(a^p_n, a^e_n), E^A_{\mathcal{C}}(a^e_n, a^p_n)\right) \end{aligned} \qquad (8.19)$$

$$\begin{tikzcd} \mathfrak{D} \arrow[r, "a^e_0"] & \mathfrak{A}_1 \arrow[r, "a^e_1"] & \mathfrak{A}_2 \arrow[r, "a^e_2"] & \mathfrak{A}_3 \arrow[r, "a^e_3"] & \cdots \end{tikzcd} \qquad (8.20)$$

$$\begin{tikzcd} \mathfrak{D} & \mathfrak{A}_1 \arrow[l, "a^p_0"'] & \mathfrak{A}_2 \arrow[l, "a^p_1"'] & \mathfrak{A}_3 \arrow[l, "a^p_2"'] & \cdots \arrow[l, "a^p_3"'] \end{tikzcd} \qquad (8.21)$$

There is a unique diagram $\hat{\mathcal{E}}^E_A$ *such that* $J \circ \hat{\mathcal{E}}^E_A = \mathcal{E}^E_A$ by (rω.4) of Def. 8.8. Since $\mathcal{V}$ has $\boldsymbol{\omega}$**Cpo**-colimits, we conclude that the conical $\boldsymbol{\omega}$**Cpo**-colimit of $\hat{\mathcal{E}}^E_A$ exists and is preserved by J – hence, $\mathcal{E}^E_A$ has a conical $\boldsymbol{\omega}$**Cpo**-colimit in $\mathcal{C}$ as well.

By the celebrated *limit-colimit coincidence* [38], since (8.20) is the chain of embeddings of a chain of ep-pairs, the colimit gives us the $\boldsymbol{\omega}$**Cpo**-limit of the associated chain $(a^p_n)_{n\in\mathbb{N}}$ of projections (8.21), denoted herein by $\mathcal{P}^E_A$. This *bilimit* of ep-pairs is absolute – this means that any $\boldsymbol{\omega}$**Cpo**-functor $H : \mathcal{C} \to \mathcal{C}'$ preserves the conical $\boldsymbol{\omega}$**Cpo**-colimit (and $\boldsymbol{\omega}$**Cpo**-limit) of $\mathcal{E}^E_A$ (respectively, $\mathcal{P}^E_A$).

Since the conical $\boldsymbol{\omega}$**Cpo**-colimit of $\mathcal{E}^E_A$ is absolute, the diagram (8.4) commutes, and J strictly preserves $\boldsymbol{\omega}$**Cpo**-colimits, we have the invertible morphism (8.22) given by the composition of the respective canonical comparison morphisms.

$$\begin{tikzcd}
J \circ E_{\mathcal{V}}^{A}\left(\operatorname{colim}\left(\hat{\mathcal{E}_A^E}\right),\operatorname{colim}\left(\hat{\mathcal{E}_A^E}\right)\right) \arrow[r, "\cong"] & E_{\mathcal{C}}^{A}\left(\operatorname{colim}\left(\mathcal{E}_A^E\right),\operatorname{colim}\left(\mathcal{E}_A^E\right)\right) \arrow[d, "\cong"'] \\
 & \operatorname{colim}\left(\mathcal{E}_A^E\right) \arrow[d, "\cong"'] \\
 & J\operatorname{colim}\left(\hat{\mathcal{E}_A^E}\right)
\end{tikzcd} \tag{8.22}$$

It should be noted that, for each $f : (J^{\mathrm{op}} \times J)^{n-1}(A) \to (J^{\mathrm{op}} \times J)^{n-1}(B)$ in $(\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{n-1}$, we have an induced $\mathcal{V}$-natural transformation $\mathcal{E}_f^E : \mathcal{E}_A^E \to \mathcal{E}_B^E$. This association extends to a *$\mathcal{V}$-functor $\mathcal{E}^E$ from $(\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{n-1}$ into the $\mathcal{V}$-category of chains in $\mathcal{C}$*. The association $A \mapsto \hat{\mathcal{E}_A^E}$ also extends to a *$\mathcal{V}$-functor $\hat{\mathcal{E}^E}$ from $(\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n-1}$ into the $\mathcal{V}$-category of chains* by the $\mathcal{V}$-faithfulness of J, .

We define the fddt operator ν_ω as follows. For each $n \in \mathbb{N}^*$, given a $(\mathcal{V}, \mathcal{T})$-parametric type $E = (E_{\mathcal{V}}, E_{\mathcal{C}})$, we define:

$$\nu_\omega E = (\nu_\omega E_{\mathcal{V}}, \nu_\omega E_{\mathcal{C}}) \overset{\mathrm{def}}{=} \left(\operatorname{colim} \circ \hat{\mathcal{E}^E}, \operatorname{colim} \circ \mathcal{E}^E\right) \tag{8.23}$$

where, by abuse of language, colim is the $\mathcal{V}$-functor from the $\mathcal{V}$-category of chains in $\mathcal{V}$ (respectively, in $\mathcal{C}$) into the $\mathcal{V}$-category $\mathcal{V}$ (respectively, $\mathcal{C}$).

Since every isomorphism is an embedding, there is only one $\omega\mathsf{roll}_A^E$ *in $\mathcal{V}$ such that $J\left(\omega\mathsf{roll}_A^E\right)$ is equal to* (8.22). The morphisms $\omega\mathsf{roll}^E = \left(\omega\mathsf{roll}_A^E\right)_{A \in (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n-1}}$ gives a $\mathcal{V}$-natural transformation $E_{\mathcal{V}}(\mathrm{id}, \nu_\omega E_{\mathcal{V}}^{\mathrm{op}}, \nu_\omega E_{\mathcal{V}}) \to \nu_\omega E_{\mathcal{V}}$ such that $J\left(\omega\mathsf{roll}^E\right)$ is invertible. Therefore $\underline{\mathsf{roll}}_\omega \overset{\mathrm{def}}{=} \left(\omega\mathsf{roll}^E\right)_{E \in \mathsf{Param}(\mathcal{V}, \mathcal{T})}$ is a rolling for ν_ω and we can define the (free) type recursion $\underline{\nu}_\omega \overset{\mathrm{def}}{=} (\nu_\omega, \underline{\mathsf{roll}}_\omega)$.

Theorem 8.9 (Underlying $rCBV$ model)**.** *There is a forgetful functor $\mathcal{U}_{r\mathcal{BV}}$: $\boldsymbol{\omega}$**CPO**-$\mathfrak{C}_{r\mathcal{BV}} \to \mathfrak{C}_{\mathcal{RBV}}$ defined by $\mathcal{U}_{r\mathcal{BV}}(\mathcal{V}, \mathcal{T}) = (\mathcal{V}, \mathcal{T}, \underline{\nu}_\omega)$, that takes every morphism H to its underlying morphism of CBV models.*

Proof: From the definition of $\underline{\nu}_\omega$ and the fact that H strictly preserves $\mathcal{V}$-colimits, we conclude that, indeed, H respects the condition of $rCBV$ model morphism described in Def. 8.3. ■

Remark 8.10. The product of *rCBV* $\boldsymbol{\omega}$**Cpo**-pairs is given by $(\mathcal{V}_0, \mathcal{T}_0) \times (\mathcal{V}_1, \mathcal{T}_1) \cong (\mathcal{V}_0 \times \mathcal{V}_1, \mathcal{T}_0 \times \mathcal{T}_1)$. Moreover, it is clear that $\mathcal{U}_{r\mathcal{BV}}$ preserves finite products.

8.7. Concrete semantics. The *CBV* pair $(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp)$ as in 5.1 clearly satisfies the conditions of Def. 8.8 and, hence, it is also an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair. By Theorem 8.5, for each $k \in \mathbb{N} \cup \{\infty\}$, we have unique *rCBV* model morphisms (8.24) and (8.25) respecting the assignments of Fig. 5.9 and (5.11). In other words, following Remark 8.6, we have only one extension of the semantics (5.7) and (5.10) to the respective languages with recursive types.

$$\llbracket - \rrbracket : \quad \left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right) \ \to \mathcal{U}_{r\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \tag{8.24}$$

$$\llbracket - \rrbracket_k : \quad \left(\mathbf{Syn}^{\mathsf{Rtr}}_{V}, \mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}}, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right) \ \to \mathcal{U}_{r\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right). \tag{8.25}$$

Moreover, by Remark 8.10, we have that the product $(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp)$ as in 5.1 is an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair.

8.8. Subscone for *rCBV* $\boldsymbol{\omega}$Cpo-pairs. The first step for our logical relations proof is to verify that, for each $(n, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, the *CBV* $\boldsymbol{\omega}$**Cpo**-pair $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}(-)_\perp\right)$ as in Theorem 7.3 yields an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair. In order to do that, we rely on Theorem 8.12 about lifting the *rCBV* $\boldsymbol{\omega}$**Cpo**-pair structure.

Definition 8.11 (Impurity preserving/purity reflecting)**.** Let $(\mathcal{V}, \mathcal{T})$ and $(\mathcal{V}', \mathcal{T}')$ be *CBV* pairs. A *CBV* pair morphism $H : \mathcal{V} \to \mathcal{V}'$ is *impurity preserving* (or, *purity reflecting*) if, whenever $H(f) = \eta'_Y \circ g$, there is $\hat{f}$ in $\mathcal{V}$ such that $\eta_Y \circ \hat{f} = f$.

Theorem 8.12. *Let $(\mathcal{V}', \mathcal{T}')$ be an rCBV* $\boldsymbol{\omega}$**Cpo**-*pair, and $(\mathcal{V}, \mathcal{T})$ a CBV pair such that $\mathcal{V}$ is a cocomplete* $\boldsymbol{\omega}$**Cpo**-*cartesian closed category and $T(0)$ is terminal.*

If $H : \mathcal{V} \to \mathcal{V}'$ is a locally full $\boldsymbol{\omega}$**Cpo**-*functor that yields an impurity preserving CBV pair morphism $(\mathcal{V}, \mathcal{T}) \to \mathcal{U}_{\mathrm{rp}}(\mathcal{V}', \mathcal{T}')$, then $(\mathcal{V}, \mathcal{T})$ is an rCBV* $\boldsymbol{\omega}$**Cpo**-*pair. If, furthermore, H strictly preserves* $\boldsymbol{\omega}$**Cpo**-*colimits, then H yields an rCBV* $\boldsymbol{\omega}$**Cpo**-*pair morphism.*

Proof: We prove that $(\mathcal{V}, \mathcal{T})$ yields an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair. By hypothesis, $(\mathcal{V}, \mathcal{T})$ satisfies (rω.1). We prove the remaining conditions of Def. 8.8 below.

(rω.2) Let η and η' be respectively the unit of $\mathcal{T}$ and $\mathcal{T}'$. Since H is locally full, it reflects full morphisms. This implies that, for any $C \in \mathcal{V}$, η_C is full since $\eta'_{H(C)} = H(\eta_C)$ is full.

(rω.3) Since $T(0)$ is terminal, $J(0)$ is a zero object. Thus, for each $A \in \mathcal{C}$, we have the pair (8.26) of unique morphisms in $\mathcal{C}$.
Since $\overline{H}$ preserves initial objects and $(\mathcal{V}', \mathcal{T}')$ is an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair, we have that (8.27) is the ep-pair of the unique morphisms. Finally, since $\overline{H}$ is a locally full $\boldsymbol{\omega}$**Cpo**-functor, it reflects ep-pairs and, hence, (8.26) is an ep-pair.

$$\left(\iota_A : J(0) \to A, \iota^A : A \to J(0)\right) \qquad (8.26)$$

$$\left(\overline{H}(\iota_A), \overline{H}\left(\iota^A\right) : \overline{H}(A) \to \mathfrak{O}\right) \qquad (8.27)$$

(rω.4) Given an ep-pair $u : J(A) \overset{\hookrightarrow}{\leftarrow} J(B)$ in $\mathcal{C}$, the image $H(u) : \overline{H}J(A) \overset{\hookrightarrow}{\leftarrow} \overline{H}J(B)$ by H is an ep-pair. Since $(\mathcal{V}', \mathcal{T}')$ is an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair, there is one morphism $\hat{\overline{H}}(u) : H(A) \to H(B)$ in $\mathcal{V}'$ such that $J'\left(\hat{\overline{H}}(u)\right) = \overline{H}(u^e)$. Since the *CBV* pair morphism $H : (\mathcal{V}, \mathcal{T}) \to \mathcal{U}_{\mathrm{rp}}(\mathcal{V}', \mathcal{T}')$ is impurity preserving, we conclude that there is $\hat{u} : A \to B$ such that $J(\hat{u}) = u^e$.

■

As a consequence, in the setting of subscones satisfying Assumption 6.3, we get:

Theorem 8.13. *Let* $(\mathcal{V}, \mathcal{T})$ *be an rCBV* $\boldsymbol{\omega}$**Cpo***-pair, and* (8.28) *the forgetful* $\boldsymbol{\omega}$**Cpo***-functor coming from a pair* $(G : \mathcal{V} \to \mathcal{D}, \mathfrak{T}_{sub})$ *satisfying Assumption 6.3.*

If $\mathcal{D}$ *is cocomplete and* $\overline{\mathcal{T}} = \left(\overline{T}, \overline{\mathrm{m}}, \overline{\eta}\right)$ *is a strong monad that is a lifting of the monad* $\mathcal{T}$ *along* (8.28) *such that* ($\mathfrak{c}$.1) *and* ($\mathfrak{c}$.2) *hold, then* $\left(\mathbf{Sub}(\mathcal{D} \downarrow G), \overline{\mathcal{T}}\right)$ *is an rCBV* $\boldsymbol{\omega}$**Cpo***-pair and* $\underline{\mathfrak{L}}$ *yields an rCBV* $\boldsymbol{\omega}$**Cpo***-pair morphism* (8.29)*.*

$\mathfrak{c}$.*1* $\overline{T}$ *takes the initial to the terminal object;*
$\mathfrak{c}$.*2 for any* $(D, C, j) \in \mathbf{Sub}(\mathcal{D} \downarrow G)$*, denoting*

$$\overline{T}(D, C, j) = \left(\underline{\overline{\mathcal{T}}(D, C, j)}, T(D), \underline{\overline{\mathcal{T}}j}\right),$$

Diag (8.30) *induced by the unit* $\overline{\eta}$ *is a pullback in* $\mathcal{D}$*.*

$$\underline{\mathcal{L}} : \mathbf{Sub}\left(\mathcal{D} \downarrow G\right) \to \mathcal{V} \tag{8.28}$$

$$\left(\mathbf{Sub}\left(\mathcal{D} \downarrow G\right), \overline{\mathcal{T}}\right) \to \left(\mathcal{V}, \mathcal{T}\right) \tag{8.29}$$

$$\begin{tikzcd} D \arrow[r] \arrow[d, "j"'] & \underline{\overline{\mathcal{T}}\left(D, C, j\right)} \arrow[d, "\underline{\overline{\mathcal{T}}j}"] \\ G(C) \arrow[r, "G(\eta_C)"'] & G(T'(C)) \end{tikzcd} \tag{8.30}$$

Proof: By Corollary 6.5, $\mathbf{Sub}\left(\mathcal{D} \downarrow G\right)$ is cocomplete $\boldsymbol{\omega}\mathbf{Cpo}$-cartesian closed. Moreover, $\underline{\mathcal{L}}$ is locally full, strict $\boldsymbol{\omega}\mathbf{Cpo}$-cartesian closed, and $\boldsymbol{\omega}\mathbf{Cpo}$-colimit preserving by Theorem 6.6. Therefore, the fact that $\overline{\mathcal{T}}$ is a lifting of $\mathcal{T}$ through $\underline{\mathcal{L}}$ implies that it yields a *CBV* pair morphism (8.29).

($\mathfrak{c}$.2) implies that the *CBV* pair morphism (8.29) is purity reflecting. Assuming ($\mathfrak{c}$.1), this implies that $\left(\mathbf{Sub}\left(\mathcal{D} \downarrow G\right), \overline{\mathcal{T}}\right)$ is indeed an *rCBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair morphism and $\underline{\mathcal{L}}$ yields an (8.29) is an *rCBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair morphism by Theorem 8.12. ■

In the particular case of interest, we conclude:

Theorem 8.14. *For each* $(n, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$,

$$\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_{\perp}\right)$$

is an rCBV $\boldsymbol{\omega}\mathbf{Cpo}$*-pair. Moreover,*

$$\underline{\mathcal{L}}_{n,k} : \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right) \to \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}$$

yields an rCBV $\boldsymbol{\omega}\mathbf{Cpo}$*-pair morphism*

$$\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_{\perp}\right) \to \left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, \left(-\right)_{\perp}\right). \tag{8.31}$$

Proof: In fact, we already know that $\underline{\mathcal{L}}_{n,k}$ comes from a pair that satisfies Assumption 6.3. Moreover, $\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, \left(-\right)_{\perp}\right)$ is an *rCBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair and $\mathcal{P}_{n,k}\left(-\right)_{\perp}$ is a lifting of $\left(-\right)_{\perp}$ along $\underline{\mathcal{L}}_{n,k}$ satisfying the conditions of Theorem 8.13. ■

By Theorems 8.14 and 8.9, we get:

Corollary 8.15. $\underline{\mathcal{L}}_{n,k}$ *yields an rCBV model morphism*

$$\mathcal{U}_{r\mathcal{BV}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_{\perp}\right) \to \mathcal{U}_{r\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, \left(-\right)_{\perp}\right).$$

8.9. Logical relations as an *rCBV* model morphism. Let $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, and let's assume that $\mathcal{D}$ is sound for primitives (see 5.7). By the universal property of the $rCBV$ model $\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$ and the chain rule for derivatives, there is only one $rCBV$ model morphism

$$\overline{[\![-]\!]}_{n,k} : \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_{\perp}\right) \tag{8.32}$$

that is consistent with the assignment given by (7.8), (7.13), (7.15), and (7.14).

Lemma 8.16. *For any* $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$*, Diag.* (8.33) *commutes.*

$$\begin{tikzcd}
\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \arrow[r, "(\mathrm{id}, \mathbb{ID})"] \arrow[d, "\overline{[\![-]\!]}_{n,k}"'] & \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \times \left(\mathbf{Syn}_V^{\mathsf{Rtr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{Rtr}}, \underline{\nu}_{\mathbf{Syn}}^{\mathbf{tr}}\right) \arrow[d, "{[\![-]\!] \times [\![-]\!]_k}"] \\
\mathcal{U}_{r\mathcal{BV}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_{\perp}\right) \arrow[r, "\mathcal{U}_{r\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right)"'] & \mathcal{U}_{r\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, \left(-\right)_{\perp}\right)
\end{tikzcd} \tag{8.33}$$

Proof: Both $\left([\![-]\!] \times [\![-]\!]_k\right) \circ \left(\mathrm{id} \times \mathbb{ID}\right)$ and $\mathcal{U}_{r\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right) \circ \overline{[\![-]\!]}_{n,k}$ yield $rCBV$ model morphisms that are consistent with the assignment given by the object $\left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right)$ and the morphisms (7.9), (7.10) and (7.11). Therefore, by the universal property of $\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$, we conclude that Diag. (8.33) indeed commutes. ∎

8.10. AD correctness theorem for non-recursive data types. The correctness theorem for non-recursive data types follows from Lemma 8.16 and Corollary 7.7. That is to say, we have:

Theorem 8.17. *Let* $t : \coprod_{r \in \mathfrak{L}} \mathbf{real}^{s_r} \to \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}\left(\coprod_{j \in L} \mathbf{real}^{l_j}\right)$ *be a morphism in* $\mathbf{Syn}_V^{\mathsf{R}}$*. We have that* $[\![t]\!] : \coprod_{r \in \mathfrak{L}} \mathbb{R}^{s_r} \to \left(\coprod_{j \in L} \mathbb{R}^{l_j}\right)_{\perp}$ *is differentiable and, for any* $k \in (\mathbb{N} \cup \{\infty\})$, $[\![\mathbb{ID}\left(t\right)]\!]_k = \mathfrak{d}^k\left([\![t]\!]\right)$*.*

8.11. AD on recursive data types. The LR argument we presented provides us with an easy way to compute the logical relations of general recursive types: namely, since $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_{\perp}\right)$ is an $rCBV$ $\boldsymbol{\omega}\mathbf{Cpo}$-pair, the recursive types will be computed out of suitable colimits.

This gives us useful information about the semantics of $\mathcal{D}(t)$ for a program $x : \tau \vdash t : \sigma$ where τ and σ are recursive types. In particular, we can extend the correctness result 8.17 to any data type, including those involving recursion.

We denote by $\mathbf{Syn}^{\mathsf{R}}_{C}$ the Kleisli $\mathbf{Syn}^{\mathsf{R}}_{V}$-category associated with $\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}\right)$. Moreover, we respectively denote by (8.34) and (8.35) the coproduct, product and n-diagonal functors.

$$\sqcup, \times : \mathbf{Syn}^{\mathsf{R}}_{V} \times \mathbf{Syn}^{\mathsf{R}}_{V} \to \mathbf{Syn}^{\mathsf{R}}_{V} \tag{8.34}$$

$$\mathrm{diag}_n : \left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V} \to \left(\left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V}\right)^{n} \tag{8.35}$$

Definition 8.18. Let $R, I, O : \left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V} \to \mathbf{Syn}^{\mathsf{R}}_{V}$ be the constant functors which are, respectively, equal to **real**, $\mathsf{1}$ and $\mathsf{0}$. *We define the set* $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)$ *inductively by (D1), (D2) and (D3).*

(D1) The functors R, I, O are in $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)$. Moreover, the projection $\pi_2 : \left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V} \to \mathbf{Syn}^{\mathsf{R}}_{V}$ belongs to
$$\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right).$$

(D2) For each $n \in \mathbb{N}^*$, if the functors (8.36) belong to $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)$, then the functors (8.37) and (8.38) are in $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)$.

(D3) If $E = \left(E_{\mathbf{Syn}^{\mathsf{R}}_{V}}, E_{\mathbf{Syn}^{\mathsf{R}}_{C}}\right) \in \mathsf{Param}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}\right)$ is such that
$$E_{\mathbf{Syn}^{\mathsf{R}}_{V}} \in \mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right),$$
then $\left(\nu_{\mathbf{Syn}} E_{\mathbf{Syn}^{\mathsf{R}}_{V}}\right)$ is in $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)$.

We define the set $\mathsf{Param}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)$ *of parametric data types by* (8.39).

$$G, G' : \left(\left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V}\right)^{n} \to \mathbf{Syn}^{\mathsf{R}}_{V} \quad (8.36) \qquad G \circ \mathrm{diag}_n : \left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V} \to \mathbf{Syn}^{\mathsf{R}}_{V} \tag{8.37}$$

$$\times \circ \left(G \times G'\right), \sqcup \circ \left(G \times G'\right) : \left(\left(\mathbf{Syn}^{\mathsf{R}}_{V}\right)^{\mathrm{op}} \times \mathbf{Syn}^{\mathsf{R}}_{V}\right)^{2n} \to \mathbf{Syn}^{\mathsf{R}}_{V} \tag{8.38}$$

$$\left\{E \in \mathsf{Param}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}\right) : E_{\mathbf{Syn}^{\mathsf{R}}_{V}} \in \mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right)\right\} \tag{8.39}$$

Theorem 8.19. *Let E be an n-variable $\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$-parametric data type, where $n \in \mathbb{N}^*$. There is a countable family of natural numbers*

$$\left(\mathsf{m}_{(j,\mathtt{T})}\right)_{(j,\mathtt{T})\in(\mathbb{I}_n\cup\{0\})\times\mathsf{Tree}}$$

such that, for any rCBV model morphism

$$H : \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}\left(\mathcal{V}, \mathcal{T}\right)$$

and any H-compatible pair (E, F), we have that (8.41) holds, where the isomorphism $\cong$ is induced by coprojections and projections[k].

$$H\left(\tau\right) = \coprod_{j\in L} H\left(\mathbf{real}\right)^{l_j} \tag{8.40}$$

$$F_{\mathcal{V}}\left(W_j, Y_j\right)_{j\in\mathbb{I}_n} \cong \coprod_{\mathtt{T}\in\mathsf{Tree}} \left(H\left(\mathbf{real}\right)^{\mathsf{m}_{(0,\mathtt{T})}} \times \prod_{j=1}^{n} Y_j^{\mathsf{m}_{(j,\mathtt{T})}} \right) \tag{8.41}$$

As a consequence, if $\tau \in \mathbf{Syn}_V^{\mathsf{R}}$ corresponds to a data type τ, then there is a countable family $\left(l_j\right)_{j\in L} \in \mathbb{N}^L$ of natural numbers such that (8.40) holds for any rCBV model morphism $H : \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}\left(\mathcal{V}, \mathcal{T}\right)$.

Proof: The result follows from induction. The non-trivial part is a consequence of the following.

Let $\left(\tilde{E}, \tilde{F}\right) \in \mathsf{Param}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}\right) \times \mathsf{Param}(\mathcal{U}_{r\mathcal{BV}}\left(\mathcal{V}, \mathcal{T}\right))$ be an H-compatible pair of $(n+1)$-variable parametric types where $\tilde{F}_{\mathcal{V}}$ is given by (8.42) for some countable family $\left(\mathsf{s}_{(i,r)}\right)_{(i,r)\in(\mathbb{I}_{n+1}\cup\{0\})\times\mathfrak{L}}$ of natural numbers. We prove below that $\left(\nu_{\mathbf{Syn}}\tilde{E}, F\right)$ is H-compatible for some F such that $F_{\mathcal{V}}$ satisfies Eq. (8.41). By the definition $rCBV$ model morphism, we have that $\left(\nu_{\mathbf{Syn}}\tilde{E}, \nu_{\omega}\tilde{F}\right)$ is H-compatible. Hence, we only need to prove that $\nu_{\omega}\tilde{F}_{\mathcal{V}}$ is given by (8.41).

(I) We inductively define the set Tree by the following. Let $r \in \mathfrak{L}$: (a) if $\mathsf{s}_{(n+1,r)} = 0$, then $r \in \mathsf{Tree}$; (b) if $\mathsf{s}_{(n+1,r)} \neq 0$, then, for any $\mathtt{T} \in \mathsf{Tree}^{\mathsf{s}_{(n+1,r)}}$, the pair $(\mathtt{T}, r)$ is in Tree.

(II) We inductively define the family $\left(\mathsf{m}_{(j,\mathtt{T})}\right)_{(j,\mathtt{T})\in(\mathbb{I}_n\cup\{0\})\times\mathsf{Tree}}$ of indices by the following. Let $r \in \mathfrak{L}$: (a) if $\mathsf{s}_{(n+1,r)} = 0$, we define $\mathsf{m}_{(j,r)} := \mathsf{s}_{(j,r)}$ for each j; (b) if $\mathsf{s}_{(n+1,r)} \neq 0$, given $\mathtt{T} = (\mathtt{T}_i)_{i\in\mathbb{I}_{\mathsf{s}_{(n+1,r)}}} \in \mathsf{Tree}^{\mathsf{s}_{(n+1,r)}}$, we define $\mathsf{m}_{(j,(\mathtt{T},r))}$ by (8.43) for each j.

[k]That is to say, it is just a reorganization of the involved coproducts and products.

$$\tilde{F}_{\mathcal{V}}\left(W_i, Y_i\right)_{i\in\mathbb{I}_{n+1}} = \coprod_{r\in\mathfrak{L}} \left(H\left(\mathbf{real}\right)^{\mathsf{s}_{(0,r)}} \times \prod_{i=1}^{n+1} Y_i^{\mathsf{s}_{(i,r)}} \right) \quad (8.42) \qquad \mathsf{m}_{(j,(\mathtt{T},r))} = \mathsf{s}_{(j,r)} + \sum_{i=1}^{\mathsf{s}_{(n+1,r)}} \mathsf{m}_{(j,\mathtt{T}_i)} \quad (8.43)$$

Let $X = (W_i, Y_i)_{i\in\mathbb{I}_n} \in (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n$, $\mathfrak{F}_X := \tilde{F}_{\mathcal{V}}^X(\mathsf{0}, -)$ and ι the obvious unique morphism. The colimit of (8.44) is isomorphic to (8.45). Hence, by the definition of the *fddt* operator ν_ω of $\mathcal{U}_{r\mathcal{B}\mathcal{V}}(\mathcal{V}, \mathcal{T}) = (\mathcal{V}, \mathcal{T}, \underline{\nu_\omega})$, $\nu_\omega \tilde{F}_{\mathcal{V}}$ is given by the formula given in (8.41). This completes the proof.

$$\mathsf{0} \xrightarrow{\iota} \mathfrak{F}_X(\mathsf{0}) \xrightarrow{\mathfrak{F}_X(\iota)} \mathfrak{F}_X^2(\mathsf{0}) \xrightarrow{\mathfrak{F}_X^2(\iota)} \mathfrak{F}_X^3(\mathsf{0}) \longrightarrow \cdots \quad (8.44) \qquad \coprod_{\mathtt{T}\in\mathsf{Tree}} \left(H\left(\mathbf{real}\right)^{\mathsf{m}_{(0,\mathtt{T})}} \times \prod_{j=1}^{n} Y_j^{\mathsf{m}_{(j,\mathtt{T})}} \right) \quad (8.45)$$

Finally, if $\mathtt{\tau} \in \mathbf{Syn}_V^{\mathsf{R}}$ corresponds to a data type τ, then the constant parametric type $\underline{\mathtt{\tau}}$ equal to $\mathtt{\tau}$ is an $\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}\right)$-parametric data type of degree 1. Hence, denoting by $\underline{H\mathtt{\tau}}$ the constant parametric type equal to $H(\mathtt{\tau})$, since $(\underline{\mathtt{\tau}}, \underline{H\mathtt{\tau}})$ is H-compatible, we conclude that (8.41) holds for some $(l_j)_{j\in L}$ where L is countable. ■

$$\overline{[\![R]\!]}_{n,k} = \coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}_{n,k}^{l_j} \quad (8.46) \qquad [\![R]\!] = \coprod_{j\in L} \mathbb{R}^{l_j} \quad (8.47)$$

Theorem 8.20. *Let $t : \mathtt{\tau} \to \mathtt{\sigma}$ be a morphism in $\mathbf{Syn}_V^{\mathsf{R}}$. If $\mathtt{\tau}$ and $\mathtt{\sigma}$ correspond to data types, $[\![t]\!] : \coprod_{r\in\mathfrak{L}} \mathbb{R}^{s_r} \to \left(\coprod_{j\in L} \mathbb{R}^{l_j}\right)_{\perp}$ is differentiable and, for any $k \in (\mathbb{N} \cup \{\infty\})$, $[\![\mathbb{ID}(t)]\!]_k = \mathfrak{d}^k([\![t]\!])$.*

Proof: First of all, indeed, by Theorem 8.19, we have that there are countable families $(s_r)_{r\in\mathfrak{L}}$ and $(l_j)_{j\in L}$ such that

$$\overline{[\![t]\!]}_{s_i,k} : \coprod_{r\in\mathfrak{L}} \overline{[\![\mathbf{real}]\!]}_{s_i,k}^{s_r} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}_{s_i,k}^{l_j}\right)_{\perp} \tag{8.48}$$

is a morphism in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{s_i,k}\right)$, for each $i \in \mathfrak{L}$ and any $k \in \mathbb{N} \cup \{\infty\}$.

By the commutativity of (8.33) for any $(s_i, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, we get that the pair $([\![t]\!], [\![\mathbb{ID}(t)]\!]_k)$ defines the morphism (8.48) for each $i \in \mathfrak{L}$. By Corollary 7.7, this implies that $[\![t]\!]$ is differentiable and $[\![\mathbb{ID}(t)]\!]_k = \mathfrak{d}^k([\![t]\!])$. ■

Finally, as a consequence, we get:

Theorem 8.21. *Assume that* **vect** *implements the vector space* $\mathbb{R}^k$*, for some* $k \in \mathbb{N} \cup \{\infty\}$*. For any program* $x : \tau \vdash t : \sigma$ *where* τ, σ *are data types (including recursive data types), we have that* $[\![t]\!]$ *is differentiable and, moreover,*

$$[\![\mathcal{D}(t)]\!]_k = \mathfrak{d}^k\left([\![t]\!]\right) \tag{8.49}$$

provided that $\mathcal{D}$ *is sound for primitives.*

Following the considerations of 7.6 and 7.7, it follows from Theorem 8.17 that $\mathcal{D}$ as defined in 8.4 *correctly* provides us with forward and reverse AD transformations for data types.

8.12. AD on arrays. Arrays are semantically the same as lists: in our language, if τ is a data type, an array of τ is given by $\mu\alpha.\mathbf{1} \sqcup \tau \times \alpha$. It should be noted that, if $x : \mu\alpha.\mathbf{1} \sqcup \tau \times \alpha \vdash t : \mu\alpha.\mathbf{1} \sqcup \tau \times \beta$, we have that

$$[\![t]\!] : \coprod_{i=1}^{\infty} [\![\tau]\!] \to \left(\coprod_{i=1}^{\infty} [\![\sigma]\!]\right)_{\perp} .$$

By Theorem 8.21, if τ and σ are data types, we get that $\mathfrak{d}^k\left([\![t]\!]\right)$ (as defined in (5.6)) is equal to $[\![\mathcal{D}(t)]\!]_k$. Therefore, Theorem 8.21 already encompasses the correctness for arrays (of data types).

9. Related Work

This is an improved version of the unpublished preprint [43]. In particular, we have simplified the correctness argument to no longer depend on diffeological or sheaf-structure and to have it apply to arbitrary differentiable (rather than merely smooth) operations. We have further simplified the subsconing technique for recursive types.

There has recently been a flurry of work studying AD from a programming language point of view, a lot of it focussing on functional formulations of AD and their correctness. Examples of such papers are [32, 12, 35, 6, 1, 19, 28, 44, 26, 18, 45, 22, 37]. Of these papers, [32, 1, 28, 37] are particularly relevant as they also consider automatic differentiation of languages with partial features. Here, [32] considers an implementation that differentiates recursive programs and the implementation of [37] even differentiates code that uses recursive types. They do not give correctness proofs, however. Existing work on differential restriction categories [9] seems to give a more abstract semantic study of the interaction between forward-mode automatic differrentiation and partiality. We found that for our purposes, a concrete semantics in terms of ω-cpos sufficed, however.

The present paper can be seen as giving a correctness proof of the techniques implemented by [37]. [1] does give a denotational correctness proof of AD on a first-order functional language with (first-order) recursion. The first-orderness of the language allows the proof to proceed by plain induction rather than needing logical technique. [28] proves the correctness of basically the same AD algorithms that we consider in this paper when restricted to PCF with a base type of real numbers and a real conditional. Their proof relies on operational semantic techniques. Our contribution is to give an alternative denotational argument, which we believe is simple and systematic, and to extend it to apply to languages which, additionally, have the complex features of recursively defined datastructures that we find in realistic ML-family languages.

Such AD for languages with expressive features such as recursion and user-defined datatypes has been called for by the machine learning community [20, 46]. Previously, the subtlety of the interaction of automatic differentiation and real conditionals had first been observed by [3].

Our work gives a relatively simple denotational semantics for recursive types, which can be considered as an important special case of bilimit compact categories [23]. Bilimit compact categories are themselves, again, an important special case of the very general semantics of recursive types in terms of algebraically compact categories [15]. We believe that working with this special case of the semantics significantly simplifies our presentation.

In particular, this simplified semantics of recursive types allows us to give a very simple but powerful (open, semantic) logical technique for recursive types. It is an alternative to the two existing techniques for logical relations for recursive types: relational properties of domains [33], which is quite general but very technical to use, in our experience, and step-indexed logical relations [2], which are restricted to logical relations arguments about syntax, hence not applicable to our situation.

Finally, we hope that our work adds to the existing body of programming languages literature on automatic differentiation and recursion (and recursive types). In particular, we believe that it provides a simple, principled denotational explaination of how AD and expressive partial language features should interact. We plan to use it to generalise and prove correct the more advanced AD technique CHAD [44, 45, 26] when applied to languages with partial features.

References

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

[2] A. J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In P. Sestoft, editor, *Programming Languages and Systems, 15th European Symposium on Programming, ESOP 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings*, volume 3924 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2006.

[3] T. Beck and H. Fischer. The if-problem in automatic differentiation. *Journal of Computational and Applied Mathematics*, 50(1-3):119–131, 1994.

[4] M. Betancourt. Double-pareto lognormal distribution in stan, Aug 2019.

[5] S. Bloom and Z. Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993.

[6] A. Brunel, D. Mazza, and M. Pagani. Backpropagation in the simply typed lambda-calculus with linear negation. In *Proc. POPL 2020*, 2020.

[7] B. Carpenter, M. Hoffman, M. Brubaker, D. Lee, P. Li, and M. Betancourt. The Stan math library: Reverse-mode automatic differentiation in C++. *arXiv preprint arXiv:1509.07164*, 2015.

[8] M. Clementino and F. Lucatelli Nunes. Lax comma 2-categories and admissible 2-functors. *arXiv e-prints*, page arXiv:2002.03132, February 2020.

[9] J. R. B. Cockett, G. S. Cruttwell, and J. D. Gallagher. Differential restriction categories. *arXiv preprint arXiv:1208.4068*, 2012.

[10] E. Dubuc. Adjoint triangles. In *Reports of the Midwest Category Seminar, II*, pages 69–91. Springer, Berlin, 1968.

[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. *Proceedings of the ACM on Programming Languages*, 2(ICFP):70, 2018.

[13] 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.

[14] S. Flaxman, S. Mishra, A. Gandy, H. J. T. Unwin, T. A. Mellan, H. Coupland, C. Whittaker, H. Zhu, T. Berah, J. W. Eaton, et al. Estimating the effects of non-pharmaceutical interventions on covid-19 in europe. *Nature*, 584(7820):257–261, 2020.

[15] P. Freyd. Algebraically complete categories. In *Category Theory*, pages 95–104. Springer, 1991.

[16] S. Goncharov, C. Rauch, and L. Schröder. Unguarded recursion on coinductive resumptions. *Electronic Notes in Theoretical Computer Science*, 319:183–198, 2015.

[17] B. Goodrich. Conway-maxwell-poisson distribution, Oct 2017.

[18] M. Huot, S. Staton, and M. Vákár. Higher order automatic differentiation of higher order functions. *CoRR*, abs/2101.06757, 2021.

[19] M. Huot, S. Staton, and M. Vákár. Correctness of automatic differentiation via diffeologies and categorical gluing. Full version, 2020. arxiv:2001.02209.

[20] E. Jeong, J. S. Jeong, S. Kim, G.-I. Yu, and B.-G. Chun. Improving the expressiveness of deep learning frameworks with recursion. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–13, 2018.

[21] G. M. Kelly and R. Street. Review of the elements of 2-categories. In *Category Seminar (Proc. Sem., Sydney, 1972/1973)*, Lecture Notes in Math., Vol. 420, pages 75–103. Springer, Berlin, 1974.

[22] F. Krawiec, S. P. Jones, N. Krishnaswami, T. Ellis, R. A. Eisenberg, and A. W. Fitzgibbon. Provably correct, asymptotically efficient, higher-order reverse-mode automatic differentiation. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022.

[23] P. Levy. *Call-by-push-value: A Functional/imperative Synthesis*, volume 2. Springer Science & Business Media, 2012.

[24] P. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Information and computation*, 185(2):182–210, 2003.

[25] F. Lucatelli Nunes. On biadjoint triangles. *Theory Appl. Categ.*, 31:Paper No. 9, 217–256, 2016.

[26] F. Lucatelli Nunes and M. Vákár. CHAD for Expressive Total Languages. *arXiv e-prints*, page arXiv:2110.00446, October 2021.

[27] F. Lucatelli Nunes. Semantic Factorization and Descent. *arXiv e-prints*, page arXiv:1902.01225, February 2019.

[28] D. Mazza and M. Pagani. Automatic differentiation in PCF. *Proc. ACM Program. Lang.*, 5(POPL):1–27, 2021.

[29] E. Meijer. Behind every great deep learning framework is an even greater programming languages concept (keynote). In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1–1, 2018.

[30] 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.

[31] E. Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

[32] B. Pearlmutter and J. Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7, 2008.

[33] A. Pitts. Relational properties of domains. *Inform. Comput.*, 127(2):66–90, 1996.

[34] G. Plotkin. Some principles of differential programming languages. *Invited talk, POPL*, 2018, 2018.

[35] A. Shaikhha, A. Fitzgibbon, D. Vytiniotis, and S. Peyton Jones. Efficient differentiable programming in a functional array-processing language. *Proceedings of the ACM on Programming Languages*, 3(ICFP):97, 2019.

[36] S. Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[37] T. Smeding and M. Vákár. Efficient Dual-Numbers Reverse AD via Well-Known Program Transformations. *arXiv e-prints*, page arXiv:2207.03418, July 2022.

[38] M. Smyth and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, 1982.

[39] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[40] R. Street. The formal theory of monads. *J. Pure Appl. Algebra*, 2(2):149–168, 1972.

[41] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[42] P. Tsiros, F. Y. Bois, A. Dokoumetzidis, G. Tsiliki, and H. Sarimveis. Population pharmacokinetic reanalysis of a diazepam pbpk model: a comparison of stan and gnu mcsim. *Journal of Pharmacokinetics and Pharmacodynamics*, 46(2):173–192, 2019.

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

[44] M. Vákár. Reverse AD at higher types: pure, principled and denotationally correct. In *Programming languages and systems. 30th European symposium on programming, ESOP 2021, held as part of the European joint conferences on theory and practice of software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021. Proceedings*, pages 607–634. Cham: Springer, 2021.

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

[46] B. van Merrienboer, O. Breuleux, A. Bergeron, and P. Lamblin. Automatic differentiation in ml: Where we are and where we should be going. In *Advances in neural information processing systems*, pages 8757–8767, 2018.

[47] X. Zhang, L. Lu, and M. Lapata. Top-down tree long short-term memory networks. In *Proceedings of NAACL-HLT*, pages 310–320, 2016.

Appendix A. Fine grain call-by-value and AD

In §4, we have discussed a standard coarse-grain CBV language, also known as the λ_C-calculus, computational λ-calculus [30], or, plainly, CBV. In this appendix, we discuss an alternative presentation in terms of fine-grain CBV [24, 23] (also known as Moggi's monadic metalanguage [31]). While it is slightly more verbose, this presentation clarifies the precise universal property that is satisfied by the syntax of our language.

A.1. Fine grain call-by-value. We consider a standard fine-grain call-by-value language (with complex values) over a ground type **real** of real numbers, real constants $\underline{c} \in \mathrm{Op}_0$ for $c \in \mathbb{R}$, and certain basic operations $\mathrm{op} \in \mathrm{Op}_n$ for each natural number $n \in \mathbb{N}$.

The types τ, σ, ρ, (complex) values v, w, u, and computations t, s, r of our language are as follows.

$$\begin{array}{lllll}
\tau,\sigma,\rho & ::= & \text{types} & \mid\ \mathbf{1} \mid \tau_1 \times \tau_2 & \text{products} \\
 & \mid\ \mathbf{real} & \quad\text{numbers} & \mid\ \tau \to \sigma & \text{function} \\
 & \mid\ \mathbf{0} \mid \tau + \sigma & \quad\text{sums} & & \\
 & & & \mid\ \mathbf{case}\,v\,\mathbf{of}\,\{ \begin{array}{c}\mathbf{inl}\,x \to w\\ \mid \mathbf{inr}\,y \to u\end{array} \} & \text{sum match} \\
v,y,u & ::= & \text{values} & & \\
 & \mid\ x,y,z & \quad\text{variables} & \mid\ \langle\,\rangle \mid \langle v,w\rangle & \text{tuples} \\
 & \mid\ \underline{c} & \quad\text{constant} & \mid\ \mathbf{case}\,v\,\mathbf{of}\,\langle x,y\rangle \to w & \text{product match} \\
 & \mid\ \mathbf{case}\,v\,\mathbf{of}\,\{\,\} & \quad\text{sum match} & \mid\ \lambda x.t & \text{abstractions} \\
 & \mid\ \mathbf{inl}\,v \mid \mathbf{inr}\,v & \quad\text{inclusions} & \mid\ \mu x.v & \text{term recursion} \\
 & & & & \\
t,s,r & ::= & \text{computations} & \mid\ \mathbf{case}\,v\,\mathbf{of}\,\{ \begin{array}{c}\mathbf{inl}\,x \to t\\ \mid \mathbf{inr}\,y \to s\end{array} \} & \text{sum match} \\
 & \mid\ t\,\mathbf{to}\,x.\,s & \quad\text{sequencing} & & \\
 & \mid\ \mathbf{return}\,v & \quad\text{pure comp.} & \mid\ \mathbf{case}\,v\,\mathbf{of}\,\langle x,y\rangle \to t & \text{product match} \\
 & \mid\ \mathrm{op}(v_1,\ldots,v_n) & \quad\text{operation} & \mid\ v\ w & \text{function app.} \\
 & \mid\ \mathbf{case}\,v\,\mathbf{of}\,\{\,\} & \quad\text{sum match} & \mid\ \mathbf{iterate}\,t\,\mathbf{from}\,x = v & \text{iteration} \\
 & & & \mid\ \mathbf{sign}\,v & \text{sign function}
\end{array}$$

We will use sugar

$$\mathbf{if}\,v\,\mathbf{then}\,t\,\mathbf{else}\,s \overset{\text{def}}{=} \mathbf{sign}\,(v)\,\mathbf{to}\,x.\,\mathbf{case}\,x\,\mathbf{of}\,\{\_ \to s \;\big|\; \_ \to r\}$$

$$\mathbf{fst}\,v \overset{\text{def}}{=} \mathbf{case}\,v\,\mathbf{of}\,\langle x, \_\rangle \to x$$

$$\mathbf{snd}\,v \overset{\text{def}}{=} \mathbf{case}\,v\,\mathbf{of}\,\langle \_, x\rangle \to x$$

$$\mathbf{let\,rec}\,f(x) = t\,\mathbf{in}\,s \overset{\text{def}}{=} (\mu f.\mathbf{return}\,(\lambda x.t))\,\mathbf{to}\,f.\,s.$$

We could also define iteration as syntactic sugar:

$$\mathbf{iterate}\,t\,\mathbf{from}\,x = v \overset{\text{def}}{=} (\mu z.\lambda x.t\,\mathbf{to}\,y.\,\mathbf{case}\,y\,\mathbf{of}\,\{\mathbf{inl}\,x' \to z\,x' \mid \mathbf{inr}\,x'' \to \mathbf{return}\,x''\})\ v.$$

The typing rules are in Figure A.1.

$$\dfrac{}{\Gamma \vdash^v x : \tau}((x:\tau)\in\Gamma) \quad \dfrac{\Gamma \vdash^c t : \tau \quad \Gamma, x:\tau \vdash^c s : \sigma}{\Gamma \vdash^c t\,\mathbf{to}\,x.\,s : \sigma} \quad \dfrac{\Gamma \vdash^v v : \tau}{\Gamma \vdash^c \mathbf{return}\,v : \tau} \quad \dfrac{}{\Gamma \vdash^v \underline{c} : \mathbf{real}}(c\in\mathbb{R})$$

$$\dfrac{\Gamma \vdash^v v_1 : \mathbf{real} \quad \cdots \quad \Gamma \vdash^v v_n : \mathbf{real}}{\Gamma \vdash^c \mathrm{op}(v_1,\ldots,v_n) : \mathbf{real}}(\mathrm{op}\in\mathrm{Op}_n) \quad \dfrac{\Gamma \vdash^v v : \mathbf{0}}{\Gamma \vdash^v \mathbf{case}\,v\,\mathbf{of}\,\{\,\} : \tau} \quad \dfrac{\Gamma \vdash^v v : \mathbf{0}}{\Gamma \vdash^c \mathbf{case}\,v\,\mathbf{of}\,\{\,\} : \tau}$$

$$\dfrac{\Gamma \vdash^v v : \tau}{\Gamma \vdash^v \mathbf{inl}\,v : \tau \sqcup \sigma} \quad \dfrac{\Gamma \vdash^v v : \sigma}{\Gamma \vdash^v \mathbf{inr}\,v : \tau \sqcup \sigma} \quad \dfrac{\Gamma \vdash^v v : \sigma \sqcup \rho \quad \Gamma, x:\sigma \vdash^v w : \tau \quad \Gamma, y:\rho \vdash^v u : \tau}{\Gamma \vdash^v \mathbf{case}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x \to w \mid \mathbf{inr}\,y \to u\} : \tau}$$

$$\dfrac{\Gamma \vdash^v v : \sigma \sqcup \rho \quad \Gamma, x:\sigma \vdash^c t : \tau \quad \Gamma, y:\rho \vdash^c s : \tau}{\Gamma \vdash^c \mathbf{case}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x \to t \mid \mathbf{inr}\,y \to s\} : \tau} \quad \dfrac{}{\Gamma \vdash^v \langle\,\rangle : \mathbf{1}} \quad \dfrac{\Gamma \vdash^v v : \tau \quad \Gamma \vdash^v w : \sigma}{\Gamma \vdash^v \langle v, w\rangle : \tau \times \sigma}$$

$$\dfrac{\Gamma \vdash^v v : \sigma \times \rho \quad \Gamma, x:\sigma, y:\rho \vdash^v w : \tau}{\Gamma \vdash^v \mathbf{case}\,v\,\mathbf{of}\,\langle x, y\rangle \to w : \tau} \quad \dfrac{\Gamma \vdash^v v : \sigma \times \rho \quad \Gamma, x:\sigma, y:\rho \vdash^c t : \tau}{\Gamma \vdash^c \mathbf{case}\,v\,\mathbf{of}\,\langle x, y\rangle \to t : \tau} \quad \dfrac{\Gamma, x:\sigma \vdash^c t : \tau}{\Gamma \vdash^v \lambda x.t : \sigma \to \tau}$$

$$\dfrac{\Gamma \vdash^v v : \sigma \to \tau \quad \Gamma \vdash^v w : \sigma}{\Gamma \vdash^c v\,w : \tau} \quad \dfrac{\Gamma, x:\sigma \vdash^c t : \sigma \sqcup \tau \quad \Gamma \vdash^v v : \sigma}{\Gamma \vdash^c \mathbf{iterate}\,t\,\mathbf{from}\,x = v : \tau}$$

$$\dfrac{\Gamma, x:\tau \vdash^v v : \tau}{\Gamma \vdash^v \mu x.v : \tau}(\tau = \sigma \to \rho) \quad \dfrac{\Gamma \vdash^v v : \mathbf{real}}{\Gamma \vdash^c \mathbf{sign}\,v : \mathbf{1} \sqcup \mathbf{1}}$$

FIGURE A.1. Typing rules for the our fine-grain CBV language with iteration and real conditionals. We use a typing judgement $\vdash^v$ for values and $\vdash^c$ for computations.

A.2. Equational theory. We consider our language up to the usual $\beta\eta$-equational theory for fine-grain CBV, which is displayed in Fig. A.2.

$$\begin{array}{ll}
\mathbf{return}\,v\,\mathbf{to}\,x.\,t = t[{}^{v}\!/_{x}] & (t\,\mathbf{to}\,x.\,s)\,\mathbf{to}\,y.\,r = t\,\mathbf{to}\,x.\,(s\,\mathbf{to}\,y.\,r) \\
\mathbf{case}\,\mathbf{inl}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x \to w \mid \mathbf{inr}\,y \to u\} = w[{}^{v}\!/_{x}] & w[{}^{v}\!/_{z}] \overset{\#x,y}{=} \mathbf{case}\,v\,\mathbf{of}\,\{ \begin{array}{c} \mathbf{inl}\,x \to w[{}^{\mathbf{inl}\,x}\!/_{z}] \\ \mid \mathbf{inr}\,y \to w[{}^{\mathbf{inr}\,y}\!/_{z}] \end{array} \} \\
\mathbf{case}\,\mathbf{inr}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x \to w \mid \mathbf{inr}\,y \to u\} = u[{}^{v}\!/_{y}] & \\
\mathbf{case}\,\langle v, w\rangle\,\mathbf{of}\,\langle x, y\rangle \to u = u[{}^{v}\!/_{x}, {}^{w}\!/_{y}] & u[{}^{v}\!/_{z}] \overset{\#x,y}{=} \mathbf{case}\,v\,\mathbf{of}\,\langle x, y\rangle \to u[{}^{\langle x,y\rangle}\!/_{z}] \\
\mathbf{case}\,\mathbf{inl}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x \to t \mid \mathbf{inr}\,y \to s\} = t[{}^{v}\!/_{x}] & t[{}^{v}\!/_{z}] \overset{\#x,y}{=} \mathbf{case}\,v\,\mathbf{of}\,\{ \begin{array}{c} \mathbf{inl}\,x \to t[{}^{\mathbf{inl}\,x}\!/_{z}] \\ \mid \mathbf{inr}\,y \to t[{}^{\mathbf{inr}\,y}\!/_{z}] \end{array} \} \\
\mathbf{case}\,\mathbf{inr}\,v\,\mathbf{of}\,\{\mathbf{inl}\,x \to t \mid \mathbf{inr}\,y \to s\} = s[{}^{v}\!/_{y}] & \\
\mathbf{case}\,\langle v, w\rangle\,\mathbf{of}\,\langle x, y\rangle \to t = t[{}^{v}\!/_{x}, {}^{w}\!/_{y}] & t[{}^{v}\!/_{z}] \overset{\#x,y}{=} \mathbf{case}\,v\,\mathbf{of}\,\langle x, y\rangle \to t[{}^{\langle x,y\rangle}\!/_{z}] \\
(\lambda x.t)\ v = t[{}^{v}\!/_{x}] & v \overset{\#x}{=} \lambda x.v\,x
\end{array}$$

FIGURE A.2. Standard $\beta\eta$-laws for fine-grain CBV. 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 terms of the same type.

Under the translation of coarse-grain CBV into fine-grain CBV, this equational theory induces precisely that of Section 4.

A.3. The syntactic *CBV* model. Our fine grain call-by-value language corresponds with a *CBV* model (see Def. 2.4).

We define the category $\mathbf{Syn}_V$ *of values*, which has types as objects. $\mathbf{Syn}_V(\tau,\sigma)$ consists of $(\alpha)\beta\eta$-equivalence classes of values $x:\tau\vdash^v v:\sigma$, where identities are $x:\tau\vdash^v x:\sigma$ and composition of $x:\tau\vdash^v v:\sigma$ and $y:\sigma\vdash^v w:\rho$ is given by $x:\tau\vdash^v w[{}^v/_y]:\rho$.

Lemma A.1. $\mathbf{Syn}_V$ *is bicartesian closed.*

Similarly, *we define the category* $\mathbf{Syn}_C$ *of computations*, which also has types as objects. $\mathbf{Syn}_C(\tau,\sigma)$ consists of $(\alpha)\beta\eta$-equivalence classes of computations $x:\tau\vdash^c t:\sigma$, where identities are $x:\tau\vdash^c \mathbf{return}\, x:\sigma$ and composition of $x:\tau\vdash^c t:\sigma$ and $y:\sigma\vdash^c s:\rho$ is given by $x:\tau\vdash^c t\,\mathbf{to}\, y.\, s:\rho$.

Lemma A.2. $\mathbf{Syn}_C$ *is a* $\mathbf{Syn}_V$*-category.*

We define the $\mathbf{Syn}_V$-functors

$$\begin{array}{rll} \mathbf{Syn}_G: & \mathbf{Syn}_C & \hookrightarrow \mathbf{Syn}_V \\ & \tau & \mapsto (\mathbf{1}\to\tau) \\ & t & \mapsto \lambda\langle\,\rangle.t \end{array} \qquad\qquad \begin{array}{rll} \mathbf{Syn}_J: & \mathbf{Syn}_V & \hookrightarrow \mathbf{Syn}_C \\ & \tau & \mapsto \tau \\ & v & \mapsto \mathbf{return}\, v. \end{array}$$

We have that $\mathbf{Syn}_J\dashv\mathbf{Syn}_G$ is a (Kleisli) $\mathbf{Syn}_V$-adjunction $\mathbf{Syn}_J\dashv\mathbf{Syn}_G$ and, hence, denoting by $\mathbf{Syn}_{\mathcal{S}}$ the induced $\mathbf{Syn}_V$-monad, $(\mathbf{Syn}_V,\mathbf{Syn}_{\mathcal{S}})$ is a *CBV* pair, as defined in 2.1. Moreover, considering the free recursion and free iteration

$$\begin{array}{rll} \mathbf{Syn}_{\mathsf{it}}: & (x:\sigma\vdash^c t:\sigma\sqcup\tau) & \mapsto \lambda y.(\mathbf{iterate}\, t\,\mathbf{from}\, x=y) \\ \mathbf{Syn}_{\mu}: & (x:\tau\vdash^v v:\tau) & \mapsto \mu x.v \quad (\tau=\sigma\to\rho), \end{array}$$

we get the *CBV* model $\big(\mathbf{Syn}_V,\mathbf{Syn}_{\mathcal{S}},\mathbf{Syn}_{\mu},\mathbf{Syn}_{\mathsf{it}}\big)$ which has the following universal property.

Theorem A.3 (Universal Property of the Syntax)**.** *Let* $(\mathcal{V},\mathcal{T},\mu,)$ *be a CBV model with chosen finite products, coproducts and exponentials. For each consistent assignment*

$$H(\mathbf{real}) \in \mathsf{ob}\,\mathcal{V} \tag{A.3}$$

$$H(\underline{c}) \in \mathcal{V}(1,H(\mathbf{real})) \tag{A.4}$$

$$H(\mathrm{op}) \in \mathcal{C}(H(\mathbf{real})^n,H(\mathbf{real}))=\mathcal{V}(H(\mathbf{real})^n,TH(\mathbf{real}))\,,\ \textit{for each}\ \mathrm{op}\in\mathrm{Op}_n \tag{A.5}$$

$$H(\mathbf{sign}\,) \in \mathcal{C}(H(\mathbf{real}),1\sqcup 1)=\mathcal{V}(H(\mathbf{real}),T(1\sqcup 1)) \tag{A.6}$$

there is a unique CBV model morphism H between $\big(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}\big)$ *and* $(\mathcal{V}, \mathcal{T}, \mu,)$ *respecting it.*

Theorem A.4 (Universal Property of the Syntax). *Let* $(\mathcal{V}, \mathcal{T}, \mu,)$ *be a CBV model with chosen finite products, coproducts and exponentials. For each consistent assignment*

$$
\begin{aligned}
H(\mathbf{real}) &\in \mathsf{ob}\,\mathcal{V} && \text{(A.7)}\\
H(\underline{c}) &\in \mathcal{V}(1, H(\mathbf{real})) && \text{(A.8)}\\
H(\mathrm{op}) &\in \mathcal{C}(H(\mathbf{real})^n, H(\mathbf{real})) = \mathcal{V}(H(\mathbf{real})^n, TH(\mathbf{real}))\,,\ \textit{for each}\ \mathrm{op} \in \mathrm{Op}_n && \text{(A.9)}\\
H(\mathbf{sign}\,) &\in \mathcal{C}(H(\mathbf{real}), 1 \sqcup 1) = \mathcal{V}(H(\mathbf{real}), T(1 \sqcup 1)) && \text{(A.10)}
\end{aligned}
$$

there is a unique CBV model morphism H between $\big(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}\big)$ *and* $(\mathcal{V}, \mathcal{T}, \mu,)$ *respecting it.*

A.4. A translation from coarse-grain CBV to fine-grain CBV.

This translation $(-)^\dagger$ operates on types and contexts as the identity. It faithfully translates terms $\Gamma \vdash t : \tau$ of coarse-grain CBV into computations $\Gamma \vdash^c t^\dagger : \tau$ of fine-grain CBV. This translation illustrates the main difference between coarse-grain and fine-grain CBV: in coarse-grain CBV, values are subset of computations, while fine-grain CBV is more explicit in keeping values and computations separate. This makes it slightly cleaner to formulate an equational theory, denotational semantics, and logical relations arguments.

We list the translation $(-)^\dagger$ below where all newly introduced variables are chosen to be fresh.

coarse-grain CBV computation t	fine-grain CBV translation $t^\dagger$
x	$\mathbf{return}\,x$
$\mathbf{let}\,x =\ t\,\mathbf{in}\,s$	$t^\dagger\,\mathbf{to}\,x.\,s^\dagger$
$\underline{c}$	$\mathbf{return}\,\underline{c}$
$\mathbf{inl}\,t$	$t^\dagger\,\mathbf{to}\,x.\,\mathbf{return}\,\mathbf{inl}\,x$
$\mathbf{inr}\,t$	$t^\dagger\,\mathbf{to}\,x.\,\mathbf{return}\,\mathbf{inr}\,x$
$\langle\,\rangle$	$\mathbf{return}\,\langle\,\rangle$
$\langle t, s\rangle$	$t^\dagger\,\mathbf{to}\,x.\,s^\dagger\,\mathbf{to}\,y.\,\mathbf{return}\,\langle x, y\rangle$
$\lambda x.t$	$\mathbf{return}\,\lambda x.t^\dagger$
$\mathrm{op}(t_1, \ldots, t_n)$	$t_1^\dagger\,\mathbf{to}\,x_1.\,\ldots t_n^\dagger\,\mathbf{to}\,x_n.\,\mathrm{op}(x_1, \ldots, x_n)$
$\mathbf{case}\,t\,\mathbf{of}\,\{\,\}$	$t^\dagger\,\mathbf{to}\,x.\,\mathbf{case}\,x\,\mathbf{of}\,\{\,\}$
$\mathbf{case}\,t\,\mathbf{of}\,\{\mathbf{inl}\,x \to s \mid \mathbf{inr}\,y \to r\}$	$t^\dagger\,\mathbf{to}\,z.\,\mathbf{case}\,z\,\mathbf{of}\,\{\mathbf{inl}\,x \to s^\dagger \mid \mathbf{inr}\,y \to r^\dagger\}$
$\mathbf{case}\,t\,\mathbf{of}\,\langle x, y\rangle \to s$	$t^\dagger\,\mathbf{to}\,z.\,\mathbf{case}\,z\,\mathbf{of}\,\langle x, y\rangle \to s^\dagger$
$t\,s$	$t^\dagger\,\mathbf{to}\,x.\,s^\dagger\,\mathbf{to}\,y.\,x\,y$
$\mathbf{iterate}\,t\,\mathbf{from}\,x = s$	$s^\dagger\,\mathbf{to}\,y.\,\mathbf{iterate}\,t^\dagger\,\mathbf{from}\,x = y$
$\mathbf{sign}\ t$	$t^\dagger\,\mathbf{to}\,x.\,\mathbf{sign}\ x$
$\mu z.t$	$\mu z.\lambda x.t^\dagger\,\mathbf{to}\,y.\,yx$

A.5. Dual numbers forward AD transformation. As before, we fix, for all $n \in \mathbb{N}$, for all $\mathrm{op} \in \mathrm{Op}_n$, for all $1 \leq i \leq n$, computations $x_1 : \mathbf{real}, \ldots, x_n : \mathbf{real} \vdash^c \partial_i \mathrm{op}(x_1, \ldots, x_n) : \mathbf{real}$, which represent the partial derivatives of op. Using these terms for representing partial derivatives, we define, in Fig. A.11, a structure preserving macro $\mathcal{D}$ on the types, values, and computations of our language for performing forward-mode AD. We observe that this induces the following AD rule for our sugar: $\mathcal{D}_{\mathcal{C}}(\mathbf{if}\, v\, \mathbf{then}\, t\, \mathbf{else}\, s\,) = \mathbf{case}\, \mathcal{D}_{\mathcal{V}}(v)\, \mathbf{of}\, \langle x, \_\rangle \to \mathbf{if}\, x\, \mathbf{then}\, \mathcal{D}_{\mathcal{C}}(t)\, \mathbf{else}\, \mathcal{D}_{\mathcal{C}}(s)$. In fact, by the universal property of $\mathbf{Syn}_J$, $\mathcal{D}$ is the unique structure preserving functor on $\mathcal{D}$ that has the right definition for constants, primitive operations and **sign**. It automatically follows that $\mathcal{D}$ respects $\beta\eta$-equality.

Under the translation of coarse-grain CBV into fine-grain CBV, this code transformation induces precisely that of §4.

Appendix B. A more efficient derivative for sign

We can define by mutual induction (for both $\mathcal{D} = \mathcal{D}, \overleftarrow{\mathcal{D}}_k$)

$$\begin{aligned}
x : \mathcal{D}(\tau) &\vdash \mathbf{p}_\tau(x) : \tau \\
x : \mathcal{D}(\mathbf{real}) &\vdash \mathbf{fst}\,(x) : \mathbf{real} \\
x : \mathcal{D}(\tau) \times \mathcal{D}(\sigma) &\vdash \langle \mathbf{p}_\tau(\mathbf{fst}\, x), \mathbf{p}_\sigma(\mathbf{snd}\, x)\rangle : \tau \times \sigma \\
x : \mathcal{D}(\tau) \sqcup \mathcal{D}(\sigma) &\vdash \mathbf{case}\, x\, \mathbf{of}\, \{\mathbf{inl}\, y \to \mathbf{inl}\, \mathbf{p}_\tau(y) \mid \mathbf{inr}\, z \to \mathbf{inr}\, \mathbf{p}_\sigma(z)\} : \tau \sqcup \sigma \\
x : \mathcal{D}(\tau) \to \mathcal{D}(\sigma) &\vdash \lambda y.\mathbf{p}_\sigma(x(\mathbf{z}_\tau(y))) : \tau \to \sigma \\
x : \mu\alpha.\mathcal{D}(\tau) &\vdash \mathbf{case}\, x\, \mathbf{of}\, \mathbf{roll}\, y \to \mathbf{roll}\, \mathbf{p}_\tau(x) : \mu\alpha.\tau \\
x : \alpha &\vdash x : \alpha
\end{aligned}$$

and

$$\begin{aligned}
x : \tau &\vdash \mathbf{z}_\tau(x) : \mathcal{D}(\tau) \\
x : \mathbf{real} &\vdash \langle x, \underline{0}\rangle : \mathcal{D}(\mathbf{real}) \\
x : \tau \times \sigma &\vdash \langle \mathbf{z}_\tau(\mathbf{fst}\, x), \mathbf{z}_\sigma(\mathbf{snd}\, x)\rangle : \mathcal{D}(\tau) \times \mathcal{D}(\sigma) \\
x : \tau \sqcup \sigma &\vdash \mathbf{case}\, x\, \mathbf{of}\, \{\mathbf{inl}\, y \to \mathbf{inl}\, \mathbf{z}_\tau(y) \mid \mathbf{inr}\, z \to \mathbf{inr}\, \mathbf{z}_\sigma(z)\} : \mathcal{D}(\tau) \sqcup \mathcal{D}(\sigma) \\
x : \tau \to \sigma &\vdash \lambda y.\mathbf{z}_\sigma(x(\mathbf{p}_\tau(y))) : \mathcal{D}(\tau) \to \mathcal{D}(\sigma) \\
x : \mu\alpha.\tau &\vdash \mathbf{case}\, x\, \mathbf{of}\, \mathbf{roll}\, y \to \mathbf{roll}\, \mathbf{z}_\tau(x) : \mu\alpha.\mathcal{D}(\tau) \\
x : \alpha &\vdash x : \alpha.
\end{aligned}$$

Then, observe that, for any $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash t : \mathbf{real}$, we have $[\![\mathbf{sign}\,(\mathbf{fst}\, \mathcal{D}(t))]\!] = [\![\mathbf{let}\, x_1 = \mathbf{p}_{\tau_1}(x_1)\, \mathbf{in} \cdots \mathbf{let}\, x_n = \mathbf{p}_{\tau_n}(x_n)\, \mathbf{in} \cdots \mathbf{sign}\, t]\!]$. Therefore, we can define, for $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash t : \mathbf{real}$,

$$\mathcal{D}(\mathbf{sign}\, t) \stackrel{\text{def}}{=} \mathbf{let}\, x_1 = \mathbf{p}_{\tau_1}(x_1)\, \mathbf{in} \cdots \mathbf{let}\, x_n = \mathbf{p}_{\tau_n}(x_n)\, \mathbf{in} \cdots \mathbf{sign}\, t.$$

This yields more efficient definitions of the forward and reverse derivatives of **sign** and **if then else** as we do not need to differentiate t at all.

$$\begin{array}{lll}
\mathcal{D}(\mathbf{real}) \stackrel{\text{def}}{=} \mathbf{real}\times\mathbf{vect} & \mathcal{D}(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0} & \mathcal{D}(\tau\sqcup\sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau)\sqcup\mathcal{D}(\sigma) \\
\mathcal{D}(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1} & \mathcal{D}(\tau\to\sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau)\to\mathcal{D}(\sigma) & \mathcal{D}(\tau\times\sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau)\times\mathcal{D}(\sigma)
\end{array}$$

$$\begin{array}{l}
\mathcal{D}_{\mathcal{V}}(x) \stackrel{\text{def}}{=} x \\
\mathcal{D}_{\mathcal{V}}(\mathbf{case}\,v\,\mathbf{of}\,\{\,\}) \stackrel{\text{def}}{=} \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v)\,\mathbf{of}\,\{\,\} \\
\mathcal{D}_{\mathcal{V}}(\mathbf{inl}\,v) \stackrel{\text{def}}{=} \mathbf{inl}\,\mathcal{D}_{\mathcal{V}}(v) \\
\mathcal{D}_{\mathcal{V}}(\mathbf{inr}\,v) \stackrel{\text{def}}{=} \mathbf{inr}\,\mathcal{D}_{\mathcal{V}}(v) \\
\mathcal{D}_{\mathcal{V}}(\mathbf{case}\,v\,\mathbf{of}\,\{\begin{array}{l}\mathbf{inl}\,x\to w\\ \mid \mathbf{inr}\,y\to u\end{array}\}) \stackrel{\text{def}}{=} \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v)\,\mathbf{of}\,\{\begin{array}{l}\mathbf{inl}\,x\to \mathcal{D}_{\mathcal{V}}(w)\\ \mid \mathbf{inr}\,y\to \mathcal{D}_{\mathcal{V}}(u)\end{array}\} \\
\mathcal{D}_{\mathcal{V}}(\langle\,\rangle) \stackrel{\text{def}}{=} \langle\,\rangle \\
\mathcal{D}_{\mathcal{V}}(\langle v, w\rangle) \stackrel{\text{def}}{=} \langle \mathcal{D}_{\mathcal{V}}(v), \mathcal{D}_{\mathcal{V}}(w)\rangle \\
\mathcal{D}_{\mathcal{V}}(\mathbf{case}\,v\,\mathbf{of}\,\langle x,y\rangle\to u) \stackrel{\text{def}}{=} \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v)\,\mathbf{of}\,\langle x,y\rangle\to \mathcal{D}_{\mathcal{V}}(u) \\
\mathcal{D}_{\mathcal{V}}(\lambda x.t) \stackrel{\text{def}}{=} \lambda x.\mathcal{D}_{\mathcal{C}}(t) \\
\mathcal{D}_{\mathcal{C}}(t\,\mathbf{to}\,x.\,s) \stackrel{\text{def}}{=} \mathcal{D}_{\mathcal{C}}(t)\,\mathbf{to}\,x.\,\mathcal{D}_{\mathcal{C}}(s) \\
\mathcal{D}_{\mathcal{C}}(\mathbf{return}\,v) \stackrel{\text{def}}{=} \mathbf{return}\,\mathcal{D}_{\mathcal{V}}(v) \\
\mathcal{D}_{\mathcal{C}}(\mathbf{case}\,v\,\mathbf{of}\,\{\,\}) \stackrel{\text{def}}{=} \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v)\,\mathbf{of}\,\{\,\} \\
\mathcal{D}_{\mathcal{C}}(\mathbf{case}\,v\,\mathbf{of}\,\{\begin{array}{l}\mathbf{inl}\,x\to t\\ \mid \mathbf{inr}\,y\to s\end{array}\}) \stackrel{\text{def}}{=} \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v)\,\mathbf{of}\,\{\begin{array}{l}\mathbf{inl}\,x\to \mathcal{D}_{\mathcal{C}}(t)\\ \mid \mathbf{inr}\,y\to \mathcal{D}_{\mathcal{C}}(s)\end{array}\} \\
\mathcal{D}_{\mathcal{C}}(\mathbf{case}\,v\,\mathbf{of}\,\langle x,y\rangle\to t) \stackrel{\text{def}}{=} \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v)\,\mathbf{of}\,\langle x,y\rangle\to \mathcal{D}_{\mathcal{C}}(t) \\
\mathcal{D}_{\mathcal{C}}(v\,w) \stackrel{\text{def}}{=} \mathcal{D}_{\mathcal{V}}(v)\,\mathcal{D}_{\mathcal{V}}(w) \\
\mathcal{D}_{\mathcal{C}}(\mathbf{iterate}\,t\,\mathbf{from}\,x=v) \stackrel{\text{def}}{=} \mathbf{iterate}\,\mathcal{D}_{\mathcal{C}}(t)\,\mathbf{from}\,x=\mathcal{D}_{\mathcal{V}}(v) \\
\mathcal{D}_{\mathcal{C}}(\mu x.t) \stackrel{\text{def}}{=} \mu x.\mathcal{D}_{\mathcal{C}}(t)
\end{array}$$

$$\begin{array}{ll}
\mathcal{D}_{\mathcal{V}}(\underline{c}) \stackrel{\text{def}}{=} & \langle \underline{c}, \underline{0}\rangle \\
\mathcal{D}_{\mathcal{C}}(\mathrm{op}(v_1,\ldots,v_n)) \stackrel{\text{def}}{=} & \mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v_1)\,\mathbf{of}\,\langle x_1,x_1'\rangle\to\ldots\to\mathbf{case}\,\mathcal{D}_{\mathcal{V}}(v_n)\,\mathbf{of}\,\langle x_n,x_n'\rangle\to \\
 & \mathrm{op}(x_1,\ldots,x_n)\,\mathbf{to}\,y. \\
 & \partial_1\mathrm{op}(x_1,\ldots,x_n)\,\mathbf{to}\,z_1.\ \ldots\partial_n\mathrm{op}(x_1,\ldots,x_n)\,\mathbf{to}\,z_n. \\
 & \mathbf{return}\,\langle y, x_1' * z_1 + \ldots + x_n' * z_n\rangle \\
\mathcal{D}_{\mathcal{C}}(\mathbf{sign}\,v) \stackrel{\text{def}}{=} & \mathbf{sign}\,(\mathbf{fst}\,\mathcal{D}_{\mathcal{V}}(v))
\end{array}$$

FIGURE A.11. A forward-mode AD macro defined on types as $\mathcal{D}(-)$, values as $\mathcal{D}_{\mathcal{V}}(-)$, and computations as $\mathcal{D}_{\mathcal{C}}(-)$. All newly introduced variables are chosen to be fresh.

Appendix C. Enriched scone

We present straightforward generalizations (enriched versions) of the results presented in [26, Section 9] below.

Considering the $\boldsymbol{\omega}\mathbf{Cpo}$-category 2 with two objects and only one non-trivial morphism between them, the $\boldsymbol{\omega}\mathbf{Cpo}$-category $2 \pitchfork \mathcal{D}$ of morphisms of $\mathcal{D}$ can be described as the $\boldsymbol{\omega}\mathbf{Cpo}$-category $\boldsymbol{\omega}\mathbf{Cpo}\text{-}\mathsf{Cat}[2,\mathcal{D}]$ of $\boldsymbol{\omega}\mathbf{Cpo}$-functors $2 \to \mathcal{D}$.

Explicitly, the objects of $2 \pitchfork \mathcal{D}$ are morphisms $f : Y_0 \to Y_1$ of $\mathcal{D}$. A morphism between f and g is a pair $\alpha = (\alpha_0, \alpha_1) : f \to g$ such that $\alpha_1 f = g\alpha_0$, that is to say, a ($\boldsymbol{\omega}\mathbf{Cpo}$-)natural transformation. Finally, the $\boldsymbol{\omega}\mathbf{Cpo}$-structure is defined by $(\alpha_0, \alpha_1) \le (\beta_0, \beta_1)$ if $\alpha_0 \le \beta_0$ and $\alpha_1 \le \beta_1$ in $\mathcal{D}$.

Given an $\boldsymbol{\omega}\mathbf{Cpo}$-functor $G : \mathcal{C} \to \mathcal{D}$, the comma category $\mathcal{D} \downarrow G$ of the identity on $\mathcal{D}$ along G in $\boldsymbol{\omega}\mathbf{Cpo}\text{-}\mathsf{Cat}$ is also known as the $\boldsymbol{\omega}\mathbf{Cpo}$-*scone* or *Artin glueing* of G. It can be described as the pullback (C.1) in $\boldsymbol{\omega}\mathbf{Cpo}\text{-}\mathsf{Cat}$, in which $\mathrm{codom} : 2 \pitchfork \mathcal{D} \to \mathcal{D}$, defined by $(\alpha = (\alpha_0, \alpha_1) : f \to g) \mapsto \alpha_1$, is the codomain $\boldsymbol{\omega}\mathbf{Cpo}$-functor.

$$\begin{tikzcd}
\mathcal{D} \downarrow G \arrow[r, "\mathsf{proj}_{2\pitchfork\mathcal{D}}"] \arrow[d, "\mathsf{proj}_{\mathcal{C}}"'] & 2 \pitchfork \mathcal{D} \arrow[d, "\mathrm{codom}"] \\
\mathcal{C} \arrow[r, "G"'] & \mathcal{D}
\end{tikzcd} \tag{C.1}$$

Since codom is an isofibration, the pullback (C.1) is equivalent to the pseudo-pullback of codom along G, which is the $\boldsymbol{\omega}\mathbf{Cpo}$-category defined as follows. The objects of the pseudo-pullback are triples

$$\Big((f : Y_0 \to Y_1) \in 2 \pitchfork \mathcal{D}, C \in \mathcal{C}, \xi : (\mathrm{codom} f) \overset{\cong}{\to} G(C)\Big)$$

where ξ is an isomorphism in $\mathcal{D}$. A morphism $(f, C, \xi) \to (f', C', \xi')$ is a pair of morphisms $(\alpha : f \to f', h : C \to C')$ such that $G(h) \circ \xi = \xi' \circ \mathrm{codom}\,(\alpha)$. Finally, the $\boldsymbol{\omega}\mathbf{Cpo}$-structure of the homs are given pointwise. That is to say, $(\alpha, h) \le (\alpha', h')$ if $\alpha \le \alpha$ in $2 \pitchfork \mathcal{D}$ and $h \le h'$ in $\mathcal{C}$.

Lemma C.1. *The forgetful $\boldsymbol{\omega}\mathbf{Cpo}$-functor $\mathcal{L} : \mathcal{D} \downarrow G \to \mathcal{D} \times \mathcal{C}$, defined in* (6.3)*, creates all absolute (weighted) limits and colimits.*

Proof: Clearly, the $\boldsymbol{\omega}\mathbf{Cpo}$-functor $\mathcal{L}$ reflects isomorphisms.

Let D be a diagram in $\mathcal{D} \downarrow G$ such that the weighted (co)limit $(co)\mathsf{lim}\,(W, \mathcal{L}D)$ exists and is preserved by any $\boldsymbol{\omega}\mathbf{Cpo}$-functor. Since $\mathcal{D} \downarrow G$ is the pullback (C.1), there is a unique pair of diagrams (D_0, D_1) such that

$$\mathsf{proj}_{2 \pitchfork \mathcal{D}} \circ D = D_0, \quad \mathsf{proj}_{\mathcal{C}} \circ D = D_1, \quad \mathrm{codom} \circ D_0 = G \circ D_1,$$

hold.

Since $\mathrm{dom} \circ D_0 = \pi_{\mathcal{D}} \circ \mathcal{L} \circ D$ and $\mathrm{codom} \circ D_0 = G \circ \pi_{\mathcal{C}} \circ \mathcal{L} \circ D$, we get that $(co)\mathsf{lim}\,(W, \mathrm{dom} D_0) \cong \pi_{\mathcal{D}}\left((co)\mathsf{lim}\,(W, \mathcal{L} \circ D)\right)$ and $(co)\mathsf{lim}\,(W, \mathrm{codom} \circ D_0) \cong G \circ \pi_{\mathcal{C}}\left((co)\mathsf{lim}\,(W, \mathcal{L} \circ D)\right)$. Therefore, $(co)\mathsf{lim}\,(W, \mathcal{L} \circ D_0)$ exists in $2 \pitchfork \mathcal{D}$, pointwise constructed out of $(co)\mathsf{lim}\,(W, \mathrm{dom} \circ D_0)$ and $(co)\mathsf{lim}\,(W, \mathrm{codom} \circ D_0)$.

Moreover, since $D_1 = \pi_{\mathcal{C}} \circ \mathcal{L} \circ D$, we have that

$$(co)\mathsf{lim}\,(W, D_1) \cong \pi_{\mathcal{C}}\left((co)\mathsf{lim}\,(W, \mathcal{L} \circ D)\right).$$

Therefore, the isomorphism ξ given by

$$\begin{aligned}\mathrm{codom}\left((co)\mathsf{lim}\,(W, D_0)\right) &\cong (co)\mathsf{lim}\,(W, \mathrm{codom} \circ D_0)\\ &\cong G \circ \pi_{\mathcal{C}}\left((co)\mathsf{lim}\,(W, \mathcal{L} \circ D)\right)\\ &\cong G\left((co)\mathsf{lim}\,(W, D_1)\right)\end{aligned}$$

together with the pair $\left((co)\mathsf{lim}\,(W, D_0), (co)\mathsf{lim}\,(W, D_1)\right)$ defines, up to isomorphism, an object of $\mathcal{D} \downarrow G$, which satisfies the universal property for $(co)\mathsf{lim}\,(W, D) = (co)\mathsf{lim}\,(W, (D_0, D_1))$.

Moreover, by the construction above, we conclude that $(co)\mathsf{lim}\,(W, D)$ is preserved by $\mathcal{L}$. In particular:

$$\mathcal{L}\left((co)\mathsf{lim}\,(W, D_0), (co)\mathsf{lim}\,(W, D_1), \xi\right) = \left((co)\mathsf{lim}\,(W, \mathrm{dom} \circ D_0), (co)\mathsf{lim}\,(W, D_1)\right).$$

The above completes the proof that the $\boldsymbol{\omega}\mathbf{Cpo}$-functor $\mathcal{L}$ creates $(co)\mathsf{lim}\,(W, D)$. ■

The $\boldsymbol{\omega}\mathbf{Cpo}$-functor $\mathcal{L}$ has a right $\boldsymbol{\omega}\mathbf{Cpo}$-adjoint provided that $\mathcal{D}$ has binary $\boldsymbol{\omega}\mathbf{Cpo}$-products. It is given by $(D \in \mathcal{D}, C \in \mathcal{C}) \mapsto \left(D \times G(C), C, \pi_{G(C)}\right)$. Therefore:

Theorem C.2. *The forgetful $\boldsymbol{\omega}\mathbf{Cpo}$-functor $\mathcal{L} : \mathcal{D} \downarrow G \to \mathcal{D} \times \mathcal{C}$ is $\boldsymbol{\omega}\mathbf{Cpo}$-comonadic provided that $\mathcal{D}$ has binary $\boldsymbol{\omega}\mathbf{Cpo}$-products.*

By duality, we get that the forgetful $\boldsymbol{\omega}\mathbf{Cpo}$-functor $F \downarrow \mathcal{C} \to \mathcal{D} \times \mathcal{C}$ is $\boldsymbol{\omega}\mathbf{Cpo}$-monadic provided that $\mathcal{C}$ has finite $\boldsymbol{\omega}\mathbf{Cpo}$-coproducts. Therefore:

Theorem C.3. *The forgetful $\boldsymbol{\omega}\mathbf{Cpo}$-functor $\mathcal{L} : \mathcal{D} \downarrow G \to \mathcal{D} \times \mathcal{C}$ is $\boldsymbol{\omega}\mathbf{Cpo}$-monadic whenever G has a left $\boldsymbol{\omega}\mathbf{Cpo}$-adjoint and $\mathcal{C}$ has finite $\boldsymbol{\omega}\mathbf{Cpo}$-coproducts.*

Proof: Indeed, by the $\boldsymbol{\omega}$**Cpo**-adjunction $F \dashv G$, we get an isomorphism $\mathcal{D} \downarrow G \cong F \downarrow \mathcal{C}$ which composed with the forgetful $\boldsymbol{\omega}$**Cpo**-functor $F \downarrow \mathcal{C} \to \mathcal{D} \times \mathcal{C}$ is equal to $\mathcal{L} : \mathcal{D} \downarrow G \to \mathcal{D} \times \mathcal{C}$. ■

As a consequence, we conclude that:

Theorem C.4. *Let $G : \mathcal{C} \to \mathcal{D}$ be a right $\boldsymbol{\omega}$**Cpo**-adjoint functor between $\boldsymbol{\omega}$**Cpo**-bicartesian closed categories. We have that the forgetful $\boldsymbol{\omega}$**Cpo**-functor $\mathcal{L}$ is $\boldsymbol{\omega}$**Cpo**-monadic and comonadic. In particular, $\mathcal{D} \downarrow G$ is an $\boldsymbol{\omega}$**Cpo**-bicartesian closed category.*

Appendix D. Haskell Code: Recursive Neural Network

```
-- example implementation of https://icml.cc/2011/papers/125_icmlpaper.pdf
-- Some of the basic datatypes we use -- we elide the implementation of some
data Tree a
  = Leaf a
  | Node (Tree a) (Tree a)
  deriving (Eq) -- \mu b. a + (b x b), leaf a = roll (iota_1 a), node l r = roll (iota_2 (l, r))

data Vector

data Scalar

data Matrix

type ActivationVectors = [Vector]

type AdjacencyMatrix = [(Tree Int, Tree Int)]

-- Some basic data and operations that we need for the RNN
-- Again, we elide much of the implementation as it is beside the point of this example
f :: Vector -> Vector -- some non-linear function, usually elementwise applied sigmoid function
f = undefined

conc :: Vector -> Vector -> Vector -- concatenate vectors
conc = undefined

mult :: Matrix -> Vector -> Vector -- matrix vector multiplication
mult = undefined

add :: Vector -> Vector -> Vector -- elementwise vector addition
add = undefined

innerprod :: Vector -> Vector -> Scalar -- vector inner product
innerprod = undefined

a :: ActivationVectors
a = undefined -- input (for example, sequence of words as vectors or image segments as vectors)

adjMat :: AdjacencyMatrix
-- start with matrix that only stores (Leaf i, Leaf j) pairs in case i is a neighbour of j;
-- we later extend adjacency to parent nodes
adjMat = undefined -- input (specify which words/image segments are neighbours )

w :: Matrix
w = undefined -- parameter to learn: weights

b :: Vector
b = undefined -- parameter to learn: biases

wScore :: Vector
wScore = undefined -- parameter to learn: scoring vector

-- The implementation of the RNN
-- version 1, without caching
modelH ((w, b, wScore), (adjMat, globalScore)) =
  let getNode (Leaf i) = a !! i
   in let getNode (Node l r) = f (w `mult` conc (getNode l) (getNode r)) `add` b
       in let parentsScores =
                map
                  (\i -> (i, innerprod wScore (getNode (uncurry Node i))))
                  adjMat -- compute scores for all parent nodes of neighbours;
                  -- super inefficient without caching getNode, but conceptually cleaner
           in let ((bp1, bp2), bestScore) =
                    foldl
                      (\(i, s) (i', s') ->
                         if s > s'
                           then (i, s)
                           else (i', s'))
                      (head parentsScores)
                      parentsScores -- find the neighbours that have the higest score
               in let globalScore2 = globalScore + bestScore
               -- add the local contribution of our chosen neighbour pair to the global score
                   in let bestPar = Node bp1 bp2
                   -- actually compute our favourite parent;
                   -- I guess we'd already done this before but it's cheap to redo
                       in let mergeParH i
                                | i == bp1 || i == bp2 = bestPar
                           in let mergeParH i
```

```
                                       | otherwise = i
                                 in let mergePar (i, j) =
                                           (mergeParH i, mergeParH j)
                                     in let adjMat2 =
                                               filter
                                                 (/= (bestPar, bestPar))
                                                 [ mergePar (i, j)
                                                 | (i, j) <- adjMat
                                                 ]
                                                 -- replace bp1 and bp2 with bestPar in adjacencyMatrix,
                                                 -- but we have a convention that nodes are not neighbours
                                                 -- of themselves
                                          in if null adjMat2
                                               then Right globalScore2
                                               else Left (adjMat, globalScore2)
                                               -- if we run out of neighbours that can be merged, we are done;
                                               -- otherwise iterate with new adjacency matrix and score

it :: ((c, a) -> Either a b) -> (c, a) -> b -- functional iteration
it f (c, a) =
  case f (c, a) of
    Left a' -> it f (c, a')
    Right b -> b

model :: ((Matrix, Vector, Vector), (AdjacencyMatrix, Scalar)) -> Scalar
model = it modelH

-- The implementation of the RNN
-- version2, with caching of getNode
modelH2 ((w, b, wScore), (adjMat, values, globalScore)) =
  let getNode (Leaf i) = look (Leaf i) values
   in let getNode (Node l r) =
             let lv = look l values
              in let rv = look r values
                  in f (w `mult` conc lv rv) `add` b
       in let parentsValScores =
                 map
                   (\i ->
                      let v = getNode (uncurry Node i)
                       in (i, v, innerprod wScore v))
                   adjMat
            in let ((bp1, bp2), bestVal, bestScore) =
                     foldl
                        (\(i, v, s) (i', v', s') ->
                           if s > s'
                           then (i, v, s)
                           else (i', v', s'))
                      (head parentsValScores)
                      parentsValScores
               in let globalScore2 = globalScore + bestScore
                   in let bestPar = Node bp1 bp2
                       in let mergeParH i
                                | i == bp1 || i == bp2 = bestPar
                           in let mergeParH i
                                    | otherwise = i
                               in let mergePar (i, j) =
                                        (mergeParH i, mergeParH j)
                                   in let adjMat2 =
                                            filter
                                              (/= (bestPar, bestPar))
                                              [ mergePar (i, j)
                                              | (i, j) <- adjMat
                                              ]
                                       in if null adjMat2
                                            then Right globalScore2
                                            else Left
                                                   ( adjMat
                                                   , (bestPar, bestVal) : values
                                                   , globalScore2)

-- initial values will be zip (map Leaf [0..], a)
look :: Tree Int -> [(Tree Int, b)] -> b -- a map operation for looking up cache
look k m =
  case lookup k m of
    Just x -> x

model2 :: ((Matrix, Vector, Vector), (AdjacencyMatrix, Scalar)) -> Scalar
model2 ((w, b, wScore), (adjMat, globalScore)) =
  it modelH2 ((w, b, wScore), (adjMat, zip (map Leaf [0 ..]) a, globalScore))
```

Fernando Lucatelli Nunes
Departement Informatica, Universiteit Utrecht, Nederland & University of Coimbra, CMUC, Department of Mathematics, Portugal
E-mail address: f.lucatellinunes@uu.nl

Matthijs Vákár
Departement Informatica, Universiteit Utrecht, Nederland
E-mail address: m.i.l.vakar@uu.nl