Mathematical Structures in Computer Science (2024), 1–60
doi:10.1017/S0960129524000215

CAMBRIDGE UNIVERSITY PRESS

PAPER

Automatic differentiation for ML-family languages: Correctness via logical relations

Fernando Lucatelli Nunes[1,2] and Matthijs Vákár[1]

[1]Department of Information and Computing Sciences, Utrecht University, Utrecht, the Netherlands and [2]Department of Mathematics, University of Coimbra, CMUC, Coimbra, Portugal

Corresponding author: Fernando Lucatelli Nunes; Email: fernandolucatellinunes@gmail.com

(Received 19 June 2023; revised 21 July 2024; accepted 27 July 2024)

Abstract

We give a simple, direct, and reusable logical relations technique for languages with term and type recursion and partially defined differentiable functions. We demonstrate it by working out the case of automatic differentiation (AD) correctness: namely, we present a correctness proof of a dual numbers style AD code transformation for realistic functional languages in the ML-family. We also show how this code transformation provides us with correct forward- and reverse-mode AD.

The starting point is to interpret a functional programming language as a suitable freely generated categorical structure. In this setting, by the universal property of the syntactic categorical structure, the dual numbers AD code transformation 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: Programming language semantics; Differentiable programming languages; Recursive types; Logical relations; Program transformations; Software correctness

1. Introduction

Logical relations. Logical relations arguments (see, e.g., Mitchell and Scedrov 1992 for a survey) are proof techniques that can be used to demonstrate properties of typed programming languages (PL), ranging from strong normalization to canonicity and adequacy. The arguments are essentially type-guided forms of induction. They seem to have been reinvented several times by different research communities and are also known under various other names, including Tait's method of computability, reducibility candidates, Artin gluing, Sierpinski cone, (sub)sconing, and Freyd cover.

Category theory gives a useful way to organize logical relations arguments: by viewing them as ways of building a new categorical semantics of a PL out of an existing ones. The new semantics then equips objects with predicates of some form and restricts the morphisms to those morphisms that respect the predicates. By choosing the right notion of predicates, we can ensure that the existence of this new semantics gives us the property we are hoping to prove about our PL.

In this paper, we present novel logical relations methods for languages with recursive features, together with an application of these techniques to correctness proofs for automatic differentiation (AD).

AD and the PL community. Automatic differentiation (AD, see, e.g., Griewank and Walther 2008 for a survey) is a popular family of techniques 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 PL community has turned toward studying AD from a new perspective. Much progress has been made toward 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 that 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 AD in a semantically unified way, making use only of the very simple concrete denotational model of ω-cpos. In doing so, we simplify existing techniques that relied on sheaf-theoretic machinery (Vákár 2020; Huot et al. 2023).

A key challenge that we tackle to achieve the correctness proofs of this paper is to have sufficiently strong categorical logical relations techniques for reasoning about partially defined differentiable functions and term and type recursion. We believe that our novel methods can be simpler than existing alternatives such as Pitts (1996) and Ahmed (2006), and they are still widely applicable, *our second major contribution*.

We refer to the companion paper (Smeding and Vákár 2022) 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 toward point (4). Parallelism preservation (point (5)) for this AD technique is discussed in Smeding and Vákár (2024).

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 (Vákár 2021; Vákár and Smeding 2022; Lucatelli Nunes and Vákár 2023), 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 (Meijer 2018; Plotkin 2018): compilers for general-purpose PL should provide built-in support for AD of any programs written in the language. Such general-purpose PL 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, a basic example being the ReLU function:

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

which is a key component of many neural networks. Conditionals are also frequently used in probabilistic programming to paste together density functions of different distributions

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

(Betancourt 2019). 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 AD* (Beck and Fischer 1994). Our solution is the one also employed by Abadi and Plotkin (2020): 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 Huot et al. (2020)'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. In Section 11, we briefly discuss how the more ambitious solution to the if-problem in the style of Lee et al. (2020), Mazza and Pagani (2021), and Huot et al. (2023) can also be achieved with our methods. In that solution, we show that the set of non-differentiable points where AD does not compute a correct derivative is of measure zero, which we achieve by choosing a different monad lifting.

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.[1] This technique played a key role in modeling the Covid-19 pandemic (Flaxman et al. 2020). For similar reasons, AD through iterative differential equation solvers is important for probabilistic modeling of pharmacokinetics (Tsiros et al. 2019). 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 (Carpenter et al. 2015). Finally, iteration gives a convenient way of achieving numerically stable approximations to complex functions (such as the Conway–Maxwell–Poisson density function (Goodrich 2017). 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

$$\mathbf{iterate}\left(\begin{array}{l}\mathbf{case}\, z\, \mathbf{of}\, \langle i, y'\rangle \to \mathbf{let}\, y = t(i, x)\, \mathbf{in} \\ \mathbf{case}\ -\epsilon < y < \epsilon\ \mathbf{of}\, \mathbf{inl}\ \_ \to \mathbf{inr}\, x \mid \mathbf{inr}\ \_ \to \mathbf{inl}\, \langle i + 1, y + y'\rangle\end{array}\right) \mathbf{from}\, z = \langle 0, 0\rangle, \qquad (1)$$

where ϵ is a cutoff for floating-point underflow.

Next, recursive neural networks (Tai et al. 2015) 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 (Zhang et al. 2016). In fact, Jeong et al. (2018) have 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 Socher et al. (2011), to give an idea of how iteration and recursive types (in the form of inductive types of labeled trees) naturally arise in a functional implementation of such neural net architectures. We imagine that more applications of AD applied to recursive programs with naturally will emerge as the technique becomes 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 online machine learning applications (Shalev-Shwartz et al. 2012), where data is processed in real time as it becomes available. Recursion and more

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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.

Prerequisites. We assume some familiarity with category theory (see, for instance, Mac Lane 2013): the concepts of and basic facts about categories, functors, natural transformations, (co)limits, adjunctions, and (co) monads. We also assume that the reader knows the most basic definitions in enriched category theory (see, for instance, Kelly 1982): the concepts of $\mathcal{V}$-categories, $\mathcal{V}$-functors, and $\mathcal{V}$-natural transformations. We recall the definitions and results we need for $\mathcal{V}$-monads and their Keisli $\mathcal{V}$-categories (the interested reader can find more details in Dubuc 1970). Later in this paper, we will also consider $\mathcal{V}$-(co)limits, $\mathcal{V}$-adjunctions, and $\mathcal{V}$-(co)monadicity but only for the specific case of $\mathcal{V} = \boldsymbol{\omega}\mathbf{Cpo}$ with its cartesian structure. In these cases, we ensure to spell out all details to make the paper as self-contained as possible.

Convention. 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.

2. Key Ideas

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

Language

We consider an idealized functional language with product types $\tau \times \sigma$, sum types $\tau \sqcup + \sigma$, and 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 $x : \mathbf{real} \vdash \mathbf{sign}\ (x) : \mathbf{1} \sqcup + \mathbf{1}$, where we write $\mathbf{1}$ for the empty product; $\mathbf{sign}\ t$ computes the *sign of a real number* t and is undefined at $t = \underline{0}$; we can use it to define a lazy conditional on real numbers $\mathbf{if}\ r\ \mathbf{then}\ t\ \mathbf{else}\ s \stackrel{\mathrm{def}}{=} \mathbf{case}\ \mathbf{sign}\ r\ \mathbf{of}\{\_ \to t \mid \_ \to r\}$ of the kind that is often used in AD libraries like Stan (Carpenter et al. 2015).

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 $\Gamma \vdash \mathbf{iterate}\ t\ \mathbf{from}\ x = s : \sigma$, 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 $\mathbf{let}\ x = \mu x.t\ \mathbf{in}\ t$; note that we can define iteration with recursion.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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 $\mathcal{D}(\mathbf{real}) \stackrel{\text{def}}{=} \mathbf{real} \times \mathbf{real}$ of reals and by replacing all constants $\underline{c}$, n-ary primitive operations op and sign function **sign** in the program as[2]

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

We extend $\mathcal{D}$ to all other types and programs in the unique homomorphic (structure preserving way), by using structural recursion. So, for example, $\mathcal{D}(\tau \to \sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau) \to \mathcal{D}(\sigma)$, $\mathcal{D}(x) \stackrel{\text{def}}{=} x$, $\mathcal{D}(\mathbf{let}\ x = t\ \mathbf{in}\ s) = \mathbf{let}\ x = \mathcal{D}(t)\ \mathbf{in}\ \mathcal{D}(s)$, 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 discrete ω-cpo $[\![\mathbf{real}]\!] \stackrel{\text{def}}{=} \mathbb{R}$ of real numbers, in which $r \leq r'$ if and only if $r = r'$. We interpret $\underline{c}$ as the constant $[\![\underline{c}]\!] \stackrel{\text{def}}{=} c \in \mathbb{R}$. We interpret 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, finally, we interpret **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 **1** and $r > 0$ 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 2.1 (Forward AD correctness, Theorem 9.1 with $k = 1$ in main text). *For any program $x : \tau \vdash t : \sigma$ for $\tau = \mathbf{real}^m, \sigma = \mathbf{real}^l$ (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 $[\![t]\!]$ is differentiable on its domain and*

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

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

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Importantly, the program t might use higher-order functions, iteration, recursion, etc. 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.

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:

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

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

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

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

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

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 $1 \leq i \leq n$, $(f_i, f_i') \in T^n_{\tau_i}$, 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. Crucial features to enable the inductive steps for iteration and recursion in the proof of the fundamental lemma are that $T^n_{\mathbf{real}}$ and P^n_τ are closed under suprema of countable chains and that P^n_τ contains the least element.

As $T^k_{\mathbf{real}^k}$ contains, in particular, $(\mathrm{id}, ((x_1, \ldots, x_k), (v_1, \ldots, v_k)) \mapsto ((x_1, v_1), \ldots, (x_n, v_k)))$, our correctness theorem follows.

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 $\boldsymbol{\mu}\alpha.\tau$, which binds α in τ and computes a canonical fixpoint of $\alpha \mapsto \tau$. We extend our AD transformation homomorphically on terms and types. For example, on types, we define

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

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 (di)natural transformations. As the category of ω-cpos and partial morphisms has the structure to interpret recursive types, we have a canonical *minimal invariant*

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

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

for the mixed-variance endofunctors $[\![\tau]\!]$ on $\boldsymbol{\omega}\mathbf{Cpo}$ that types τ denote (Levy 2012). We interpret $[\![\boldsymbol{\mu}\alpha.\tau]\!] \stackrel{\text{def}}{=} \mu[\![\tau]\!]$.

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

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

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 Pitts (1996). 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, 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 a right adjoint $\boldsymbol{\omega}\mathbf{Cpo}$-enriched functor $G : \boldsymbol{\omega}\mathbf{Cpo}^n \to \boldsymbol{\omega}\mathbf{Cpo}$ (such as $G(X) = \boldsymbol{\omega}\mathbf{Cpo}^n(Y, X)$ for some $Y \in \boldsymbol{\omega}\mathbf{Cpo}^n$), consider the category $\mathbf{SScone}$ of n-ary logical relations, which has objects (X, T), where $X \in \boldsymbol{\omega}\mathbf{Cpo}^n$ and T is a (full) sub-ω-cpo of GX, and morphisms $(T, X) \to (T', X')$ are $\boldsymbol{\omega}\mathbf{Cpo}^n$-morphisms $f : X \to X'$ such that $y \in T$ implies $Gf(y) \in T'$.

Theorem 2.2 (Logical relations for recursive types, special case of Theorem 10.14 in main text). *Let L be a strong monad on $\mathbf{SScone}$ 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 L takes the initial object to the terminal one and that $G(\eta_X)^{-1}(L(T, X)) = T$, where we write $\eta_X : X \to X_\perp$ for the unit of the partiality monad on $\boldsymbol{\omega}\mathbf{Cpo}^n$. Then, the Kleisli functor $\mathbf{SScone} \hookrightarrow \mathbf{SScone}_L$ gives a model for recursive types.*

Spelled out in non-categorical terms, we are considering logical relations

$$T_\tau \subseteq G([\![\tau]\!]) \qquad\qquad P_\tau = L(T_\tau, [\![\tau]\!]) \subseteq G([\![\tau]\!]_\perp)$$

and we require that the relation P_0 at the initial object (empty type) is precisely the singleton set $\{\perp\}$ (containing the least element) and $G([\![\tau]\!] \subseteq [\![\tau]\!]_\perp)^{-1}(P_\tau)$ (which we think of as the total elements in P_τ) coincide with T_τ.

In particular, in our case, we work binary logical relations ($n = 2$) with

$$G(X, X') = \boldsymbol{\omega}\mathbf{Cpo}^2((\mathbb{R}^n, \mathbb{R}^n \times \mathbb{R}^n), (X, X'))$$

and the monad lifting

$$L(T, (X, X')) = \Big\{(\gamma, \gamma') \mid \gamma^{-1}(X) \times \mathbb{R}^n = \gamma'^{-1}(X') \text{ is a proper open subset and for all differentiable } \delta : \mathbb{R}^n \to \gamma^{-1}(X) \text{ we have } (\gamma \circ \delta, (x, v) \mapsto (\gamma(\delta(x)), \gamma'(D\delta(x, v)))) \in T\Big\} \cup T.$$

Consequently, we can define the logical relations $T_{\boldsymbol{\mu}\alpha.\tau}$ 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 $\overleftarrow{\mathcal{D}}(\mathbf{real}) \stackrel{\text{def}}{=} \mathbf{real} \times \mathbf{vect}$ and

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

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

$$\mathrm{Wrap}_s(x) \stackrel{\text{def}}{=} \mathbf{case}\ x\ \mathbf{of}\ \langle x_1, \ldots, x_s \rangle \to \langle \langle x_1, \bar{e}_1 \rangle, \ldots, \langle x_s, \bar{e}_s \rangle \rangle. \tag{1.1}$$

We define $[\![\mathbf{vect}]\!] \stackrel{\text{def}}{=} \mathbb{R}^\infty \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \mathbb{R}^k$ 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.[3] We show that $\overleftarrow{\mathcal{D}}(t)$ implements the transposed derivative $D[\![t]\!]^t$ of $[\![t]\!]$ in the following sense.

Theorem 2.3 (Reverse AD correctness, Theorem 9.1 with $k = \infty$ in main text). *For any program* $x : \tau \vdash t : \sigma$ *for* $\tau = \mathbf{real}^s, \sigma = \mathbf{real}^l$,

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

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

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

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

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

where we consider $(\mathbb{R}^\infty)^n$ as a type of linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^\infty$ and we write e_i for the i-th standard basis vector of $\mathbb{R}^n$. 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 $1 \leq i \leq n$, $(f_i, f_i') \in T^n_{\tau_i}$, then $(x \mapsto [\![t]\!](f_1(x), \ldots, f_n(x)), (x, L) \mapsto [\![\overleftarrow{\mathcal{D}}(t)]\!](f_1'(x, L), \ldots, f_n'(x, L))) \in P^n_\sigma$.

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

$$(\mathrm{id}, ((x_1, \ldots, x_s), L) \mapsto ((x_1, Le_1), \ldots, (x_s, Le_s))),$$

our theorem follows.

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 $\boldsymbol{\mu}\alpha.\mathbf{1} \sqcup +\alpha \times \mathbf{real}$, 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, for example

$$\mathcal{D}(\tau[]) \stackrel{\text{def}}{=} \mathcal{D}(\tau)[] \qquad \mathcal{D}(\mathbf{generate}) \stackrel{\text{def}}{=} \mathbf{generate} \qquad \mathcal{D}(\mathbf{map}) \stackrel{\text{def}}{=} \mathbf{map} \qquad \mathcal{D}(\mathbf{foldr}) \stackrel{\text{def}}{=} \mathbf{foldr}$$

and similarly for dual numbers reverse AD.

Almost everywhere differentiability

Taking inspiration from Lee et al. (2020),Mazza and Pagani (2021), and Huot et al. (2023), we can increase our ambitions and show that, given sufficiently nice primitive operations, our AD methods compute correct derivatives *almost everywhere* for (almost everywhere) terminating programs in our language. In fact, a minor adaptation of our methods yields these results. Indeed, as long as we assume that all our (partial) primitive operations denote functions that are piecewise analytic under analytic partition (PAP) and are defined on a c-analytic subset (meaning: a countable union of analytic subsets) of $\mathbb{R}^n$, then we can simply redefine our logical relations

$$T^n_{\mathbf{real}} \stackrel{\text{def}}{=} \{(\gamma, \gamma') \mid \gamma \text{ is PAP and } \gamma' = (x, v) \mapsto (\gamma(x), \gamma''(x, v)) \text{ for an intensional derivative } \gamma'' \text{ of } \gamma\}$$
$$\{A_i \subseteq \mathbb{R}^n\}_{i \in I} \text{ of } \gamma'^{-1}([\![\mathcal{D}(\tau)]\!]) \text{ and there exist open neighbourhoods } U_i \text{ of } A_i \text{ with functions}$$
$$\gamma_i : U_i \to [\![\tau]\!], \gamma_i' : U_i \times \mathbb{R}^n \to [\![\mathcal{D}(\tau)]\!] \text{ such that } \gamma|_{A_i} = \gamma_i|_{A_i} \text{ and } \gamma'|_{A_i \times \mathbb{R}^n} = \gamma_i'|_{A_i \times \mathbb{R}^n} \text{ and}$$
$$\text{for all analytic } \delta : \mathbb{R}^n \to U_i \text{ we have that } (\gamma_i \circ \delta, (x, v) \mapsto (\gamma_i(\delta(x)), \gamma_i'(D\delta(x, v)))) \in T^n_\tau\Big\}.$$

to conclude that

- any program $x : \tau \vdash t : \sigma$ for $\tau = \mathbf{real}^m, \sigma = \mathbf{real}^l$ in our language denotes a partial PAP function $[\![t]\!]$defined on a c-analytic subset;
- our AD transformation $\mathcal{D}(t)$ computes $\left([\![t]\!], g\right)$ for an intensional derivative g of $[\![t]\!]$, which coincides almost everywhere in the domain with the (standard) derivative $D[\![t]\!]$ of $[\![t]\!]$.

Consequently, if our program terminates almost everywhere, the AD transformation computes the correct derivative almost everywhere.

3. Overview

We briefly sketch the high-level plan of attack that we will follow in this paper. In this work, our guiding philosophy is to consider categorical models of functional PL as categories with a certain kind of structure:

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

- certain chosen types **real** and morphisms op for basic types of real numbers and primitive operations (such as cos and multiplication) between real numbers;
- finite products, to represent tuples;
- finite coproducts, to represent variants;
- exponentials, to build types of curried and higher-order functions;
- a (partiality) monad such that the Kleisli category supports certain morphism-level fixpoint operators to represent (call-by-value) iteration (while-loops) and recursion;
- certain object-level fixpoint operators to represent recursively defined types.

Due to its technical complexity, we isolate the discussion of the last feature (recursive types) in Section 10.

A crisp formulation for the last two bullet points above is hard to find in the literature. Therefore, we develop such a formulation in detail in Sections 4 and 10.2. We further, in Sections 5 and 10.6, show that there are particularly well-behaved models of these features if we have enrichment over ω-cpos (ω-chain complete partial orders). All the models we consider, except for the syntax, will fall into this well-behaved class.

We will generally identify the syntax of a PL, up to $\beta\eta$-equality, with the *freely generated* (or initial) such category **Syn**. We can then understand AD as a structure preserving functor (preserving all the structure described above)

$$\mathcal{D} : \mathbf{Syn} \to \mathbf{Syn}$$

that sends **real** to a type of pairs **real** $\times$ **vect** (for storing both values and derivatives) and each primitive operation op to its derivative. We discuss this in Sections 6, 10.1, and 10.4.

In order to phrase the correctness of AD, we first need to fix the meaning of the programs in our language. That is sometimes done using an operational semantics that describes how programs are evaluated in time. Here, we work, instead, with a denotational semantics that systematically assigns spaces (in our case, ω-cpos) to types and mathematical functions (in our case, ω-cocontinuous, monotone functions) to programs. We can understand such a denotational semantics as a structure preserving functor to the category $\boldsymbol{\omega}$**Cpo** of ω-cpos:

$$[\![-]\!] : \mathbf{Syn} \to \boldsymbol{\omega}\mathbf{Cpo},$$

which sends **real** to the real numbers $\mathbb{R}$ and each primitive operation op to the function $[\![\mathrm{op}]\!]$ it intends to implement. Importantly, we are now in a position to phrase a correctness theorem for AD by relating the semantics of an AD-transformed program to the mathematical derivative of that program. We discuss this in Sections 7 and 10.7.

Our proof strategy for this correctness theorem is a logical relations proof, which we again phrase categorically. Given a functor $G : \boldsymbol{\omega}\mathbf{Cpo}^n \to \boldsymbol{\omega}\mathbf{Cpo}$, we can consider the category **SScone** of n-ary logical relations, which has objects (X, T), where $X \in \boldsymbol{\omega}\mathbf{Cpo}^n$ and T is a (full) sub-ω-cpo GX and morphisms $(T, X) \to (T', X')$ are $\boldsymbol{\omega}\mathbf{Cpo}^n$-morphisms $f : X \to X'$ such that $y \in T$ implies $Gf(y) \in T'$. Our proof proceeds by making a sensible choice of G (Section 9.1) and giving a new categorical semantics $\overline{[\![-]\!]} : \mathbf{Syn} \to \mathbf{SScone}$ in the category of logical relations, such that the following diagram commutes and that the commuting of this diagram immediately implies the correctness of AD (Sections 9.4, 9.5, 9.6, and 9.7):

$$\begin{array}{ccc} \mathbf{Syn} & \xrightarrow{\overline{[\![-]\!]}} & \mathbf{SScone} \\ {\scriptstyle (\mathrm{id},\mathcal{D})}\downarrow & & \downarrow{\scriptstyle \mathrm{forget}} \\ \mathbf{Syn} \times \mathbf{Syn} & \xrightarrow[[\![-]\!]\times[\![-]\!]]{} & \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}. \end{array}$$

How do we construct such a semantics though? For that, we need to show that the category of logical relations has all the structure needed to interpret our language. That is:

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

- we show that products, coproducts, exponentials, and morphism-level fixpoint operators for iteration and recursion exist in our category of logical relations (Sections 8 and 9.2);
- we show that object-level fixpoint operators for recursive types exist in our category of logical relations (Section 10.8);
- we choose a sensible logical relation $\overline{[\![\mathbf{real}]\!]}$ for **real** to precisely capture correct differentiation, and we demonstrate that each primitive operation respects the chosen logical relations (Section 9.3).

The only choices that need to be made to construct this interpretation are the choice of logical relations associated with **real** and with partial functions (in the form of a lifting of the partiality monad to logical relations). All other required structure is unique thanks to a universal property. Further, the commutativity of the diagram above follows automatically from the initiality of **Syn** among all categorical models.

We believe that our methods for interpreting morphism- and object-level fixpoint combinators in categories of logical relations can be a simplification compared to existing methods. We therefore aim to present them in a reusable way.

4. Categorical Models for CBV Languages: *CBV* Pairs and Models

The aim of this section is to establish a class of categorical models for call-by-value (CBV) languages with free notions of recursion and iteration. This material will be of importance as we will later consider particular examples of such models constructed from (1) the syntax of PL, (2) a concrete denotational semantics for PL in terms of ω-cpos, and (3) that concrete semantics decorated with logical relations to enable correctness proofs of AD.

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

$$\mathcal{T} = \left(T : \mathcal{V} \to \mathcal{V}, \mathrm{m} : T^2 \to T, \eta : \mathrm{id}_{\mathcal{V}} \to T\right), \tag{3}$$

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 $\mathrm{m} \cdot (\eta T) = \mathrm{id}_T = \mathrm{m} \cdot (T\eta)$.[4]

Let $\mathcal{T} = (T, \mathrm{m}, \eta)$ and $\mathcal{T}' = \left(T', \mathrm{m}', \eta'\right)$ 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

$$\left(H : \mathcal{V} \to \mathcal{V}', \phi : HT \to T'H\right), \tag{4}$$

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

$$\phi \cdot (H\eta) = \left(\eta' H\right) \qquad \text{and} \qquad \left(\mathrm{m}'H\right) \cdot \left(T'\phi\right) \cdot (\phi T) = \phi \cdot (H\mathrm{m}). \tag{5}$$

By the universal property of Kleisli categories, denoting by $J : \mathcal{V} \to C$ and $J : \mathcal{V}' \to C'$ the universal Kleisli functors, the oplax morphims (4) correspond bijectively with pairs of functors $\left(H : \mathcal{V} \to \mathcal{V}', \overline{H} : C \to C'\right)$ such that the diagram (7) commutes.

Definition 4.1 (*CBV* pair). *A CBV pair is a pair $(\mathcal{V}, \mathcal{T})$ where $\mathcal{V}$ is bicartesian closed category (i.e., a cartesian closed category with finite coproducts) 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 $\left(\mathcal{V}', \mathcal{T}'\right)$ is a strictly bicartesian closed functor $H : \mathcal{V} \to \mathcal{V}'$ that is a strict morphism between $\mathcal{T}$ and $\mathcal{T}'$, that is such that $HT = T'H$ and (H, id) defines a monad op-functor (4). This defines a category of CBV pairs and CBV pair morphisms, denoted herein by $\mathfrak{C}_{\mathrm{p}}$.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Remark 4.2. If $(\mathcal{V}, \mathcal{T})$ is a *CBV* pair, since $\mathcal{T}$ is $\mathcal{V}$-enriched, we get a $\mathcal{V}$-enriched Kleisli category C. We denote by

$$C\,[-,-] = \left(- \Rightarrow^k -\right) : C^{\mathrm{op}} \times C \to \mathcal{V} \tag{6}$$

the $\mathcal{V}$-enriched hom functor. It should be noted that, if we denote by $(X \Rightarrow Y) = \mathcal{V}\,[X, Y]$ the exponential in $\mathcal{V}$, we have that $C\,[X, Y] = \left(X \Rightarrow^k Y\right) = (X \Rightarrow TY)$, which is the so-called Kleisli exponential and corresponds to the function types for our language.

Denoting by C and C' the respective Kleisli categories, each morphism

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

of *CBV* pairs gives rise to a commutative square

$$\begin{array}{ccc} C & \xrightarrow{\overline{H}} & C' \\ {\scriptstyle J}\uparrow & & \uparrow{\scriptstyle J'} \\ \mathcal{V} & \xrightarrow[H]{} & \mathcal{V}' \end{array} \tag{7}$$

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 C. That is to say, $\left(H, \overline{H}\right)$ strictly preserves the distributive closed Freyd-categorical structure.[5]

4.1 CBV models: term recursion and iteration

In order to interpret our language defined in Section 6, 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. In contrast with most other references we are aware of, we give an explicit discussion of the case of iteration, even though it is definable in terms of recursion. The reason is that there are interesting PL with iteration but without recursion, of which we might want to prove properties.

Definition 4.3 (Free recursion and iteration). *Let $(\mathcal{V}, \mathcal{T})$ be a CBV pair and C the corresponding $\mathcal{V}$-enriched Kleisli category.*

- *A free recursion for $(\mathcal{V}, \mathcal{T})$ is a family of morphisms*

$$\mu = \left(\mu^{W,Y} : \mathcal{V}\,[C\,[W, Y], C\,[W, Y]] \longrightarrow C\,[W, Y]\right)_{(W,Y)\in C\times C} \tag{8}$$

  in $\mathcal{V}$.
- *A free iteration for $(\mathcal{V}, \mathcal{T})$ is a family of morphisms*

$$\mathsf{itt} = \left(\mathsf{itt}^{W,Y} : C\,[W, W \sqcup Y] \longrightarrow C\,[W, Y]\right)_{(W,Y)\in C\times C} \tag{9}$$

  in $\mathcal{V}$.

Definition 4.4 (*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 $(\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$ and $\left(\mathcal{V}', \mathcal{T}', \mu', \mathsf{itt}'\right)$ is a morphism H between the underlying CBV pairs such that $H\left(\mu^{W,Y}\right) = \mu'^{HW,HY}$ and $H\left(\mathsf{itt}^{W,Y}\right) =$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$\mathsf{itt}'^{HW,HY}$, *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 two *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

$$(\mathcal{V}_0 \times \mathcal{V}_1, \mathcal{T}_0 \times \mathcal{T}_1, (\mu_0, \mu_1), (\mathsf{itt}_0, \mathsf{itt}_1)) \tag{10}$$

where $(\mu_0 \times \mu_1)^{\left(W,W'\right),\left(Y,Y'\right)} = \mu_0^{W,Y} \times \mu_1^{W',Y'}$ and $(\mathsf{itt}_0 \times \mathsf{itt}_1)^{\left(W,W'\right),\left(Y,Y'\right)} = \left(\mathsf{itt}_0^{W,Y} \times \mathsf{itt}_1^{W',Y'}\right)$.

5. Canonical Fixed Points from 2-Dimensional Structure

The aim of this section is to specialize to a class of particularly well-behaved *CBV* pairs and models, as they possess canonical iteration and recursion constructs that arise from a universal property of being a least fixed point. To phrase this universal property (and thus obtain uniqueness), we need the homsets of our models to be categories (or, posets, if we do not care to distinguish different ways of comparing morphisms), leading us to consider $\mathcal{V}$ that are **Cat**- or **Pos**-enriched. To obtain existence of these fixed points, it is sufficient to have colimits of countable chains, leading us to specialize to $\mathcal{V}$ that are enriched over ω-cocomplete categories or posets.

We denote by $\boldsymbol{\omega}$**Cpo** the usual category of ω-cpos. The objects of $\boldsymbol{\omega}$**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 $f \in \boldsymbol{\omega}$**Cpo** (W, Y) is a pointed $\boldsymbol{\omega}$**Cpo**-morphism if W is pointed and f preserves the least element.

It is well known that $\boldsymbol{\omega}$**Cpo** is bicartesian closed.[6] We consider $\boldsymbol{\omega}$**Cpo**-enriched categories w.r.t. the cartesian structure. Henceforth, if $\mathcal{V}$ is an $\boldsymbol{\omega}$**Cpo**-enriched category and W, Y are objects of $\mathcal{V}$, we denote by $\mathcal{V}(W, Y)$ the $\boldsymbol{\omega}$**Cpo**-enriched hom, that is to say, the ω-cpo of morphisms between W and Y.

An $\boldsymbol{\omega}$**Cpo**-category $\mathcal{V}$ has finite $\boldsymbol{\omega}$**Cpo**-products if it has a terminal object and we have a natural isomorphism of ω-cpos

$$(-,-) : \mathcal{V}(Z, W) \times \mathcal{V}(Z, Y) \cong \mathcal{V}(Z, W \times Y),$$

or, equivalently, if it has finite products and tupling is an $\boldsymbol{\omega}$**Cpo**-morphism. Dually, an $\boldsymbol{\omega}$**Cpo**-category $\mathcal{V}$ has finite $\boldsymbol{\omega}$**Cpo**-coproducts if it has an initial object and we have a natural isomorphism of ω-cpos

$$[-,-] : \mathcal{V}(W, Z) \times \mathcal{V}(Y, Z) \cong \mathcal{V}(W \sqcup Y, Z),$$

or, equivalently, if it has finite coproducts and cotupling is an $\boldsymbol{\omega}$**Cpo**-morphism. We say that an $\boldsymbol{\omega}$**Cpo**-functor $F : \mathcal{V} \to \mathcal{V}'$ has an $\boldsymbol{\omega}$**Cpo**-right adjoint $U : \mathcal{V}' \to \mathcal{V}$ if we have a natural isomorphism of ω-cpos

$$\mathcal{V}'(FZ, W') \cong \mathcal{V}(Z, UW'),$$

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

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

$$\mathfrak{G}_{\mathcal{V}(Z,-)} : \mathcal{V}\text{-}\mathbf{Cat} \to \boldsymbol{\omega}\mathbf{Cpo}\text{-}\mathbf{Cat} \tag{11}$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

Definition 5.1 (*CBV* **ωCpo**-pair). *A CBV* **ωCpo**-*pair is a CBV pair* $(\mathcal{V}, \mathcal{T})$ *in which* $\mathcal{V}$ *is an* **ωCpo**-*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 **ωCpo**-*pair morphism between* $(\mathcal{V}, \mathcal{T})$ *and* $(\mathcal{V}', \mathcal{T}')$ *is an* **ωCpo**-*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* **ωCpo**-*morphism for any* $(W, Y) \in \mathsf{ob}\,\mathcal{V} \times \mathsf{ob}\,\mathcal{V}$. *This defines a category of CBV* **ωCpo**-*pairs, denoted herein by* **ωCPO**-$\mathfrak{C}_{\mathcal{BV}}$.

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

5.1 Fixpoints: term recursion and iteration

Recall that, if A is a pointed ω-cpo and $q : A \to A$ is an endomorphism in **ωCpo**, then q has a *least fixed point* given by the colimit of the chain

$$\bot \nrightarrow q(\bot) \nrightarrow \cdots \nrightarrow q^n(\bot) \nrightarrow \cdots \tag{12}$$

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

Henceforth, let $(\mathcal{V}, \mathcal{T})$ be a *CBV* **ωCpo**-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, corresponds to the usual action of $\mathcal{V}$ on $\mathcal{C}$.

By hypothesis, for any $W, Y, Z \in \mathcal{V}$, the ω-cpo $\mathcal{C}(Z \otimes W, Y) \cong \mathcal{V}(Z, \mathcal{C}[W, Y])$ is pointed. Let us write $\Lambda_Z^{W,Y}$ for the isomorphism from left to right. Then, we can define

$$\begin{aligned} \overline{\mu}_Z^{W,Y} : \mathcal{V}(Z \times \mathcal{C}[W, Y], \mathcal{C}[W, Y]) &\to \mathcal{V}(Z, \mathcal{C}[W, Y]) \\ f &\mapsto \mathrm{lfp}\left(h \mapsto f \circ \left(Z \times h\right) \circ \partial_Z\right) \end{aligned} \tag{13}$$

$$\begin{aligned} \overline{\mathsf{it}}_Z^{W,Y} : \mathcal{C}(Z \otimes W, W \sqcup Y) &\to \mathcal{C}(Z \otimes W, Y) \\ f &\mapsto \mathrm{lfp}\left(\begin{array}{c} h \mapsto [h, J(\pi_Y)] \circ d_{Z,W,Y} \circ \left(Z \otimes f\right) \\ \circ a_{Z,Z,W} \circ (\partial_Z \otimes \mathrm{id}_W) \end{array}\right) \end{aligned} \tag{14}$$

where $\partial_Z = (\mathrm{id}_Z, \mathrm{id}_Z) : Z \to Z \times Z$ is the diagonal morphism, $d_{Z,W,Y} : Z \otimes (W \sqcup Y) \to (Z \otimes W) \sqcup (Z \otimes Y)$ is the distributor, and $a_{Z,W,Y} : (Z \times W) \otimes Y \to Z \otimes (W \otimes Y)$ is the associator. Since the morphisms above are **ωCpo**-natural in $Z \in \mathcal{V}$, they give rise to the families of morphisms

$$\mu_\omega = \left(\mu_\omega^{W,Y}\right)_{(W,Y) \in \mathcal{C} \times \mathcal{C}} \stackrel{\mathrm{def}}{=} \left(\overline{\mu}^{W,Y}_{\mathcal{V}[\mathcal{C}[W,Y],\mathcal{C}[W,Y]]}\left(\mathsf{eval}_{\mathcal{C}[W,Y],\mathcal{C}[W,Y]}\right)\right)_{(W,Y) \in \mathcal{C} \times \mathcal{C}} \tag{15}$$

$$\mathsf{it}_\omega = \left(\mathsf{it}_\omega^{W,Y}\right)_{(W,Y) \in \mathcal{C} \times \mathcal{C}} \stackrel{\mathrm{def}}{=} \Lambda^{W,Y}_{\mathcal{V}[W,T(W \sqcup Y)]}\left(\overline{\mathsf{it}}^{W,Y}_{\mathcal{V}[W,T(W \sqcup Y)]}\left(\mathsf{eval}_{W,T(W \sqcup Y)}\right)\right)_{(W,Y) \in \mathcal{C} \times \mathcal{C}} \tag{16}$$

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.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Lemma 5.2 (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 $\boldsymbol{\omega}\mathbf{Cpo}$-functor and, for any $(W, Y) \in \mathsf{ob}\,\mathcal{V} \times \mathsf{ob}\,\mathcal{V}$,

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

is a pointed $\boldsymbol{\omega}\mathbf{Cpo}$-morphism, we get that, indeed, H respects the free iteration and free recursion as defined in (15) and (16). □

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

$$(\mathcal{V}_0, \mathcal{T}_0) \times (\mathcal{V}_1, \mathcal{T}_1) = (\mathcal{V}_0 \times \mathcal{V}_1, \mathcal{T}_0 \times \mathcal{T}_1) \tag{17}$$

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

6. 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 Smeding and Vákár (2023).

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 Section 6.4). The source language contains the programs that we differentiate, while we use the target language to represent those derivatives.

6.1 Source language as a standard call-by-value language with iteration and recursion

We consider a standard (coarse-grain) CBV 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 Section 7.3, **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 $\mathbb{R} \rightharpoonup \mathbf{1} \sqcup \mathbf{1}$ defined on $\mathbb{R}^- \cup \mathbb{R}^+$, which takes $\mathbb{R}^-$ to the left component and $\mathbb{R}^+$ to the right component.

Although it is straightforward to consider more general settings, we also add the assumption that the primitive operations implement differentiable functions (see Assumption 7.6).

We treat this operations in a schematic way as this reflects the reality of practical AD 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.

$\tau, \sigma, \rho ::=$		types	|	$\mathbf{1} \mid \tau_1 \times \tau_2$	products
	| **real**	numbers	|	$\tau \to \sigma$	function
	| $\mathbf{0} \mid \tau \sqcup + \sigma$	sums			
$v, w, u ::=$		values	|	$\langle\,\rangle \mid \langle v, w\rangle$	tuples
	| x, y, z	variables	|	$\lambda x.t$	abstractions
	| $\underline{c}$	constants	|	$\mu x.t$	term recursion
	| **inl** v | **inr** v	sum inclusions			

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

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

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

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

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

Figure 1. Typing rules for a basic source language with real conditionals, where $R\subset\mathbb{R}$ is a fixed set of real numbers containing 0.

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

Figure 2. Typing rules for term recursion and iteration.

$t,s,r ::=$		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
$\mid$	$\mathbf{case}\, r\, \mathbf{of}\, \{\begin{matrix}\mathbf{inl}\, x\to t\\ \mid \mathbf{inr}\, y\to s\end{matrix}\}$	sum match			

We use sugar $\mathbf{if}\, r\, \mathbf{then}\, t\, \mathbf{else}\, s \stackrel{\mathrm{def}}{=} \mathbf{case}\, \mathbf{sign}\, r\, \mathbf{of}\, \{\_\to t \mid \_\to r\}$, $\mathbf{fst}\, t \stackrel{\mathrm{def}}{=} \mathbf{case}\, t\, \mathbf{of}\, \langle x,\_\rangle \to x$, $\mathbf{snd}\, t \stackrel{\mathrm{def}}{=} \mathbf{case}\, t\, \mathbf{of}\, \langle\_,x\rangle \to x$, and $\mathbf{let\ rec}\, f(x) = t\, \mathbf{in}\, s \stackrel{\mathrm{def}}{=} \mathbf{let}\, f = \mu f.\lambda x.t\, \mathbf{in}\, s$. In fact, we can consider iteration as syntactic sugar as well:

$\mathbf{iterate}\, t\, \mathbf{from}\, x = s \stackrel{\mathrm{def}}{=} (\mu z.\lambda x.\mathbf{case}\, t\, \mathbf{of}\, \{\mathbf{inl}\, x' \to z\, x' \mid \mathbf{inr}\, x'' \to x''\})\, s.$

The computations are typed according to the rules of Figs. 1 and 2, *where* $\mathsf{R}\subset\mathbb{R}$ *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 Moggi (1989) for our language, which we list in Fig. 3. We could impose further equations for the iteration construct as is done in Bloom and Ésik (1993) and Goncharov et al. (2015) as well as for the basic operations op and the sign function **sign** . However, such equations are unnecessary for our development.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

Figure 3. Basic $\beta\eta$-equational theory for our language. We write $\beta\eta$-equality as $\equiv$ to distinguish it from equality in let-bindings. We write $\#x_1, \ldots, x_n\equiv$ 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.

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

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

Figure 4. 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 $\mathbf{real}^{i+1} = \mathbf{real}^i \times \mathbf{real}$.

6.2. Target language

We define our *target language* by extending the source language adding the following syntax, with the typing rules of Fig. 4.

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

The operational semantics of the target language depends on the intended behavior for the *AD* macro $\mathcal{D}$ defined in Section 6.4. In our context, we want **vect** to implement a vector space ((co)tangent vectors), 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 (Smeding and Vákár 2023).

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 Section 9.6 and 9.7. Although this perspective is unimportant for our correctness statement, the reader might want to view **vect** as a computation type encompassing *computational effects* for the vector space operations $\bar{e}_i$, $(*)$, and $(+)$ with *handlers* given by the terms $\mathfrak{h}_i t$.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

For each primitive operation $\mathrm{op} \in \mathrm{Op}_n$ ($n \in \mathbb{N}$) and each constant $c \in \mathsf{R}$:
$H(\mathbf{real}) \in \mathrm{ob}\,\mathcal{V}$; $H(\mathbf{sign}) \in \mathcal{C}(H(\mathbf{real}), 1 \sqcup 1) = \mathcal{V}(H(\mathbf{real}), T(1 \sqcup 1))$;
$H(\underline{c}) \in \mathcal{V}(1, H(\mathbf{real}))$; $H(\mathrm{op}) \in \mathcal{C}(H(\mathbf{real})^n, H(\mathbf{real})) = \mathcal{V}(H(\mathbf{real})^n, TH(\mathbf{real}))$.

Figure 5. Assignment that gives the universal property of the source language.

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

Figure 6. Assignment that gives the universal property of the target language.

We are particularly interested in the case that $(\mathbf{vect}, +, *, \overline{0})$ implements the vector space $\left(\mathbb{R}^k, +, *, 0\right)$, for some $k \in \mathbb{N} \cup \{\infty\}$,[7] 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

$$\mathfrak{p}_{k \to i} : \mathbb{R}^k \to \mathbb{R}^i, \tag{18}$$

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 Section 7.4.

6.3 *The CBV models* $(\mathbf{Syn_V}, \mathbf{Syn_S}, \mathbf{Syn_\mu}, \mathbf{Syn_{it}})$ *and* $(\mathbf{Syn_V^{tr}}, \mathbf{Syn_S^{tr}}, \mathbf{Syn_\mu^{tr}}, \mathbf{Syn_{it}^{tr}})$

As discussed in Appendix A, we can translate our coarse-grain languages to fine-grain CBV languages. The fine-grain languages corresponding to the source and target languages correspond to the *CBV* models

$$\left(\mathbf{Syn}_V, \mathbf{Syn}_\mathcal{S}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right) \qquad \text{and} \qquad \left(\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_\mathcal{S}^{\mathbf{tr}}, \mathbf{Syn}_\mu^{\mathbf{tr}}, \mathbf{Syn}_{\mathsf{it}}^{\mathbf{tr}}\right) \tag{19}$$

with the following universal properties.

Proposition 6.1 (Universal property of *CBV* models (19)). *Let* $(\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$ *be a CBV model. Assume that Figs. 5 and 6 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. 5.*
2. *There is a unique CBV model morphism* $\mathcal{H} : \left(\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_\mathcal{S}^{\mathbf{tr}}, \mathbf{Syn}_\mu^{\mathbf{tr}}, \mathbf{Syn}_{\mathsf{it}}^{\mathbf{tr}}\right) \to (\mathcal{V}, \mathcal{T}, \mu, \mathsf{itt})$ *that extends* H *and respects the assignment of Fig. 6.*

6.4. *Dual numbers AD transformation for term recursion and iteration*

Let us fix, for all $n \in \mathbb{N}$, for all $\mathrm{op} \in \mathrm{Op}_n$, and 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. 7, a structure preserving macro $\mathcal{D}$ on the types and computations of our language for performing AD.

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

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

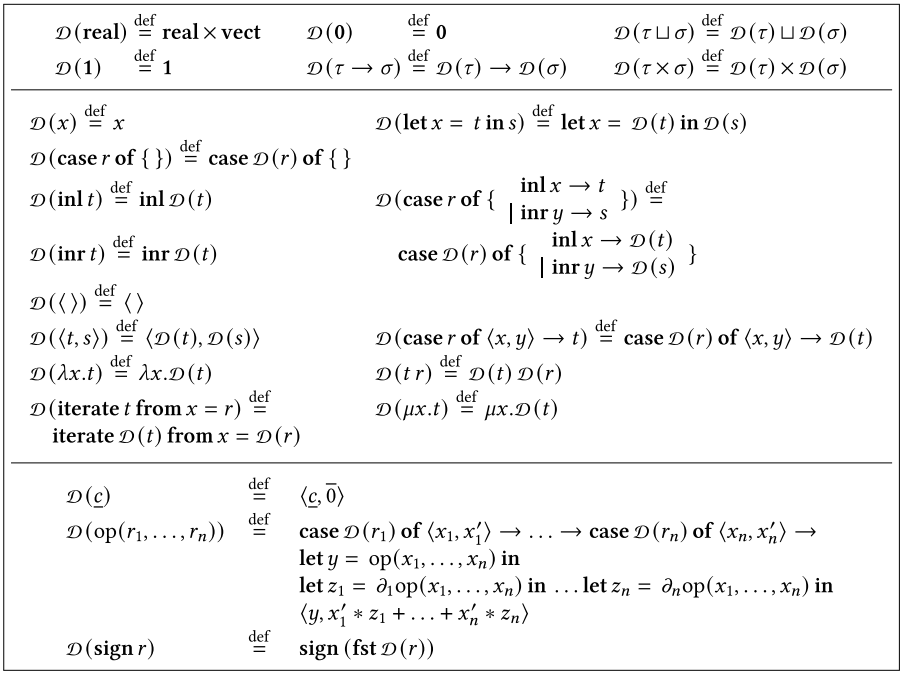


Figure 7. AD macro $\mathcal{D}(-)$ defined on types and computations. All newly introduced variables are chosen to be fresh. We provide a more efficient way of differentiating **sign** in Appendix B.

Lemma 6.2 (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* $t \equiv s$, *then* $\mathcal{D}(t) \equiv \mathcal{D}(s)$.

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 Section 6.2).

As an example, for the program of Equation (1), $\mathcal{D}$ computes, modulo some $\beta\eta$-equality to aid legibility, the following derivative (where we also define $\mathcal{D}(\mathbf{int}) \stackrel{\text{def}}{=} \mathbf{int}$, $\mathcal{D}(t < s < r) \stackrel{\text{def}}{=} \mathbf{fst}(\mathcal{D}(t)) < \mathbf{fst}(\mathcal{D}(s)) < \mathbf{fst}(\mathcal{D}(r))$, and $\partial_i(+)(x, y) \stackrel{\text{def}}{=} 1$):

$$\mathbf{iterate} \left(\begin{array}{l} \mathbf{case}\, z\, \mathbf{of}\, \langle i, \langle y_1', y_2'\rangle\rangle \to \mathbf{let}\, \langle y_1, y_2\rangle = \mathcal{D}(t)(i, x)\, \mathbf{in} \\ \mathbf{case} - \epsilon < y_1 < \epsilon\, \mathbf{of}\, \{\mathbf{inl}\, \_ \to \mathbf{inr}\, x \mid \mathbf{inr}\, \_ \to \mathbf{inl}\, \langle i + 1, \langle y_1 + y_1', y_2 + y_2'\rangle\rangle\} \end{array} \right) \mathbf{from}\, z = \langle 0, 0\rangle.$$

6.5. AD transformation as a CBV model morphism

By the universal property of $(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathrm{it}})$ established in Proposition 6.1, the assignment defined in Fig. 8 induces a unique *CBV* model morphism

$$\mathbb{D} : (\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathrm{it}}) \to (\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathbf{tr}}, \mathbf{Syn}_\mu^{\mathbf{tr}}, \mathbf{Syn}_{\mathrm{it}}^{\mathbf{tr}}). \tag{20}$$

The macro $\mathcal{D}$ defined in Fig. 7 is encompassed by (20).

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\mathbb{D}(\mathbf{real}) \stackrel{\text{def}}{=} \mathbf{real} \times \mathbf{vect} \in \mathsf{ob}\,\mathbf{Syn}_V^{\mathbf{tr}}, \qquad \mathbb{D}(\underline{c}) \stackrel{\text{def}}{=} (\underline{c}, \underline{0}) \in \mathbf{Syn}_V^{\mathbf{tr}}(1, \mathbf{real} \times \mathbf{vect}),$$
$$\mathbb{D}(\mathrm{op}) \stackrel{\text{def}}{=} \lambda y_1.\lambda \ldots \lambda y_n.\overrightarrow{d}\,\mathrm{op}\,(y_1, \ldots, y_n) \in \mathbf{Syn}_V^{\mathbf{tr}}\left((\mathbf{real} \times \mathbf{vect})^n, \mathbf{Syn}_S\,(\mathbf{real} \times \mathbf{vect})\right),$$
$$\mathbb{D}(\mathbf{sign}) \stackrel{\text{def}}{=} (\mathbf{sign} \circ \pi_1) \in \mathbf{Syn}_V^{\mathbf{tr}}\left(\mathbf{real} \times \mathbf{vect}, \mathbf{Syn}_S\,(1 \sqcup 1)\right),$$

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

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

Figure 8. AD assignment.

7. Concrete Semantics for the AD Transformation

We give a concrete denotational semantics for our source and target languages in terms of ω-cpos. In fact, our semantics for the target language will be parameterized by $k \in \mathbb{N} \cup \{\infty\}$. This parameter allows us to give a uniform treatment of various variants of AD. For basic forward mode AD, we will use $k = 1$. Other $k \in \mathbb{N}$ correspond to vectorized forms of forward mode AD, and $k = \infty$ is primarily of interest for dual numbers reverse AD, which can be viewed as an optimized version of a vectorized forward AD with dynamically sized tangent vectors.

We will use these semantics to phrase and prove correctness of AD in the rest of this paper. We also recall some facts about and fix notation for derivatives, in order to phrase sufficient and necessary conditions on the semantics of primitive operations and their AD transformations.

7.1 Basic concrete model

The most fundamental example of a *CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair is given by $\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$ where $(-)_\perp$ is the (lax idempotent) monad that freely adds a least element $\perp$ 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 $\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \times \left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) = \left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$, where, by abuse of language, $\left((C, C')\right)_\perp = \left((C)_\perp, \left(C'\right)_\perp\right)$. By Lemma 5.2, we obtain *CBV* models

$$\mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \qquad \text{and}$$
$$\mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) = \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \times \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right).$$

For example, the program from Equation (1) is interpreted as the function

$$\mathbb{R} \to \mathbb{R}_\perp \tag{21}$$

$$x \mapsto \begin{cases} \perp & \text{if } N_{[\![t]\!],x} = \infty \\ \perp & \text{if } [\![t]\!](i, x) = \epsilon \text{ for some } i < N_{[\![t]\!],x} \\ \sum_{i=0}^{N_{[\![t]\!],x}-1} [\![t]\!](i, x) & \text{otherwise} \end{cases} \tag{22}$$

where

- $\lceil z \rceil^\epsilon = z$ if $|z| > \epsilon$, $\lceil z \rceil^\epsilon = 0$ if $|z| < \epsilon$ and $\lceil z \rceil^\epsilon = \perp$ otherwise;
- $N_{[\![t]\!],x}$ is the smallest natural number i such that $\lceil [\![t]\!](i, x) \rceil^\epsilon = 0$.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

7.2 Differentiable functions and interleaved derivatives

Henceforth, unless stated otherwise, the cartesian spaces $\mathbb{R}^n$ and its subspaces are endowed with the respective discrete **ωCpo***-structures, in which $r \leq r'$ if and only if $r = r'$.*

Definition 7.1 (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* **ωCpo** *with the respective discrete* **ωCpo***-structures)*

$$\begin{aligned} \phi_{n,k} : \quad \mathbb{R}^n \times \left(\mathbb{R}^k\right)^n &\to \left(\mathbb{R} \times \mathbb{R}^k\right)^n \\ \left((x_j)_{j\in\mathbb{I}_n}, (y_j)_{j\in\mathbb{I}_n}\right) &\mapsto \left(x_j, y_j\right)_{j\in\mathbb{I}_n}. \end{aligned} \tag{23}$$

For each open subset $U \subset \mathbb{R}^n$, 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 Definition 7.2, Remark 7.3, and Lemma 7.4, 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 7.2 (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 $W_i \subset \mathbb{R}^n$ and $V_i \subset \mathbb{R}^{m_i}$. In this case, for each $k \in (\mathbb{N} \cup \{\infty\})$, we define the function:*

$$\begin{aligned} \mathfrak{D}^k g : \phi_{n,k}\left(U \times \left(\mathbb{R}^k\right)^n\right) &\to \coprod_{j\in L}\left(\phi_{m_j,k}\left(V_i \times \left(\mathbb{R}^k\right)^{m_i}\right)\right) \\ z \qquad\qquad &\mapsto \iota_{m_j} \circ \phi^{V_j}_{m_j,k}\left(g(x), \tilde{w}\cdot g'(x)^t\right), \text{ if } \phi^{-1}_{n,k}(z) = (x,w) \in W_i \times \left(\mathbb{R}^k\right)^n \end{aligned} \tag{24}$$

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 **ωCpo***), and $g'(x)^t : \mathbb{R}^{m_i} \to \mathbb{R}^n$ 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 7.3. It should be noted that, in Definition 7.2, 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 (6.4).

Lemma 7.4. *Let $\dot{g}$ be a function with domain as in (6.4). 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 7.5 (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* **ωCpo**. *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 $\mathbb{R}^{n_i}$;*

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

- *the corresponding total function* (25) *is differentiable.*

$$\underline{h_i} = h|_{U_i} : U_i \to \coprod_{j\in L} \mathbb{R}^{m_j} \tag{25}$$

$$\mathfrak{d}^k(h) : \coprod_{r\in K} \left(\mathbb{R}\times\mathbb{R}^k\right)^{n_r} \to \left(\coprod_{j\in L}\left(\mathbb{R}\times\mathbb{R}^k\right)^{m_j}\right)_\perp \tag{26}$$

In this case, for each $k \in \mathbb{N}\cup\{\infty\}$, *we define* (26) *to be the morphism induced by* $[\mathfrak{d}^k(h_r)]_{r\in K}$ *where, for each* $i \in K$, $\mathfrak{d}^k(h_i)$ *is defined by* (27), *which is just the corresponding canonical extension of the map* $\mathfrak{D}^k h_i$.

$$\mathfrak{d}^k(h_i) : \left(\mathbb{R}\times\mathbb{R}^k\right)^{n_i} \to \left(\coprod_{j\in L}\left(\mathbb{R}\times\mathbb{R}^k\right)^{m_j}\right)_\perp \tag{27}$$

$$z \mapsto \begin{cases} \mathfrak{D}^k h_i(z), & \text{if } z \in \phi_{n_i,k}\left(U_i \times \left(\mathbb{R}^k\right)^{n_i}\right) \subset \left(\mathbb{R}\times\mathbb{R}^k\right)^{n_i}; \\ \perp, & \text{otherwise.} \end{cases}$$

For example, the partial function h of Equation (21) has the following derivative $\mathfrak{D}^k(h)$:

$$\mathbb{R}\times\mathbb{R}^k \to (\mathbb{R}\times\mathbb{R}^k)_\perp \tag{28}$$

$$(x,v) \mapsto \begin{cases} \perp & \text{if } N_{[\![t]\!],x} = \infty \\ \perp & \text{if } [\![t]\!](i,x) = \epsilon \text{ for some } i < N_{[\![t]\!],x} \\ \sum_{i=0}^{N_{[\![t]\!],x}-1} \mathfrak{D}^k([\![t]\!](i,-))(x,v) & \text{otherwise} \end{cases} \tag{29}$$

7.3 The semantics for the source language

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

We denote by $\mathbb{R}$ the discrete ω-cpo of real numbers, in which $r \le r'$ if and only if $r = r'$, and we define $\mathsf{sign} : \mathbb{R} \to (1\sqcup 1)_\perp$ by (31), where $\iota_1, \iota_2 : 1 \to 1\sqcup 1$ are the two coprojections of the coproduct.

$$[\![-]\!] : \left(\mathbf{Syn}_V, \mathbf{Syn}_T, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}}\right) \to \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \tag{30}$$

$$\mathsf{sign}(x) = \begin{cases} \perp, & \text{if } x = 0 \\ \iota_1(*), & \text{if } x < 0 \\ \iota_2(*), & \text{if } x > 0 \end{cases} \tag{31}$$

By the universal property of $(\mathbf{Syn}_V, \mathbf{Syn}_S, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathsf{it}})$, there is only one *CBV* model morphism (30) consistent with the assignment of Fig. 9 where c is the constant that $\underline{c}$ intends to implement, and, for each $\mathsf{op} \in \mathrm{Op}_n$, f_{op} is the partial map that op intends to implement.

The *CBV* model morphism (30) (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 7.6. For each $n \in \mathbb{N}$ and $\mathsf{op} \in \mathrm{Op}_n$, $[\![\mathsf{op}]\!] = f_{\mathsf{op}} : \mathbb{R}^n \to (\mathbb{R})_\perp$ is differentiable.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$[\![\mathbf{real}]\!] \stackrel{\text{def}}{=} \mathbb{R} \in \mathrm{ob}\,\boldsymbol{\omega}\mathbf{Cpo}; \qquad [\![\underline{c}]\!] \stackrel{\text{def}}{=} \mathrm{c} \in \boldsymbol{\omega}\mathbf{Cpo}\,(1,\mathbb{R})\,;$$
$$[\![\mathrm{op}]\!] \stackrel{\text{def}}{=} f_{\mathrm{op}} \in \boldsymbol{\omega}\mathbf{Cpo}\,(\mathbb{R}^n,(\mathbb{R})_\perp)\,; \qquad [\![\mathbf{sign}\,]\!] \stackrel{\text{def}}{=} \mathrm{sign} \in \boldsymbol{\omega}\mathbf{Cpo}\,(\mathbb{R},(1\sqcup 1)_\perp)\,.$$

Figure 9. Semantics' assignment for each primitive operation $\mathrm{op} \in \mathrm{Op}_n$ ($n \in \mathbb{N}$) and each constant $c \in R$.

7.4 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 Proposition 6.1, there is a unique *CBV* model morphism (32) that extends $[\![-]\!]$ and is consistent with the assignment given by the vector structure (33) 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 $i \in \mathbb{N}^*$.

$$[\![-]\!]_k : \left(\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathbf{tr}}, \mathbf{Syn}_{\mu}^{\mathbf{tr}}, \mathbf{Syn}_{\mathrm{it}}^{\mathbf{tr}}\right) \to \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \tag{32}$$

$$\left([\![\mathbf{vect}]\!]_k, [\![+]\!]_k, [\![*]\!]_k, [\![\bar{0}]\!]_k\right) := \left(\mathbb{R}^k, +, *, 0\right) \tag{33}$$

7.5 Soundness of $\mathcal{D}$ for primitive operations

Definition 7.7 (Sound for primitives). *A macro $\mathcal{D}$ as defined in Fig. 7 and its corresponding CBV model morphism $\mathbb{D}$ as defined in (20) 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 $\mathfrak{d}_j\left(f\right)(x_1,\ldots,x_n) = \mathfrak{d}^1\left(f\right) \circ \phi_{n,1}\left((x_1,\ldots,x_n), e_j^n\right)$, where e_j^n the j-th vector of the canonical basis of $\mathbb{R}^n$.

Lemma 7.8. *The macro $\mathcal{D}$ defined in Fig. 7 is sound for primitives provided that*

$$[\![\langle \mathrm{op}(y_1,\ldots,y_n), \partial_j\mathrm{op}(y_1,\ldots,y_n)\rangle]\!] = \mathfrak{d}_j\left([\![\mathrm{op}]\!]\right), \tag{34}$$

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

8. Enriched Scone and Subscone

Here, we present general, reusable results about logical relations proofs for languages with recursive features. We phrase these in terms of category theory. Concretely, we discuss two categorical perspectives on logical relations, both of which are constructions to build a new categorical semantics out of two existing semantics $\mathcal{B}$ and $\mathcal{D}$. The first perspective, called the *scone*, is as simple as a plain comma category $\mathcal{D}\downarrow G$ of the identity along a suitable functor $G : \mathcal{B} \to \mathcal{D}$ between the two existing semantics. It gives a proof-relevant perspective in which we may distinguish different witnesses demonstrating the truth of a predicate. The second perspective, called the *subscone*, arises as a suitable reflective subcategory of the scone. It crucial property is that its objects are chosen such that they represent only proof-irrelevant predicates, meaning that we can think of its morphisms simply as $\mathcal{B}$-morphisms that respect the predicates.

Here, we focus, in particular, on characterizing when the scone and subscone are $\boldsymbol{\omega}\mathbf{Cpo}$-bicartesian closed categories, getting us most of the way to a *CBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair. We discuss the remaining ingredient of lifting the (pointed) monad to the subscone in Section 9.2.[8]

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

8.1 Scone: proof-relevant categorical logical relations

Given an **ωCpo**-functor $G : \mathcal{B} \to \mathcal{D}$, the comma **ωCpo**-category $\mathcal{D} \downarrow G$ of the identity along G in **ωCpo-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 $\mathcal{D}$; we think of these as pairs of a $\mathcal{B}$-object C and a proof-relevant predicate (D, j) on $G(C)$;
- a morphism $(D, C, j) \to (D', C', h)$ between objects of $\mathcal{D} \downarrow G$ is a pair (35) making (36) commutative in $\mathcal{D}$; we think of these as $\mathcal{B}$-morphisms α_1 that respect the predicates, as evidenced by α_0:

$$\alpha = \left(\alpha_0 : D \to D', \alpha_1 : C \to C'\right) \tag{35}$$

$$\begin{array}{ccc} D & \xrightarrow{\alpha_0} & D' \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle h} \\ G(C) & \xrightarrow[G(\alpha_1)]{} & G(C') \end{array} \tag{36}$$

- if $\alpha = \left(\alpha_0 : D \to D', \alpha_1 : C \to C'\right), \beta = \left(\beta_0 : D \to D', \beta_1 : C \to C'\right) : \left(D, C, j\right) \to \left(D', C', h\right)$, 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 Lucatelli Nunes and Vákár (Lucatelli Nunes 2022, Section 9), we have:

Theorem 8.1 (Monadic-comonadic scone). *Let $G : \mathcal{B} \to \mathcal{D}$ be a right* **ωCpo***-adjoint functor. Assuming that $\mathcal{D}$ has finite* **ωCpo***-products and $\mathcal{B}$ has finite* **ωCpo***-coproducts, the* **ωCpo***-functor*

$$\mathcal{L} : \mathcal{D} \downarrow G \to \mathcal{D} \times \mathcal{B}, \tag{37}$$

defined by $\left(D \in \mathcal{D}, C \in \mathcal{B}, j : D \to G(C)\right) \mapsto (D, C)$, is **ωCpo***-comonadic and* **ωCpo***-monadic.*[9] *This implies, in particular, that $\mathcal{L}$ creates (and strictly preserves)* **ωCpo** *-limits and colimits.*[10]

By Theorem 8.1 and the enriched adjoint triangle theorem,[11] we have:

Corollary 8.2. *Let $G : \mathcal{B} \to \mathcal{D}$ be a right* **ωCpo***-adjoint functor between* **ωCpo***-bicartesian closed categories. In this case, $\mathcal{D} \downarrow G$ is an* **ωCpo***-bicartesian closed category. Moreover, if $\mathcal{D} \times \mathcal{B}$ is* **ωCpo***-cocomplete, so is $\mathcal{D} \downarrow G$.*

Theorem 8.1 and Corollary 8.2 are **ωCpo**-enriched versions of the fundamental results of Lucatelli Nunes and Vákár (Lucatelli Nunes 2022, Section 9). The details and proofs are presented in Appendix C.

8.2 Subscone: proof-irrelevant categorical logical relations

Henceforth, we assume that **Sub** *$(\mathcal{D} \downarrow G)$ is a full*[12] *reflective*[13] *and replete*[14] **ωCpo***-subcategory of $\mathcal{D} \downarrow G$. We denote, herein, by $\mathfrak{T}_{sub}$ the idempotent* **ωCpo***-monad induced by the* **ωCpo***-adjunction.*

Recall that *a morphism q in* **ωCpo** *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* **ωCpo***-category $\mathcal{B}$ is full* if $\mathcal{B}\left(B, j\right)$ is full in **ωCpo** for any $B \in \mathcal{B}$.

Furthermore, recall that *an* **ωCpo***-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* **ωCpo***-morphism.* It should be noted that the 2-functor underlying a locally full **ωCpo**-functor is *locally fully faithful.*

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Moreover, since every full morphism in **ωCpo** is injective on objects, *every locally full* **ωCpo**-*functor is faithful (locally injective on objects).*

Assumption 8.3. We require that:

(Sub. 1) whenever $\big(D \in \mathcal{D}, C \in \mathcal{B}, j\big) \in \mathbf{Sub}\,(\mathcal{D} \downarrow G)$, j is a full morphism in $\mathcal{B}$;

(Sub. 2) $G : \mathcal{B} \to \mathcal{D}$ is a right **ωCpo**-adjoint functor between **ωCpo**-bicartesian closed categories;

(Sub. 3) $\mathfrak{T}_{sub}$ strictly preserves **ωCpo**-products;

(Sub. 4) Diag. (39) commutes.

$$\mathbf{Sub}\,(\mathcal{D} \downarrow G) \longrightarrow \mathcal{D} \downarrow G \xrightarrow{\mathcal{L}} \mathcal{D} \times \mathcal{B} \xrightarrow{\pi_{\mathcal{B}}} \mathcal{B} \tag{38}$$

$$\begin{array}{ccccc} \mathcal{D} \downarrow G & & \xrightarrow{\mathfrak{T}_{sub}} & & \mathcal{D} \downarrow G \\ & {\scriptstyle \mathcal{L}} \searrow & & \swarrow {\scriptstyle \mathcal{L}} & \\ \mathcal{D} \times \mathcal{B} & & \xrightarrow[\mathcal{B}]{} \mathcal{B} \xleftarrow[\pi_{\mathcal{B}}]{} & & \mathcal{D} \times \mathcal{B} \end{array} \tag{39}$$

We denote by $\underline{\mathcal{L}} : \mathbf{Sub}\,(\mathcal{D} \downarrow G) \to \mathcal{B}$ the **ωCpo**-functor given by the composition (38) where the unlabeled arrow is the full inclusion.

Proposition 8.4. *The full inclusion* $\mathbf{Sub}\,(\mathcal{D} \downarrow G) \to \mathcal{D} \downarrow G$ *creates (and strictly preserves)* **ωCpo**-*limits and* **ωCpo**-*exponentials. Moreover, if* $\mathcal{D} \downarrow G$ *is* **ωCpo**-*cocomplete, so is* $\mathbf{Sub}\,(\mathcal{D} \downarrow G)$.

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

By Assumption (sub. 3) of Assumption 8.3, $\mathfrak{T}_{sub}$ is commutative, and hence, $\mathbf{Sub}\,(\mathcal{D} \downarrow G) \to \mathcal{D} \downarrow G$ creates **ωCpo**-exponentials.

Since $\mathfrak{T}_{sub}$ is idempotent, $\mathbf{Sub}\,(\mathcal{D} \downarrow G)$ is **ωCpo**-cocomplete whenever $\mathcal{D} \downarrow G$ is. □

Corollary 8.5. $\mathbf{Sub}\,(\mathcal{D} \downarrow G)$ *is an* **ωCpo**-*bicartesian closed category. Moreover, if* $\mathcal{D} \times \mathcal{B}$ *is* **ωCpo**-*cocomplete, so is* $\mathbf{Sub}\,(\mathcal{D} \downarrow G)$.

Proof It follows from Proposition 8.4 and Corollary 8.2. □

Theorem 8.6. *The* **ωCpo**-*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* **ωCpo**-*colimits.*

Proof The **ωCpo**-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 **ωCpo**-weighted limits and colimits. Since $\mathfrak{T}_{sub}$ is idempotent and (39) commutes, this implies that $\underline{\mathcal{L}}$ strictly preserves **ωCpo**-limits and colimits.

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

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

The locally fully faithfulness (and, hence, faithfulness) of $\underline{\mathcal{L}}$ follows from Condition (sub. 1) of Assumption 8.3. □

Remark 8.7 (Proof-irrelevance). Condition (sub. 1) of Assumption 8.3 ensures that our subscone indeed gives us a proof-irrelevant approach to logical relations: in particular, as stressed above, it

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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 $\mathbf{Sub}\,(\mathcal{D} \downarrow G)$.

Generally, we see a tradeoff between using proof-relevant logical relations proofs via an interpretation in the scone or proof-irrelevant ones via an interpretation in the subscone. The scone is generally better behaved as a category, as it tends to be both monadic and comonadic by Theorem 8.1, while the subscone tends to only be monadic. The objects and morphisms of the subscone, however, can be simpler to work with, as we do not need to track witnesses thanks to their uniqueness. In the rest of this paper, we work with the (proof-irrelevant) subscone, mostly to conform to conventions in the literature.

9. Correctness of Dual Numbers AD

In this section, we show that, as long as the macro $\mathcal{D}$ defined in Fig. 7 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 9.1. *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 (i.e., types not containing function types), we have that* $[\![t]\!]$ *is differentiable and, moreover,*

$$[\![\mathcal{D}\,(t)]\!]_k = \mathfrak{d}^k\,([\![t]\!]) \tag{40}$$

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

We take the following steps to achieve this result:

- In Section 9.1, we fix a particular functor $G : \mathcal{B} \to \mathcal{D}$ for which to consider the scone, as well as a particular reflective subscone of the scone. This sets the concrete stage in which our logical relations proof will take place.
- In Section 9.2, we choose a particular lifting of the partiality monad to this subscone, to establish a reasoning principle for derivatives of partial functions.
- In Section 9.3, we fix a lifting of the interpretation of the primitive type **real** to the subscone, establishing a reasoning principle for derivatives of real-valued functions. We further show that, for a macro $\mathcal{D}$ that is sound for primitives, $[\![\mathcal{D}\,(-)]\!]_k$ respects the logical relation and hence yields an interpretation of our full language in the subscone.
- In Section 9.4, we show that logical relations at data types (composite types not containing function types) also capture correct differentiation.
- In Section 9.5, we derive our fundamental AD correctness theorem from the interpretation of our language in the subscone, and in Sections 9.6 and 9.7, we spell out in more detail what this correctness theorem entails for the choice of semantics $[\![\mathbf{vect}]\!]_k = \mathbb{R}^k$.

9.1 *Fixing a particular subscone* $\mathbf{Sub}\,(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k})$

Henceforth, we follow the notation and definitions established in Section 7. In particular, *unless stated otherwise, the cartesian spaces* $\mathbb{R}^n$ *and its subspaces are endowed with the discrete* $\boldsymbol{\omega}\mathbf{Cpo}$*-structure, in which* $r \leq r'$ *if and only if* $r = r'$*.*

For each $(n, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, we define the $\boldsymbol{\omega}\mathbf{Cpo}$-functor (41). We consider the full reflective $\boldsymbol{\omega}\mathbf{Cpo}$-subcategory $\mathbf{Sub}\,(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k})$ of $\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}$ whose objects are triples (42) such that j is full (and, hence, injective on objects).

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$G_{n,k} \stackrel{\text{def}}{=} \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}\left(\left(\mathbb{R}^n, \left(\mathbb{R} \times \mathbb{R}^k\right)^n\right), (-,-)\right) : \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo} \to \boldsymbol{\omega}\mathbf{Cpo} \tag{41}$$

$$\left(D \in \boldsymbol{\omega}\mathbf{Cpo},\ \left(C, C'\right) \in \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo},\ \left(j : D \to G_{n,k}\left(C, C'\right)\right) \in \boldsymbol{\omega}\mathbf{Cpo}\right) \tag{42}$$

That is, we are considering what Barthe et al. (2020) calls *open logical relations* (where closed logical relations would correspond to the case of $G_{n,k} = \boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}\,((1,1),-)$).

The $\boldsymbol{\omega}\mathbf{Cpo}$-functor $G_{n,k}$ together with $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ satisfies Assumption 8.3. Therefore:

Proposition 9.2. $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ *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 8.5 and Theorem 8.6. □

9.2 Lifting the partiality monad to the subscone

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

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

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

$$\mathcal{P}_{n,k}\left(D, \left(C, C'\right), j\right)_\perp \stackrel{\text{def}}{=} \left(\underline{\mathcal{P}_{n,k}\left(D, \left(C, C'\right), j\right)_\perp}, \left(\left(C\right)_\perp, \left(C'\right)_\perp\right), \mathtt{j}_X\right) \tag{43}$$

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

$$\{\perp\} \sqcup D \sqcup \left(\coprod_{U \in \mathfrak{O}_n} \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)\left(\mathsf{Diff}_{(U,n,k)}, \left(D, \left(C, C'\right), j\right)\right)\right) \tag{44}$$

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 $G_{n,k}\left(\eta_C, \eta_{C'}\right) \circ j : D \to G_{n,k}\left(C, C'\right) \to G_{n,k}\left(\left(C\right)_\perp, \left(C'\right)_\perp\right)$;
- the injections $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)\left(\mathsf{Diff}_{(U,n,k)}, \left(D, \left(C, C'\right), j\right)\right) \to G_{n,k}\left(\left(C\right)_\perp, \left(C'\right)_\perp\right)$, for $U \in \mathfrak{O}_n$, defined by

$$\begin{aligned}\left(\alpha_0, \alpha_1 = \left(\beta_0 : U \to C, \beta_1 : \phi_{n,k}\left(U \times \left(\mathbb{R}^k\right)^n\right) \to C'\right)\right) \\ \mapsto \left(\overline{\beta_0} : \mathbb{R}^n \to \left(C\right)_\perp, \overline{\beta_1} : \left(\mathbb{R} \times \mathbb{R}^k\right)^n \to \left(C'\right)_\perp\right),\end{aligned}$$

  where $\overline{\beta_0}$ and $\overline{\beta_1}$ are the respective corresponding canonical extensions. The image of $\mathtt{j}_X$ forms a sub-ω-cpo because the union $\bigcup_{n \in \mathbb{N}} U_n$ of open sets U_n is open and because D is an ω-cpo.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

$$\overline{\mathrm{m}}_{(D,(C,C'),j)} : \mathcal{P}_{n,k}\left(\mathcal{P}_{n,k}\left(D,\left(C,C'\right),j\right)_\perp\right)_\perp \to \mathcal{P}_{n,k}\left(D,\left(C,C'\right),j\right)_\perp \tag{45}$$

$$\overline{\eta}_{(D,(C,C'),j)} : \left(D,\left(C,C'\right),j\right) \to \mathcal{P}_{n,k}\left(D,\left(C,C'\right),j\right)_\perp . \tag{46}$$

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 $\mathcal{P}_{n,k}(-)_\perp$, 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 $\mathcal{P}_{n,k}(-)_\perp$ 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 $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}(-)_\perp\right)$ 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 $\perp : \left(C_0,C_0'\right) \to \left((C_1)_\perp, \left(C_1'\right)_\perp\right)$, of $\boldsymbol{\omega}\mathbf{Cpo}\,(C_0,(C_1)_\perp) \times \boldsymbol{\omega}\mathbf{Cpo}\left(C_0',\left(C_1'\right)_\perp\right)$ 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 $\mathcal{P}_{n,k}(-)_\perp$, the multiplication and the identity are clearly lifted from $(-)_\perp$ through $\underline{\mathcal{L}}_{n,k}$ as defined above, we have:

Proposition 9.3. *For each* $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}(-)_\perp\right)$ *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* $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}(-)_\perp\right)$ *and* $\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$.

Therefore, by Lemma 5.2, $\mathcal{U}_{\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right)$ is a *CBV* model morphism between the underlying *CBV* models of $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}(-)_\perp\right)$ and $\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$.

9.3 *Logical relations for* real *and deriving a CBV model morphism*

Henceforth, we assume that the macro $\mathcal{D}$ *is sound for primitives (see Definition 7.5).* We establish the *CBV* model morphism (55). We start by establishing the logical relations' assignment.

Let $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. We define the object (47) in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$.

$$\overline{[\![\mathbf{real}]\!]}_{n,k} \stackrel{\text{def}}{=} \left(\left\{\left(f : \mathbb{R}^n \to \mathbb{R}, f^*\right) : f \text{ is differentiable}, f^* = \mathfrak{D}^k f\right\}, \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right), \text{incl.}\right) \tag{47}$$

For each $m \in \mathbb{N}$, $\mathrm{op} \in \mathrm{Op}_m$, and $c \in \mathtt{R}$, we define the morphisms (48), (49), and (50) 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 (20), (30), and (32).

$$\overline{[\![\mathbf{sign}\,]\!]}_k \stackrel{\text{def}}{=} \left(\mathrm{sign}, \mathfrak{d}^k(\mathrm{sign})\right) = \left(\mathrm{sign}, [\![\mathbb{D}\left(\mathbf{sign}\,\right)]\!]_k\right) : \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right) \to \left((1 \sqcup 1)_\perp, (1 \sqcup 1)_\perp\right) \tag{48}$$

$$\overline{[\![\underline{c}]\!]}_k \stackrel{\text{def}}{=} \left(\mathtt{c}, \mathfrak{d}^k(\mathtt{c})\right) : (1,1) \to \left(\mathbb{R}, \mathbb{R} \times \mathbb{R}^k\right) \tag{49}$$

$$\overline{[\![\mathrm{op}]\!]}_k \stackrel{\text{def}}{=} \left([\![\mathrm{op}]\!], \mathfrak{d}^k\left([\![\mathrm{op}]\!]\right)\right) : \left(\mathbb{R}^m, \left(\mathbb{R} \times \mathbb{R}^k\right)^m\right) \to \left((\mathbb{R})_\perp, \left(\mathbb{R} \times \mathbb{R}^k\right)_\perp\right) \tag{50}$$

By Proposition 8.4, we have that the product $\overline{[\![\mathbf{real}]\!]}^m_{n,k}$ in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$ is given by (51). Therefore, *by the chain rule for derivatives*, we have that (48), (49), and (50), respectively, define the morphisms (52), (53), and (54) in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)$, where $\overline{1} \sqcup \overline{1}$ denotes the coproduct of the terminal $\overline{1} = (1,(1,1),\mathrm{id})$ with itself.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\left(\left\{\left(f_j : \mathbb{R}^n \to \mathbb{R}, f_j^*\right)_{j\in\mathbb{I}_m} : f_j^* \text{ is differentiable and } f_j^* = \mathfrak{D}^k f_j, \forall j \in \mathbb{I}_m\right\}, \left(\mathbb{R}, \mathbb{R}\times\mathbb{R}^k\right)^m, \text{incl.}\right)$$
$$\cong \left(\left\{\left(f : \mathbb{R}^n \to \mathbb{R}^m, f^*\right) : f \text{ is differentiable}, f^* = \mathfrak{D}^k f\right\}, \left(\mathbb{R}^m, \left(\mathbb{R}\times\mathbb{R}^k\right)^m\right), \text{incl.}\right). \tag{51}$$

$$\overline{[\![\mathbf{sign}\,]\!]}_{n,k} : \overline{[\![\mathbf{real}]\!]}_{n,k} \to \mathcal{P}_{n,k}\left(\overline{1} \sqcup \overline{1}\right)_\perp \tag{52}$$

$$\overline{[\![\underline{c}]\!]}_{n,k} : \overline{1} \to \overline{[\![\mathbf{real}]\!]}_{n,k} \tag{53}$$

$$\overline{[\![\mathrm{op}]\!]}_{n,k} : \overline{[\![\mathbf{real}]\!]}_{n,k}^m \to \mathcal{P}_{n,k}\left(\overline{[\![\mathbf{real}]\!]}_{n,k}\right)_\perp \tag{54}$$

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

Proposition 9.4. *For each $(n,k) \in \mathbb{N}\times(\mathbb{N}\cup\{\infty\})$, there is only one CBV model morphism*

$$\overline{[\![-]\!]}_{n,k} : \left(\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathbf{tr}}, \mathbf{Syn}_\mu^{\mathbf{tr}}, \mathbf{Syn}_{\mathrm{it}}^{\mathbf{tr}}\right) \to \mathcal{U}_{\mathcal{BV}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_\perp\right) \tag{55}$$

that is consistent with the assignment given by (47), (52), (54), *and* (53). *Moreover, Diag.* (56) *commutes.*

$$\begin{array}{ccc}
\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathrm{it}}\right) & \xrightarrow{(\mathrm{id},\mathbb{D})} & \left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathrm{it}}\right) \times \left(\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathbf{tr}}, \mathbf{Syn}_\mu^{\mathbf{tr}}, \mathbf{Syn}_{\mathrm{it}}^{\mathbf{tr}}\right) \\
\downarrow{\scriptstyle \overline{[\![-]\!]}_{n,k}} & & \downarrow{\scriptstyle [\![-]\!]\times[\![-]\!]_k} \\
\mathcal{U}_{\mathcal{BV}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right), \mathcal{P}_{n,k}\left(-\right)_\perp\right) & \xrightarrow[\mathcal{U}_{\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right)]{} & \mathcal{U}_{\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}\times\boldsymbol{\omega}\mathbf{Cpo}, \left(-\right)_\perp\right)
\end{array} \tag{56}$$

Proof Both $([\![-]\!]\times[\![-]\!]_k)\circ(\mathrm{id}\times\mathbb{D})$ and $\mathcal{U}_{\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right)\circ\overline{[\![-]\!]}_{n,k}$ 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 (48), (49), and (50). □

9.4 AD logical relations for data types

As a consequence of Proposition 9.4, we establish a fundamental result on the logical relations $\overline{[\![-]\!]}_{n,k}$ for data types (i.e., types not containing function types) in our setting: namely, Proposition 9.6. Observe that, by distributivity of products over coproducts, any such data type is isomorphic to $\bigsqcup_{j\in L}\mathbf{real}^{l_j}$ for some finite set L and $l_j \in \mathbb{N}$. Therefore, we start by establishing Lemma 9.5 about our logical relations and the coproducts in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right)$.

Lemma 9.5. *Let $(n,k) \in \mathbb{N}\times(\mathbb{N}\cup\{\infty\})$. If $(g,\dot{g}) \in \coprod_{j\in L}\overline{[\![\mathbf{real}]\!]}_{n,k}^{l_j}$, 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 Proposition 9.2, $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right)$ has coproducts. Moreover, we can conclude that $(g,\dot{g}) \in \coprod_{j\in L}\overline{[\![\mathbf{real}]\!]}_{n,k}^{l_j}$ implies that, for some $r \in L$, we have a pair

$$\left(\underline{g} : \mathbb{R}^n \to \mathbb{R}^{l_r}, \mathfrak{D}^k g : \left(\mathbb{R}\times\mathbb{R}^k\right)^n \to \left(\mathbb{R}\times\mathbb{R}^k\right)^{l_r}\right) \tag{57}$$

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 Definition 7.2, this completes our proof. □

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Proposition 9.6. *Let* $(n,k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. *If* $(g,\dot{g}) \in \underline{\mathcal{P}_{n,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{n,k}\right)_\perp}$, *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 $\underline{\mathcal{P}_{n,k}(-)_\perp}$, 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 $(\underline{g},\underline{\dot{g}}) \in \coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{n,k}$;

s3. $g^{-1}\left(\coprod_{j\in L} \mathbb{R}^{l_j}\right) = W$ is open. Moreover, denoting by (58) the pair consisting of the corresponding total functions, we have that (59) holds for any differentiable map $\alpha : \mathbb{R}^n \to W$.

$$\left(\underline{g} : W \to \left(\coprod_{j\in L} \mathbb{R}^{l_j}\right), \underline{\dot{g}}\right) \tag{58}$$

$$\left(\underline{g} \circ \alpha,\ \underline{\dot{g}} \circ \mathfrak{D}^k\alpha\right) \in \coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{n,k}. \tag{59}$$

If (s1.) holds, following Definition 7.5, we get that g is differentiable and $\dot{g} = \mathfrak{d}^k(g)$ by Remark 7.3.

In case of (s2.), we get $\underline{g}$ is differentiable and $\underline{\dot{g}} = \mathfrak{D}^k\underline{g}$ by Lemma 9.5. Hence g is differentiable and $\dot{g} = \mathfrak{d}^k(g)$.

Finally, in case of (s3.), by Lemma 9.5, 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 7.4, this implies that $\underline{g}$ is differentiable and $\mathfrak{D}^k\underline{g} = \underline{\dot{g}}$. Following Definition 7.5, this completes the proof that g is differentiable and $\dot{g} = \mathfrak{d}^k(g)$. □

Corollary 9.7. *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* (60) *in* $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{s_i,k}\right)$, *then* $g : \coprod_{r\in\mathfrak{L}} \mathbb{R}^{s_r} \to \left(\coprod_{j\in L} \mathbb{R}^{l_j}\right)_\perp$ *is differentiable and* $\dot{g} = \mathfrak{d}^k(g)$.

$$\mathrm{g} : \coprod_{r\in\mathfrak{L}} \overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_\perp \tag{60}$$

$$\iota_i : \overline{[\![\mathbf{real}]\!]}^{s_i}_{s_i,k} \to \coprod_{r\in K} \overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \tag{61}$$

Proof From the hypothesis, for each $i \in \mathfrak{L}$, we conclude that the pair (62) defines the morphism (64), since $\left(\iota_{\mathbb{R}^{s_i}}, \iota_{(\mathbb{R}\times\mathbb{R}^k)^{s_i}}\right)$ defines the coprojection (61) in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{s_i,k}\right)$.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\left(g_i \overset{\text{def}}{=} g \circ \iota_{\mathbb{R}^{s_i}},\ \dot{g}_i \overset{\text{def}}{=} \dot{g} \circ \iota_{(\mathbb{R}\times\mathbb{R}^k)^{s_i}}\right) \tag{62}$$

$$\mathrm{g}_i \overset{\text{def}}{=} \mathrm{g} \circ \iota_i \ :\ \overline{[\![\mathbf{real}]\!]}^{s_i}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_\perp \tag{63}$$

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

$$\left(g_i, \dot{g}_i\right) \in \underline{\mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_\perp}\ . \tag{64}$$

By Proposition 9.6, (64) proves that g_i is differentiable and $\dot{g}_i = \mathfrak{d}^k\left(g_i\right)$. Since this result holds for any $i \in \mathfrak{L}$, we conclude that g is differentiable and $\dot{g} = \mathfrak{d}^k\left(g\right)$. □

9.5 Fundamental AD correctness theorem

We prove Theorem 9.8, which completes the proof of Theorem 9.1.

Theorem 9.8. *Let* $t : \coprod_{r\in\mathfrak{L}} \mathbf{real}^{s_r} \to \mathbf{Syn}_S\left(\coprod_{j\in L} \mathbf{real}^{l_j}\right)$ *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 (65) 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 (65) defines the morphism (66) in $\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{s_i,k}\right)$. Therefore, by Corollary 9.7, we conclude that $[\![t]\!]$ is differentiable and $[\![\mathbb{D}(t)]\!]_k = \mathfrak{d}^k\,([\![t]\!])$.

$$\left([\![t]\!],\ [\![\mathbb{D}(t)]\!]_k\right) \tag{65}$$

$$\overline{[\![t]\!]}_{s_i,k} \ :\ \coprod_{r\in K} \overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_\perp \tag{66}$$

□

9.6 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 9.1, 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 Definition 7.5;
- 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 $z := \phi_{n_i,1}\left(y, w\right)$,

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$[\![\mathcal{D}(t)]\!]_1\left(\phi_{n_i,1}\left(y,w\right)\right) = \mathfrak{d}^1\left([\![t]\!]\right)(z) = \mathfrak{D}^1[\![t]\!]|_{W_j}(z) = \phi_{m_j,1}\left([\![t]\!]\left(y\right), \tilde{w}\cdot[\![t]\!]'(y)^t\right)$$
$$= \phi_{l,1}\left([\![t]\!]\left(y\right), [\![t]\!]'(y)(w)\right), \tag{67}$$

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.

9.7 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 (18) . 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 $\mathbf{wrap}_s \overset{\text{def}}{=} \left(\pi_j, \bar{e}_j\right)_{j\in\mathbb{I}_s} : \mathbf{real}^s \to (\mathbf{real}\times\mathbf{vect})^s$ in $\mathbf{Syn}_V^{\mathbf{tr}}$, which corresponds to the wrapper defined in (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 $\mathtt{wrap}_s\left(y\right) = \phi_{s,k}\left(y, e_1^k, \ldots, e_s^k\right)$.

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

$$[\![\mathcal{D}(t)\circ\mathbf{wrap}_s]\!]_k\left(y\right) = \mathfrak{d}^k\left([\![t]\!]\right)\circ\mathtt{wrap}_s\left(y\right) = \mathfrak{D}^k[\![t]\!]\circ\mathtt{wrap}_s\left(y\right)$$
$$= \mathfrak{D}^k[\![t]\!]\circ\phi_{s,k}\left(y, e_1^k, \ldots, e_s^k\right)$$
$$= \phi_{l,k}\left([\![t]\!]\left(y\right), \mathfrak{p}_{s\to k}[\![t]\!]'(y)^t\right)$$

by Theorem 9.1. 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

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

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

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

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

Again, by Theorem 9.1, 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.

10. AD for Recursive Types and ML-Polymorphism

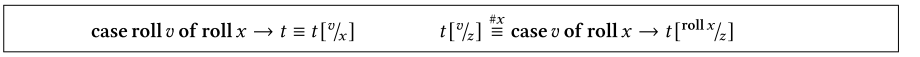
10.1 Syntax for recursive types

We extend both our source and target languages of Sections 6.1 and 6.2 with ML-style polymorphism and type recursion in the sense of FPC (Fiore and Plotkin 1994). That is, we extend types, values, and computations for each of the two languages as

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

Figure 10. Typing rules for the recursive types extension.



$$\mathbf{case\ roll}\, v\, \mathbf{of\ roll}\, x \to t \equiv t[{}^{v}/_{x}] \qquad t[{}^{v}/_{z}] \overset{\#x}{\equiv} \mathbf{case}\, v\, \mathbf{of\ roll}\, x \to t[{}^{\mathbf{roll}\, x}/_{z}]$$

Figure 11. The standard $\beta\eta$-equational theory for recursive types in CBV.

τ, σ, ρ	::=	types	$\mid$	α, β, γ	type variables
	$\mid$	... as before	$\mid$	$\boldsymbol{\mu}\alpha.\tau$	recursive type
v, w, u	::=	values	$\mid$	$\mathbf{roll}\, v$	recursive type introduction
	$\mid$	... as before			
t, s, r	::=	computations	$\mid$	$\mathbf{roll}\, t$	recursive type introduction
	$\mid$	... as before	$\mid$	$\mathbf{case}\, t\, \mathbf{of\ roll}\, x \to s$	recursive type elimination

The new values and computations according to the rules in Fig. 10.

Here, kinding contexts Δ are lists of type variables $\alpha_1, \ldots, \alpha_n$. We consider judgments $\Delta \mid \Gamma \vdash t : \tau$, 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. 11.

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

$$\mathbf{unroll}\, t \overset{\text{def}}{=} \mathbf{case}\, t\, \mathbf{of\ roll}\, x \to x$$

$$\mu x : \tau.t \overset{\text{def}}{=} \mathbf{let}\, body : \chi \to \tau = (\lambda y : \chi.\lambda z : \sigma.\mathbf{let}\, x : \tau = \mathbf{unroll}\, y\, y\, \mathbf{in}\, t\, z)\, \mathbf{in}\, body(\mathbf{roll}\ body). \tag{68}$$

The semantics of the language is, of course, expected to be consistent – meaning that the interpretations of term recursion and recursive types should be compatible according to the definition above. Alternatively, we can consider that the source language is given by the basic language with the typing rules given by Fig. 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 Section 6.2.

10.2 Categorical models for recursive types: rCBV models

Here, we establish the basic categorical model for the syntax of CBV 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 **Cat** $(2, \mathcal{V}\text{-}\mathbf{Cat})$ 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 **Cat** $(2, \mathcal{V}\text{-}\mathbf{Cat})$. In other words, it consists of a pair $E = (E_{\mathcal{V}}, E_{\mathcal{C}})$ of $\mathcal{V}$-enriched functors such that (69) commutes. *A $(\mathcal{V}, \mathcal{T})$-parametric type of degree* 0 *(71) can be identified with the corresponding object $\mathcal{V}$.*

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\begin{array}{ccc} (\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^n & \xrightarrow{E_{\mathcal{C}}} & \mathcal{C} \\ {\scriptstyle (J^{\mathrm{op}} \times J)^n}\uparrow & & \uparrow{\scriptstyle J} \\ (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n & \xrightarrow[E_{\mathcal{V}}]{} & \mathcal{V} \end{array} \tag{69}$$

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 semantics of 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), exponentiation (Kleisli exponential), and type recursion.

Definition 10.1 (Free type recursion). *A free decreasing degree type operator (fddt operator) for $(\mathcal{V}, \mathcal{T})$ is a function (70) 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 $n \in \mathbb{N}$.*

$$\nu : \mathsf{Param}\,(\mathcal{V}, \mathcal{T}) \quad \to \quad \mathsf{Param}\,(\mathcal{V}, \mathcal{T})$$

$$\begin{array}{ccc} (\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{n+1} & \xrightarrow{E_{\mathcal{C}}} & \mathcal{C} \\ {\scriptstyle (J^{\mathrm{op}} \times J)^{n+1}}\uparrow & & \uparrow{\scriptstyle J} \\ (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n+1} & \xrightarrow[E_{\mathcal{V}}]{} & \mathcal{V} \end{array} \quad \mapsto \quad \begin{array}{ccc} (\mathcal{C}^{\mathrm{op}} \times \mathcal{C})^{n} & \xrightarrow{\nu E_{\mathcal{C}}} & \mathcal{C} \\ {\scriptstyle (J^{\mathrm{op}} \times J)^{n}}\uparrow & & \uparrow{\scriptstyle J} \\ (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n} & \xrightarrow[\nu E_{\mathcal{V}}]{} & \mathcal{V} \end{array} \tag{70}$$

A rolling for (70) is a collection (72) of natural transformations such that (73) 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.

$$\left(\left(\mathcal{V}^{\mathrm{op}} \times \mathcal{V}\right)^0 \to \mathcal{V}, \left(\mathcal{C}^{\mathrm{op}} \times \mathcal{C}\right)^0 \to \mathcal{C}\right) \tag{71}$$

$$\underline{\mathsf{roll}} = \left(\mathsf{roll}^E\right)_{E=(E_{\mathcal{V}},E_{\mathcal{C}}) \in \mathsf{Param}(\mathcal{V},\mathcal{T})} \tag{72}$$

$$\begin{array}{ccccc} (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n & & \xrightarrow{\left(\mathrm{id}, \nu E_{\mathcal{V}}^{\mathrm{op}}, \nu E_{\mathcal{V}}\right)} & & (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n+1} \\ & \searrow^{\nu E_{\mathcal{V}}} & \Leftarrow^{\mathsf{roll}^E} & & \downarrow{\scriptstyle E_{\mathcal{V}}} \\ \mathcal{C} & & \xleftarrow[J]{} & & \mathcal{V} \end{array} \tag{73}$$

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

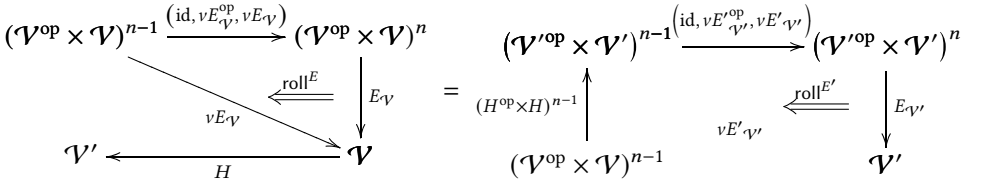
Definition 10.2 (H-compatible). *Let H be a CBV pair morphism between CBV pairs $(\mathcal{V}, \mathcal{T})$ and $\left(\mathcal{V}', \mathcal{T}'\right)$. A pair $\left(E, E'\right) \in \mathsf{Param}\,(\mathcal{V}, \mathcal{T}) \times \mathsf{Param}\left(\mathcal{V}', \mathcal{T}'\right)$ of parametric types is H-compatible if they have the same degree n and the diagram (74) commutes. In particular, if $n = 0$, the pair $\left(E, E'\right)$ is H-compatible if $H\,(E_{\mathcal{V}}) = E'_{\mathcal{V}'}$.*

$$\begin{array}{ccc} (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^n & \xrightarrow{E_{\mathcal{V}}} & \mathcal{V} \\ {\scriptstyle (H^{\mathrm{op}} \times H)^n}\downarrow & & \downarrow{\scriptstyle H} \\ (\mathcal{V}'^{\mathrm{op}} \times \mathcal{V}')^n & \xrightarrow[E'_{\mathcal{V}'}]{} & \mathcal{V}' \end{array} \tag{74}$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Definition 10.3 (*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* $n > 0$, (75) *holds, that is to say,* $H\,(\mathsf{roll}^E) = \mathsf{roll}^E_{(H^{\mathrm{op}} \times H)^{n-1}}$. *The rCBV models and rCBV model morphisms define a category, denoted herein by* $\mathfrak{C}_{\mathcal{RBV}}$.



(75)

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

Remark 10.4. 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 (68)). This defines a forgetful functor

$$\mathcal{R} : \mathfrak{C}_{\mathcal{RBV}} \to \mathfrak{C}_{\mathcal{BV}}. \tag{76}$$

10.3 The rCBV models $(\mathbf{Syn}^{R}_{V}, \mathbf{Syn}^{R}_{\mathcal{S}}, \nu_{\mathbf{Syn}})$ *and* $(\mathbf{Syn}^{\mathbf{Rtr}}_{V}, \mathbf{Syn}^{\mathbf{Rtr}}_{\mathcal{S}}, \nu^{\mathbf{tr}}_{\mathbf{Syn}})$

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

$$\left(\mathbf{Syn}^{\mathsf{R}}_{V}, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}, \underline{\nu}_{\mathbf{Syn}}\right) \qquad \text{and} \qquad \left(\mathbf{Syn}^{\mathsf{Rtr}}_{V}, \mathbf{Syn}^{\mathsf{Rtr}}_{\mathcal{S}}, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right) \tag{77}$$

with the universal property described in Proposition 10.5.

Proposition 10.5 (Universal property of the *rCBV* models (75)). *Let* $(\mathcal{V}, \mathcal{T}, \underline{\nu})$ *be an rCBV model. Assume that Figs. 7 and 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. 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. 8.*

Remark 10.6. By Proposition 6.1, we have (unique) *CBV* model morphisms

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

and

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

that are identity on the primitive operations and types.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

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

Figure 12. The definitions of AD on recursive types.

Proposition 10.5 states that $H \mapsto \mathcal{R}(H) \circ \mathbb{s}$ and $\mathcal{H} \mapsto \mathcal{R}(\mathcal{H}) \circ \mathbb{s}^{\mathsf{t}}$ give the bijections (78) and (79), respectively, showing that our syntax extension for recursive types give a free rCBV model on the syntax without recursive types.

$$\mathfrak{C}_{\mathcal{RBV}}\left(\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right), (\mathcal{V}, \mathcal{T}, \underline{\nu})\right) \cong \mathfrak{C}_{\mathcal{BV}}\left((\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}), \mathcal{R}(\mathcal{V}, \mathcal{T}, \underline{\nu})\right) \quad (78)$$

$$\mathfrak{C}_{\mathcal{RBV}}\left(\left(\mathbf{Syn}_V^{\mathsf{Rtr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{Rtr}}, \underline{\nu}_{\mathbf{Syn}}^{\mathbf{tr}}\right), (\mathcal{V}, \mathcal{T}, \underline{\nu})\right) \cong \mathfrak{C}_{\mathcal{BV}}\left((\mathbf{Syn}_V^{\mathbf{tr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathbf{tr}}, \mathbf{Syn}_{\mu}^{\mathbf{tr}}, \mathbf{Syn}_{\mathsf{it}}^{\mathbf{tr}}), \mathcal{R}(\mathcal{V}, \mathcal{T}, \underline{\nu})\right) \quad (79)$$

10.4 Automatic differentiation for languages with recursive types

We extend our definition of AD to recursive types in Fig. 12. We note that our extension is compatible with our previous definitions if we view term recursion (and iteration) as syntactic sugar.

Lemma 10.7 (Type preservation). *If* $\Delta \mid \Gamma \vdash t : \tau$, *then* $\Delta \mid \mathcal{D}(\Gamma) \vdash \mathcal{D}(t) : \mathcal{D}(\tau)$.

10.5 AD transformation as an rCBV model morphism

By Proposition 10.5, the assignment defined in Fig. 8 induces a unique *rCBV* model morphism (80), which *encompasses the macro* $\mathcal{D}$ *defined by Fig. 7 and extended in Fig. 12.*

$$\mathbb{ID} : \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \left(\mathbf{Syn}_V^{\mathsf{Rtr}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{Rtr}}, \underline{\nu}_{\mathbf{Syn}}^{\mathbf{tr}}\right). \quad (80)$$

10.6 $\boldsymbol{\omega}$**Cpo***-enriched categorical models for recursive types: 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 $\boldsymbol{\omega}$**Cpo**-enriched models, the *rCBV* $\boldsymbol{\omega}$**Cpo**-pairs established in Definition 10.8.[15]

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 C is a pair $u = (u^e, u^p)$ consisting of a C-morphism $u^e : A \to B$, the *embedding*, and a 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[16] whose unit is the identity. In this context, it is also called a lari adjunction (*left adjoint right-inverse*) (see Clementino and Lucatelli Nunes 2024, Section 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[17] $\mathfrak{O}$ in an $\boldsymbol{\omega}$**Cpo**-category 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 10.8 (*rCBV* $\boldsymbol{\omega}$**Cpo**-pair). *An rCBV* $\boldsymbol{\omega}$**Cpo***-pair is a CBV pair* $(\mathcal{V}, \mathcal{T})$ *such that, denoting by* $J : \mathcal{V} \to C$ *the corresponding universal Kleisli* $\mathcal{V}$*-functor,*

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

[rω.1] $\mathcal{V}$ *is a cocomplete* **ωCpo***-cartesian closed category*[18];

[rω.2] the unit of $\mathcal{T}$ *is pointwise a full morphism (hence,* J *is a locally full* **ωCpo***-functor);*

[rω.3] C *has an ep-zero object* $\mathfrak{O} = J(0)$*, where* 0 *is initial in* $\mathcal{V}$*;*

[rω.4] whenever $u : J(A) \overset{\hookrightarrow}{\leftarrow} J(B)$ *is an ep-pair in* C*, there is one morphism* $\hat{u} : A \to B$ *in* $\mathcal{V}$ *such that* $J(\hat{u}) = u^e$*.*

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

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

10.6.1 rCBV **ωCpo***-pairs are rCBV models*

Let $(\mathcal{V}, \mathcal{T})$ be an *rCBV* **ωCpo**-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 (69) be an n-variable $(\mathcal{V}, \mathcal{T})$-parametric type. For each $A \in \left(\mathcal{V}^{\mathrm{op}} \times \mathcal{V}\right)^{n-1}$, 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 (82) in 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 (81).

$$\begin{aligned} a_0 &\overset{\text{def}}{=} \left(\iota^e : \mathfrak{O} \to E^A_{\mathcal{C}}(\mathfrak{O}, \mathfrak{O}), \iota^p : E^A_{\mathcal{C}}(\mathfrak{O}, \mathfrak{O}) \to \mathfrak{O}\right) \\ a_{n+1} &\overset{\text{def}}{=} \left(E^A_{\mathcal{C}}\left(a^p_n, a^e_n\right), E^A_{\mathcal{C}}\left(a^e_n, a^p_n\right)\right) \end{aligned} \tag{81}$$

$$\mathfrak{O} \xrightarrow{a^e_0} \mathfrak{A}_1 \xrightarrow{a^e_1} \mathfrak{A}_2 \xrightarrow{a^e_2} \mathfrak{A}_3 \xrightarrow{a^e_3} \cdots \tag{82}$$

$$\mathfrak{O} \xleftarrow{a^p_0} \mathfrak{A}_1 \xleftarrow{a^p_1} \mathfrak{A}_2 \xleftarrow{a^p_2} \mathfrak{A}_3 \xleftarrow{a_3} \cdots \tag{83}$$

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 Definition 10.8. Since $\mathcal{V}$ has **ωCpo**-colimits, we conclude that the conical **ωCpo**-colimit of $\hat{\mathcal{E}}^E_A$ exists and is preserved by J (being an **ωCpo**-left adjoint) – hence, $\mathcal{E}^E_A$ has a conical **ωCpo**-colimit in C as well.

We recall the following variation on Smyth and Plotkin (1982)'s celebrated *limit-colimit coincidence* result.

Lemma 10.9 (Limit-colimit coincidence, à la Smyth and Plotkin 1982). *For any* ω*-chain* $(a^e_n \dashv a^p_n)_{n\in\mathbb{N}}$ *of ep-pairs in an* **ωCpo***-category* C*, any* **ωCpo***-colimiting cocone on* $(a^e_n)_{n\in\mathbb{N}}$ *consists of embeddings and the corresponding projections form an* **ωCpo***-limiting cone on* $(a^p_n)_{n\in\mathbb{N}}$*.*

Since (82) is the chain of embeddings of a chain of ep-pairs, the **ωCpo**-colimit of these embeddings coincides with the **ωCpo**-limit of the associated chain $\left(a^p_n\right)_{n\in\mathbb{N}}$ of projections (84), denoted herein by $\mathcal{P}^E_A$. Such a *bilimit* of ep-pairs is absolute in the sense that any **ωCpo**-functor $H : C \to C'$ preserves the conical **ωCpo**-colimit (and **ωCpo**-limit) of $\mathcal{E}^E_A$ (respectively, $\mathcal{P}^E_A$).

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Since the conical $\boldsymbol{\omega}\mathbf{Cpo}$-colimit of $\mathcal{E}^E_A$ is absolute, the diagram (69) commutes, and J strictly preserves $\boldsymbol{\omega}\mathbf{Cpo}$-colimits, we have the invertible morphism (85) given by the composition of the respective canonical comparison morphisms.

$$J \circ E^A_{\mathcal{V}}\left(\mathrm{colim}\left(\hat{\mathcal{E}^E_A}\right), \mathrm{colim}\left(\hat{\mathcal{E}^E_A}\right)\right) \xrightarrow{\cong} E^A_C\left(\mathbf{colim}\left(\mathcal{E}^E_A\right), \mathbf{colim}\left(\mathcal{E}^E_A\right)\right) \xrightarrow{\cong} \mathbf{colim}\left(\mathcal{E}^E_A\right) \xrightarrow{\cong} J\mathrm{colim}\left(\hat{\mathcal{E}^E_A}\right) \tag{84}$$

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 $\mathcal{E}^E_f : \mathcal{E}^E_A \to \mathcal{E}^E_B$. 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 $A \mapsto \hat{\mathcal{E}^E_A}$ also extends to a $\mathcal{V}$-*functor* $\hat{\mathcal{E}^E}$ from $\left(\mathcal{V}^{\mathrm{op}} \times \mathcal{V}\right)^{n-1}$ 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:

$$\nu_\omega E = (\nu_\omega E_{\mathcal{V}}, \nu_\omega E_{\mathcal{C}}) \stackrel{\mathrm{def}}{=} \left(\mathrm{colim} \circ \hat{\mathcal{E}^E}, \mathrm{colim} \circ \mathcal{E}^E\right) \tag{85}$$

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 $\omega\mathsf{roll}^E_A$ in $\mathcal{V}$ such that $J\left(\omega\mathsf{roll}^E_A\right)$ *is equal to* (85). The morphisms $\omega\mathsf{roll}^E = \left(\omega\mathsf{roll}^E_A\right)_{A \in (\mathcal{V}^{\mathrm{op}} \times \mathcal{V})^{n-1}}$ gives a $\mathcal{V}$-natural transformation $E_{\mathcal{V}}\left(\mathrm{id}, \nu_\omega E^{\mathrm{op}}_{\mathcal{V}}, \nu_\omega E_{\mathcal{V}}\right) \to \nu_\omega E_{\mathcal{V}}$ such that $J\left(\omega\mathsf{roll}^E\right)$ is invertible. Therefore, $\underline{\mathsf{roll}}_\omega \stackrel{\mathrm{def}}{=} \left(\omega\mathsf{roll}^E\right)_{E \in \mathsf{Param}(\mathcal{V},\mathcal{T})}$ is a rolling for ν_ω, and we can define the (free) type recursion $\underline{\nu}_\omega \stackrel{\mathrm{def}}{=} \left(\nu_\omega, \underline{\mathsf{roll}}_\omega\right)$.

Lemma 10.10 (Underlying *rCBV* model). *There is a forgetful functor* $\mathcal{U}_{r\mathcal{BV}} : \boldsymbol{\omega}\mathbf{CPO}\text{-}\mathfrak{C}_{r\mathcal{BV}} \to \mathfrak{C}_{\mathcal{RBV}}$ *defined by* $\mathcal{U}_{r\mathcal{BV}}(\mathcal{V}, \mathcal{T}) = \left(\mathcal{V}, \mathcal{T}, \underline{\nu}_\omega\right)$ *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 conditions of a *rCBV* model morphism described in Definition 10.3. □

Remark 10.11. The product of *rCBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pairs is computed as expected: $(\mathcal{V}_0, \mathcal{T}_0) \times (\mathcal{V}_1, \mathcal{T}_1) \cong (\mathcal{V}_0 \times \mathcal{V}_1, \mathcal{T}_0 \times \mathcal{T}_1)$. Moreover, it is clear that $\mathcal{U}_{r\mathcal{BV}}$ preserves finite products.

10.7 Concrete semantics

The *CBV* pair $\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$ as in Section 7.1 clearly satisfies the conditions of Definition 10.8, and hence, it is also an *rCBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair. By Proposition 10.5, for each $k \in \mathbb{N} \cup \{\infty\}$, we have unique *rCBV* model morphisms (87) and (88) respecting the assignments of Fig. 12 and (33). In other words, following Remark 10.6, we have only one extension of the semantics (30) and (32) to the respective languages with recursive types.

$$[\![-]\!] : \left(\mathbf{Syn}^{\mathsf{R}}_V, \mathbf{Syn}^{\mathsf{R}}_S, \underline{\nu}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right) \tag{86}$$

$$[\![-]\!]_k : \left(\mathbf{Syn}^{\mathsf{Rtr}}_V, \mathbf{Syn}^{\mathsf{Rtr}}_S, \underline{\nu}^{\mathbf{tr}}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}\left(\boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right). \tag{87}$$

Moreover, by Remark 10.11, we have that the product $\left(\boldsymbol{\omega}\mathbf{Cpo} \times \boldsymbol{\omega}\mathbf{Cpo}, (-)_\perp\right)$ as in Section 7.1 is an *rCBV* $\boldsymbol{\omega}\mathbf{Cpo}$-pair.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

*10.8 Subscone for rCBV ω**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* ***ω*Cpo**-pair $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}(-)_{\perp}\right)$ as in Proposition 9.3 yields an *rCBV* ***ω*Cpo**-pair. In order to do that, we rely on Theorem 10.13 about lifting the *rCBV* ***ω*Cpo**-pair structure.

Definition 10.12 (Impurity preserving/purity reflecting). *Let $(\mathcal{V},\mathcal{T})$ and $\left(\mathcal{V}',\mathcal{T}'\right)$ 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 10.13. *Let $\left(\mathcal{V}',\mathcal{T}'\right)$ be an rCBV* ***ω*Cpo**-*pair and $(\mathcal{V},\mathcal{T})$ a CBV pair such that $\mathcal{V}$ is a cocomplete* ***ω*Cpo**-*cartesian closed category and $T(0)$ is terminal.*

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

Proof We prove that $(\mathcal{V},\mathcal{T})$ yields an *rCBV* ***ω*Cpo**-pair. By hypothesis, $(\mathcal{V},\mathcal{T})$ satisfies (rω.1). We prove the remaining conditions of Definition 10.8 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 (89) of unique morphisms in $\mathcal{C}$. Since $\overline{H}$ preserves initial objects and $\left(\mathcal{V}',\mathcal{T}'\right)$ is an *rCBV* ***ω*Cpo**-pair, we have that (90) is the ep-pair of the unique morphisms. Finally, since $\overline{H}$ is a locally full ***ω*Cpo**-functor, it reflects ep-pairs, and hence, (89) is an ep-pair.

$$\left(\iota_A : J(0) \to A, \iota^A : A \to J(0)\right) \tag{88}$$

$$\left(\overline{H}(\iota_A), \overline{H}\left(\iota^A\right) : \overline{H}(A) \to \mathfrak{O}\right) \tag{89}$$

(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 $\left(\mathcal{V}',\mathcal{T}'\right)$ is an *rCBV* ***ω*Cpo**-pair, there is one morphism $\overline{H}\hat{(u)} : H(A) \to H(B)$ in $\mathcal{V}'$ such that $J'\left(\overline{H}\hat{(u)}\right) = \overline{H}(u^e)$. Since the *CBV* pair morphism $H : (\mathcal{V},\mathcal{T}) \to \mathcal{U}_{\mathrm{rp}}\left(\mathcal{V}',\mathcal{T}'\right)$ is impurity preserving, we conclude that there is $\hat{u} : A \to B$ such that $J\left(\hat{u}\right) = u^e$. □

As a consequence, in the setting of subscones satisfying Assumption 8.3, we get:

Theorem 10.14. *Let $(\mathcal{V},\mathcal{T})$ be an rCBV* ***ω*Cpo**-*pair and (91) the forgetful* ***ω*Cpo**-*functor coming from a pair $(G : \mathcal{V} \to \mathcal{D}, \mathfrak{T}_{sub})$ satisfying Assumption 8.3.*

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 (91) such that (c.1) *and* (c.2) *hold, then $\left(\mathbf{Sub}(\mathcal{D} \downarrow G), \overline{\mathcal{T}}\right)$ is an rCBV* ***ω*Cpo**-*pair and $\underline{\mathcal{L}}$ yields an rCBV* ***ω*Cpo**-*pair morphism (92).*

c.1 *$\overline{T}$ takes the initial to the terminal object;*

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

$$\underline{\mathcal{L}} : \mathbf{Sub}(\mathcal{D} \downarrow G) \to \mathcal{V} \tag{90}$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\left(\mathbf{Sub}\left(\mathcal{D}\downarrow G\right),\overline{\mathcal{T}}\right)\ \rightarrow\ \left(\mathcal{V},\mathcal{T}\right) \tag{91}$$

$$\begin{array}{ccc} D & \longrightarrow & \underline{\overline{\mathcal{T}}\left(D,C,j\right)} \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle \underline{\overline{\mathcal{T}}j}} \\ G(C) & \xrightarrow[G(\eta_C)]{} & G(T'(C)) \end{array} \tag{92}$$

Proof By Corollary 8.5, **Sub** $(\mathcal{D}\downarrow G)$ is cocomplete $\boldsymbol{\omega}$**Cpo**-cartesian closed. Moreover, $\underline{\mathcal{L}}$ is locally full, strict $\boldsymbol{\omega}$**Cpo**-cartesian closed, and $\boldsymbol{\omega}$**Cpo**-colimit preserving by Theorem 8.6. 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 (92).

(c.2) implies that the *CBV* pair morphism (92) is purity reflecting. Assuming (c.1), this implies that $\left(\mathbf{Sub}\left(\mathcal{D}\downarrow G\right),\overline{\mathcal{T}}\right)$ is indeed an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair morphism and $\underline{\mathcal{L}}$ yields an (92) is an *rCBV* $\boldsymbol{\omega}$**Cpo**-pair morphism by Theorem 10.13. □

In the particular case of interest, we conclude:

Proposition 10.15. *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* $\boldsymbol{\omega}$**Cpo***-pair. Moreover,* $\underline{\mathcal{L}}_{n,k}:\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right)\rightarrow\boldsymbol{\omega}\mathbf{Cpo}\times\boldsymbol{\omega}\mathbf{Cpo}$ *yields an rCBV* $\boldsymbol{\omega}$**Cpo***-pair morphism*

$$\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right),\mathcal{P}_{n,k}\left(-\right)_{\perp}\right)\ \rightarrow\ \left(\boldsymbol{\omega}\mathbf{Cpo}\times\boldsymbol{\omega}\mathbf{Cpo},\left(-\right)_{\perp}\right). \tag{93}$$

Proof In fact, we already know that $\underline{\mathcal{L}}_{n,k}$ comes from a pair that satisfies Assumption 8.3. Moreover, $\left(\boldsymbol{\omega}\mathbf{Cpo}\times\boldsymbol{\omega}\mathbf{Cpo},\left(-\right)_{\perp}\right)$ is an *rCBV* $\boldsymbol{\omega}$**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 10.14. □

By Proposition 10.15 and Lemma 10.10, we get:

Corollary 10.16. $\underline{\mathcal{L}}_{n,k}$ *yields an rCBV model morphism*

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

10.9 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 Definition 7.7). 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

$$\overline{[\![-]\!]}_{n,k}:\left(\mathbf{Syn}_V^{\mathsf{R}},\mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}},\underline{\nu}_{\mathbf{Syn}}\right)\rightarrow\mathcal{U}_{r\mathcal{B}\mathcal{V}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right),\mathcal{P}_{n,k}\left(-\right)_{\perp}\right) \tag{94}$$

that is consistent with the assignment given by (47), (52), (54), and (53).

Lemma 10.17. *For any* $(n,k)\in\mathbb{N}\times(\mathbb{N}\cup\{\infty\})$, *Diag.* (96) *commutes.*

$$\begin{array}{ccc} \left(\mathbf{Syn}_V^{\mathsf{R}},\mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}},\underline{\nu}_{\mathbf{Syn}}\right) & \xrightarrow{(\mathrm{id},\mathbb{D})} & \left(\mathbf{Syn}_V^{\mathsf{R}},\mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}},\underline{\nu}_{\mathbf{Syn}}\right)\times\left(\mathbf{Syn}_V^{\mathbf{Rtr}},\mathbf{Syn}_{\mathcal{S}}^{\mathbf{Rtr}},\underline{\nu}_{\mathbf{Syn}}^{\mathbf{tr}}\right) \\ {\scriptstyle \overline{[\![-]\!]}_{n,k}}\downarrow & & \downarrow{\scriptstyle [\![-]\!]\times[\![-]\!]_k} \\ \mathcal{U}_{r\mathcal{B}\mathcal{V}}\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo}\downarrow G_{n,k}\right),\mathcal{P}_{n,k}\left(-\right)_{\perp}\right) & \xrightarrow[\mathcal{U}_{r\mathcal{B}\mathcal{V}}\left(\underline{\mathcal{L}}_{n,k}\right)]{} & \mathcal{U}_{r\mathcal{B}\mathcal{V}}\left(\boldsymbol{\omega}\mathbf{Cpo}\times\boldsymbol{\omega}\mathbf{Cpo},\left(-\right)_{\perp}\right) \end{array} \tag{95}$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Proof Both $(\llbracket - \rrbracket \times \llbracket - \rrbracket_k) \circ (\mathrm{id} \times \mathbb{ID})$ and $\mathcal{U}_{r\mathcal{BV}}\left(\underline{\mathcal{L}}_{n,k}\right) \circ \overline{\llbracket - \rrbracket}_{n,k}$ 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 (48), (49), and (50). 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. (96) indeed commutes. □

10.10 AD correctness theorem for non-recursive data types

The correctness theorem for non-recursive data types (i.e., types formed from **real**, products, and coproducts) follows from Lemma 10.17 and Corollary 9.7. That is to say, we have:

Theorem 10.18. *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* $\llbracket t \rrbracket : \coprod_{r \in \mathfrak{L}} \mathbb{R}^{s_r} \to \left(\coprod_{j \in L} \mathbb{R}^{l_j}\right)_{\perp}$ *is differentiable and, for any* $k \in (\mathbb{N} \cup \{\infty\})$, $\llbracket \mathbb{ID}\,(t) \rrbracket_k = \mathfrak{d}^k\,(\llbracket t \rrbracket)$.

10.11 AD on recursive data types

The logical relations argument we presented provides us with an easy way to compute the logical relations of general recursive types: namely, since $\left(\mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right), \mathcal{P}_{n,k}\,(-)_{\perp}\right)$ 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 of Theorem 10.18 to any *recursive data type*. By that, we mean any type τ built from the grammar

$$\tau, \sigma ::= \alpha \mid \mathbf{real} \mid \mathbf{0} \mid \mathbf{1} \mid \tau \times \sigma \mid \tau \sqcup \sigma \mid \mu\alpha.\tau,$$

that is, any type not involving function types.

We can define these (recursive) data type more formally as follows. We denote by $\mathbf{Syn}_C^{\mathsf{R}}$ the Kleisli $\mathbf{Syn}_V^{\mathsf{R}}$-category associated with $\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}\right)$. Moreover, we, respectively, denote by (97) and (98) the coproduct, product, and n-diagonal functors.

$$\sqcup, \times \; : \; \mathbf{Syn}_V^{\mathsf{R}} \times \mathbf{Syn}_V^{\mathsf{R}} \to \mathbf{Syn}_V^{\mathsf{R}} \tag{96}$$

$$\mathrm{diag}_n \; : \; \left(\mathbf{Syn}_V^{\mathsf{R}}\right)^{\mathrm{op}} \times \mathbf{Syn}_V^{\mathsf{R}} \to \left(\left(\mathbf{Syn}_V^{\mathsf{R}}\right)^{\mathrm{op}} \times \mathbf{Syn}_V^{\mathsf{R}}\right)^n \tag{97}$$

Definition 10.19. *Let* $R, I, O : \left(\mathbf{Syn}_V^{\mathsf{R}}\right)^{\mathrm{op}} \times \mathbf{Syn}_V^{\mathsf{R}} \to \mathbf{Syn}_V^{\mathsf{R}}$ *be the constant functors which are, respectively, equal to* **real**, $\mathtt{1}$ *and* $\mathtt{0}$. *We define the set* $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$ *inductively by* (D1), (D2), *and* (D3).

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

(D2) *For each* $n \in \mathbb{N}^*$, *if the functors* (99) *belong to* $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$, *then the functors* (100) *and* (101) *are in* $\mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

We define the set $\mathsf{Param}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)$ *of parametric data types by* (102).

$$G, G' \,:\, \left(\left(\mathbf{Syn}_V^{\mathsf{R}}\right)^{\mathrm{op}} \times \mathbf{Syn}_V^{\mathsf{R}}\right)^n \to \mathbf{Syn}_V^{\mathsf{R}} \tag{98}$$

$$G \circ \mathrm{diag}_n \,:\, \left(\mathbf{Syn}_V^{\mathsf{R}}\right)^{\mathrm{op}} \times \mathbf{Syn}_V^{\mathsf{R}} \to \mathbf{Syn}_V^{\mathsf{R}} \tag{99}$$

$$\times \circ \left(G \times G'\right), \sqcup \circ \left(G \times G'\right) \,:\, \left(\left(\mathbf{Syn}_V^{\mathsf{R}}\right)^{\mathrm{op}} \times \mathbf{Syn}_V^{\mathsf{R}}\right)^{2n} \to \mathbf{Syn}_V^{\mathsf{R}} \tag{100}$$

$$\mathsf{Param}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) := \left\{E \in \mathsf{Param}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}\right) : E_{\mathbf{Syn}_V^{\mathsf{R}}} \in \mathfrak{P}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right)\right\} \tag{101}$$

All such (recursive) data types are, up to isomorphism, of a particularly simple form: a sum of products.

Proposition 10.20. *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* $\left(\mathsf{m}_{(j,\mathsf{T})}\right)_{(j,\mathsf{T}) \in (\mathbb{I}_n \cup \{0\}) \times \mathsf{Tree}}$ *such that, for any rCBV model morphism* $H : \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}(\mathcal{V}, \mathcal{T})$ *and any* H*-compatible pair* (E, F), *we have that* (104) *holds, where the isomorphism* $\cong$ *is induced by coprojections and projections*[19].

$$H(\tau) = \coprod_{j \in L} H(\mathbf{real})^{l_j} \tag{102}$$

$$F_{\mathcal{V}}\left(W_j, Y_j\right)_{j \in \mathbb{I}_n} \cong \coprod_{\mathsf{T} \in \mathsf{Tree}} \left(H(\mathbf{real})^{\mathsf{m}_{(0,\mathsf{T})}} \times \prod_{j=1}^{n} Y_j^{\mathsf{m}_{(j,\mathsf{T})}}\right) \tag{103}$$

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* (103) *holds for any rCBV model morphism* $H : \left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}, \underline{\nu}_{\mathbf{Syn}}\right) \to \mathcal{U}_{r\mathcal{BV}}(\mathcal{V}, \mathcal{T})$.

Proof The result follows from induction. The nontrivial part is a consequence of the following.

Let $\left(\tilde{E}, \tilde{F}\right) \in \mathsf{Param}^{\mathfrak{d}}\left(\mathbf{Syn}_V^{\mathsf{R}}, \mathbf{Syn}_{\mathcal{S}}^{\mathsf{R}}\right) \times \mathsf{Param}(\mathcal{U}_{r\mathcal{BV}}(\mathcal{V}, \mathcal{T}))$ be an H-compatible pair of $(n+1)$-variable parametric types where $\tilde{F}_{\mathcal{V}}$ is given by (105) 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 Equation (104). 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 (104).

(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 $\mathsf{T} \in \mathsf{Tree}^{\mathsf{s}_{(n+1,r)}}$, the pair (T, r) is in Tree.

(II) We inductively define the family $\left(\mathsf{m}_{(j,\mathsf{T})}\right)_{(j,\mathsf{T}) \in (\mathbb{I}_n \cup \{0\}) \times \mathsf{Tree}}$ 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 $\mathsf{T} = (\mathsf{T}_i)_{i \in \mathbb{I}_{\mathsf{s}_{(n+1,r)}}} \in \mathsf{Tree}^{\mathsf{s}_{(n+1,r)}}$, we define $\mathsf{m}_{(j,(\mathsf{T},r))}$ by (106) for each j.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\tilde{F}_{\mathcal{V}}\left(W_i, Y_i\right)_{i\in\mathbb{I}_{n+1}} = \coprod_{r\in\mathfrak{L}}\left(H\left(\mathbf{real}\right)^{\mathsf{s}_{(0,r)}} \times \prod_{i=1}^{n+1} Y_i^{\mathsf{s}_{(i,r)}}\right) \tag{104}$$

$$\mathsf{m}_{(j,(\mathrm{T},r))} = \mathsf{s}_{(j,r)} + \sum_{i=1}^{\mathsf{s}_{(n+1,r)}} \mathsf{m}_{(j,\mathrm{T}_i)} \tag{105}$$

Let $X = \left(W_i, Y_i\right)_{i\in\mathbb{I}_n} \in \left(\mathcal{V}^{\mathrm{op}} \times \mathcal{V}\right)^n$, $\mathfrak{F}_X := \tilde{F}^X_{\mathcal{V}}\left(0, -\right)$ and ι the obvious unique morphism. The colimit of (107) is isomorphic to (108). Hence, by the definition of the *fddt* operator ν_ω of $\mathcal{U}_{r\mathcal{B}\mathcal{V}}\left(\mathcal{V}, \mathcal{T}\right) = \left(\mathcal{V}, \mathcal{T}, \underline{\nu}_\omega\right)$, $\nu_\omega\tilde{F}_{\mathcal{V}}$ is given by the formula given in (104). This completes the proof.

$$0 \xrightarrow{\iota} \mathfrak{F}_X\left(0\right) \xrightarrow{\mathfrak{F}_X(\iota)} \mathfrak{F}^2_X\left(0\right) \xrightarrow{\mathfrak{F}^2_X(\iota)} \mathfrak{F}^3_X\left(0\right) \to \cdots \tag{106}$$

$$\coprod_{\mathrm{T}\in\mathsf{Tree}}\left(H\left(\mathbf{real}\right)^{\mathsf{m}_{(0,\mathrm{T})}} \times \prod_{j=1}^{n} Y_j^{\mathsf{m}_{(j,\mathrm{T})}}\right) \tag{107}$$

Finally, if $\tau \in \mathbf{Syn}^{\mathsf{R}}_V$ corresponds to a data type τ, then the constant parametric type $\underline{\tau}$ equal to τ is an $\left(\mathbf{Syn}^{\mathsf{R}}_V, \mathbf{Syn}^{\mathsf{R}}_{\mathcal{S}}\right)$-parametric data type of degree 1. Hence, denoting by $\underline{H\tau}$ the constant parametric type equal to $H\left(\tau\right)$, since $\left(\underline{\tau}, \underline{H\tau}\right)$ is H-compatible, we conclude that (104) holds for some $\left(l_j\right)_{j\in L}$ where L is countable. □

In particular, for any nonparametric (meaning: 0-variable parametric) recursive data type R, we have the following:

$$\overline{[\![R]\!]}_{n,k} = \coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{n,k} \tag{108}$$

$$[\![R]\!] = \coprod_{j\in L} \mathbb{R}^{l_j}. \tag{109}$$

This lets us strengthen our correctness theorem to apply also to programs between recursive data types:

Proposition 10.21. *Let $t : \tau \to \sigma$ be a morphism in $\mathbf{Syn}^{\mathsf{R}}_V$. If τ and σ 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 $k \in (\mathbb{N}\cup\{\infty\})$, $[\![\mathbb{ID}\left(t\right)]\!]_k = \mathfrak{d}^k\left([\![t]\!]\right)$.*

Proof First of all, indeed, by Proposition 10.20, we have that there are countable families $(s_r)_{r\in\mathfrak{L}}$ and $\left(l_j\right)_{j\in L}$ such that

$$\overline{[\![t]\!]}_{s_i,k} : \coprod_{r\in\mathfrak{L}} \overline{[\![\mathbf{real}]\!]}^{s_r}_{s_i,k} \to \mathcal{P}_{s_i,k}\left(\coprod_{j\in L} \overline{[\![\mathbf{real}]\!]}^{l_j}_{s_i,k}\right)_\perp \tag{110}$$

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 $k \in \mathbb{N}\cup\{\infty\}$.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

By the commutativity of (96) for any $(s_i, k) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$, we get that the pair $(\llbracket t \rrbracket, \llbracket \mathbb{ID}(t) \rrbracket_k)$ defines the morphism (111) for each $i \in \mathfrak{L}$. By Corollary 9.7, this implies that $\llbracket t \rrbracket$ is differentiable and $\llbracket \mathbb{ID}\,(t)\,\rrbracket_k = \mathfrak{d}^k\,(\llbracket t \rrbracket)$. □

Finally, as a consequence, we get:

Theorem 10.22. *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* $\llbracket t \rrbracket$ *is differentiable and, moreover,*

$$\llbracket \mathcal{D}\,(t) \rrbracket_k = \mathfrak{d}^k\,(\llbracket t \rrbracket) \tag{111}$$

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

Following the considerations of Section 9.6 and 9.7, it follows from Theorem 10.18 that $\mathcal{D}$ as defined in Section 10.4 *correctly* provides us with forward and reverse AD transformations for data types.

10.12 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 $\boldsymbol{\mu}\alpha.\mathbf{1} \sqcup + \tau \times \alpha$. It should be noted that, if $x : \boldsymbol{\mu}\alpha.\mathbf{1} \sqcup \tau \times \alpha \vdash t : \boldsymbol{\mu}\alpha.\mathbf{1} \sqcup \tau \times \beta$, we have that

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

By Theorem 10.22, if τ and σ are data types, we get that $\mathfrak{d}^k\,(\llbracket t \rrbracket)$ (as defined in (27)) is equal to $\llbracket \mathcal{D}\,(t) \rrbracket_k$. Therefore, Theorem 10.22 already encompasses the correctness for arrays (of data types).

11. Almost Everywhere Correct AD

Here, we show how some of the arguments of Huot et al. (2023) about almost everywhere differentiability can be accommodated in our framework, by making use of a minor variation of our chosen logical relations over ω-cpos. The resulting arguments use plain logical relations over ω-cpos and do not rely on sheaf-structure. They are also a bit more general, as they apply to languages with coproduct and recursive types.

The central notion is Lee et al. (2020)'s concept of functions that are PAP. We recall some of the required notions to talk about PAP functions first.

Definition 11.1 (Analytic function). *A function* $f : U \to V$*, for* $U \subseteq \mathbb{R}^n$ *and* $V \subseteq \mathbb{R}^m$*, is analytic if, for all* $x \in U$*, its Taylor series converges pointwise to* f *on an open neighborhood of* x*.*

Definition 11.2 ((c)-Analytic set). *A subset* $A \subseteq \mathbb{R}^n$ *is called analytic if there exist analytic functions* $g_1, \ldots, g_m : U \to \mathbb{R}$ *defined on an open neighborhood* U *of* A*, such that*

$$A = \{x \in U \mid g_i(x) \geq 0 \text{ for } 1 \leq i \leq m\}.$$

A subset $A \subseteq \mathbb{R}^n$ *is called c-analytic if it is the countable union of analytic subsets.*

As noted by Huot et al. (2023), we can equivalently define a c-analytic set as a countable *disjoint* union of analytic subsets.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Definition 11.3 (PAP function). *A function $f : U \to V$, for $U \subseteq \mathbb{R}^n$ and $V \subseteq \mathbb{R}^m$, is called piecewise analytic under analytic partition (PAP) if it has a PAP representation in the sense of a countable family $\{(A_i, U_i, f_i)\}_{i\in I}$ such that:*

- *the sets A_i are analytic and form a partition of U;*
- *each $f_i : U_i \to V$ is an analytic function defined on an open neighborhood U_i of A_i;*
- *$f_i|_{A_i} = f|_{A_i}$ in the sense that $f_i(x) = f(x)$ for all $x \in A_i$.*

A crucial observation by Lee et al. (2020) is that PAP functions are closed under composition. As noted by Huot et al. (2023), a subset $A \subseteq \mathbb{R}^n$ is c-analytic if and only if the inclusion $A \hookrightarrow \mathbb{R}^n$ is a PAP function.

We consider the following notion of partial PAP function.

Definition 11.4 (Partial PAP function). *We call a partial function $f : U \rightharpoonup V$ a partial PAP function if its domain of definition is c-analytic and it restricts to a (total) PAP function on its domain.*

As noted by Huot et al. (2023), such partial PAP functions are closed under composition.

Definition 11.5 (Intensional derivative). *Each particular PAP representation $\{(A_i, U_i, f_i)\}_{i\in I}$ of a PAP function f gives rise to a unique intensional derivative $\{(A_i, U_i, Df_i)\}_{i\in I}$, where we write Df_i for the (standard) derivative of f_i, such that $Df_i = Df$ on A_i.*

A given PAP function may therefore have several distinct intensional derivatives, arising from the different PAP representations. However, Lee et al. (2020) show that such PAP functions f are differentiable almost everywhere and that each intensional derivative corresponds almost everywhere with the (standard) derivative of f.

Next, we redefine our logical relations for **real** and monadic types from Sections 9.3 and 9.2. First, we redefine

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

Second, we denote by $\mathfrak{O}_n$ the set of countable families $\{(A_i, U_i)\}_{i\in I}$ of pairs of analytic subsets $A_i \subseteq \mathbb{R}^n$ and open neighborhoods U_i of A_i in, such that all A_i are pair-wise disjoint and $\bigsqcup_{i\in I} A_i \neq \emptyset, \mathbb{R}^n$. Then, for each $\{(A_i, U_i)\}_{i\in I} \in \mathfrak{O}_n$, we redefine

$$\begin{aligned}\mathsf{Diff}_{(\{(A_i,U_i)\}_{i\in I},n,k)} \stackrel{\text{def}}{=} \bigsqcup_{i\in I} \left(\left\{\left(g : \mathbb{R}^n \to U_i, \mathfrak{D}^k g\right) : g \text{ is analytic}\right\}, \left(U_i, \phi_{n,k}\left(U_i \times \left(\mathbb{R}^k\right)^n\right)\right), \text{incl.}\right)\\ \in \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right).\end{aligned}$$

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

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

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

$$\{\perp\} \sqcup D \sqcup \left(\coprod_{\{(A_i,U_i)\}_{i\in I}\in\mathfrak{O}_n} \mathbf{Sub}\left(\boldsymbol{\omega}\mathbf{Cpo} \downarrow G_{n,k}\right)\left(\mathsf{Diff}_{(\{(A_i,U_i)\}_{i\in I},n,k)}, \left(D, \left(C, C'\right), j\right)\right)\right) / \sim,$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

where we identify $[(\gamma_i, \gamma_i') \mid i \in I] \sim [(\overline{\gamma}_j, \overline{\gamma}_j') \mid j \in J]$ if their domains of definition coincide ($\bigsqcup_{i\in I} A_i = \bigsqcup_{j\in J} \overline{A}_j$) and they define the same function on this domain. To be more formal, we define the identification $[(\gamma_i, \gamma_i') \mid i \in I] \sim [(\overline{\gamma}_j, \overline{\gamma}'_j) \mid j \in J]$ if $\bigsqcup_{i\in I} A_i = \bigsqcup_{j\in J} \overline{A}_j$ and $[\gamma_i \circ \iota_i \mid i \in I] = [\overline{\gamma}_j \circ \bar{\iota}_j \mid j \in J]$ and $[\gamma_i' \circ \phi_{n,k} \circ (\iota_i \times \mathrm{id}_{(\mathbb{R}^k)^n}) \circ \phi_{n,k}^{-1} \mid i \in I] = [\overline{\gamma}_j' \circ \phi_{n,k} \circ (\bar{\iota}_j \times \mathrm{id}_{(\mathbb{R}^k)^n}) \circ \phi_{n,k}^{-1} \mid j \in J]$, where we use the inclusions $\iota_i : A_i \hookrightarrow U_i$ and $\bar{\iota}_j : \overline{A}_j \hookrightarrow \overline{U}_j$. The structure of the monad is defined entirely analogously to that in Section 9.2. Closure under suprema of ω-chains follows from (Huot et al. 2023, Corollary B.9). It is easy to see that the conditions of Theorem 10.14 are satisfied as before.

The rest of the development remains essentially unchanged, except for the minor modification that we work with (1) PAP functions rather than differentiable functions and (2) countable families of analytic subsets with open neighborhoods rather than open subsets.

If we spell out the resulting definitions for the logical relations (focusing on the k-semantics for $k = 1$), the result is as follows:

$$T^n_{\mathbf{real}} \stackrel{\text{def}}{=} \{(\gamma, \gamma') \mid \gamma \text{ is PAP and } \gamma' = (x, v) \mapsto (\gamma(x), \gamma''(x, v)) \text{ for an intensional derivative } \gamma'' \text{ of } \gamma\}$$

$$P^n_\tau \stackrel{\text{def}}{=} \Big\{(\gamma, \gamma') \mid \gamma^{-1}(\llbracket\tau\rrbracket) \times \mathbb{R}^n = \gamma'^{-1}(\llbracket\mathcal{D}(\tau)\rrbracket) \text{ and there exists a countable analytic partition } \{A_i \subseteq \mathbb{R}^n\}_{i\in I} \text{ of } \gamma'^{-1}(\llbracket\mathcal{D}(\tau)\rrbracket) \text{ and there exist open neighbourhoods } U_i \text{ of } A_i \text{ with functions } \gamma_i : U_i \to \llbracket\tau\rrbracket, \gamma_i' : U_i \times \mathbb{R}^n \to \llbracket\mathcal{D}(\tau)\rrbracket \text{ such that } \gamma|_{A_i} = \gamma_i|_{A_i} \text{ and } \gamma'|_{A_i\times\mathbb{R}^n} = \gamma_i'|_{A_i\times\mathbb{R}^n} \text{ and for all analytic } \delta : \mathbb{R}^n \to U_i \text{ we have that } (\gamma_i \circ \delta, (x, v) \mapsto (\gamma_i(\delta(x)), \gamma_i'(D\delta(x, v)))) \in T^n_\tau\Big\}.$$

We see that $P^n_{\mathbf{real}^m}$ precisely captures Huot et al. (2023)'s notion of partial PAP functions and their intensional derivatives, if we note that we can use (analytic) δ to define for any point $y \in \mathbb{R}^n$ an arbitrary small neighborhood: $x \mapsto \frac{x*\epsilon}{\sqrt{1+\|x\|^2}} + y$ is an analytic isomorphism between $\mathbb{R}^n$ and an ϵ-ball centered at y. We can show (by induction) that P^n_τ is closed under suprema of ω-chains using (Huot et al. 2023, Corollary B.9).

With these new definitions, our entire development goes through again. As long as we ensure that all our primitive operations denote partial PAP functions, we obtain versions of Theorem III.2 and Corollary III.3. of Huot et al. (2023) for a language that additionally includes recursive types, by using a plain logical relations argument over ω-cpos:

Theorem 11.6 (Almost everywhere differentiability). *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* $\llbracket t\rrbracket$ *is differentiable almost everywhere on its domain and, moreover,*

$$\llbracket\mathcal{D}(t)\rrbracket_k = \mathfrak{d}^k(\llbracket t\rrbracket) \tag{112}$$

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

Consequently, we obtain the correct derivative almost everywhere for any program t that terminates almost everywhere. Importantly, this result remains true if we change the semantics of **sign** t to be defined even for $t = 0$, as is done in Huot et al. (2023) and Mazza and Pagani (2021):

$$\llbracket\mathbf{sign}\ t\rrbracket \stackrel{\text{def}}{=} \begin{cases} \iota_1() & \text{if } \llbracket t\rrbracket \le 0 \\ \iota_2() & \text{otherwise} \end{cases}.$$

Indeed, this semantics is still logical relation respecting, thanks to our choice of lifting of the partiality monad to logical relations.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

12. Related Work

This is an improved version of the unpublished preprint Vákár (2020). 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 PL point of view, a lot of it focusing on functional formulations of AD and their correctness. Examples of such papers are Pearlmutter and Siskind (2008), Elliott (2018), Shaikhha et al. (2019), Brunel et al. (2020), Abadi and Plotkin (2020), Barthe et al. (2020), Lee et al. (2020), Huot et al. (2020), Mazza and Pagani (2021), Vákár (2021), Lucatelli Nunes (2022), Huot et al. (2021), Vákár and Smeding (2022), Krawiec et al. (2022), Smeding and Vákár (2023), and Huot et al. (2023). Of these papers, Pearlmutter and Siskind (2008), Abadi and Plotkin (2020), Lee et al. (2020), Mazza and Pagani (2021), Smeding and Vákár (2023),and Huot et al. (2023) are particularly relevant as they also consider AD of languages with partial features.

Here, Pearlmutter and Siskind (2008) consider an implementation that differentiates recursive programs and the implementation of Smeding and Vákár (2022) even differentiates code that uses recursive types. They do not give correctness proofs, however.

Abadi and Plotkin (2020) pioneer a notion of correctness that we use for most of this paper, where points of non-differentiability are essentially ignored by making a function undefined at such points. They use it to 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 a logical technique.

Lee et al. (2020) introduce a more ambitious notion of correctness in the sense of almost everywhere correct AD. Mazza and Pagani (2021) prove 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. Importantly, they also take care to prove almost everywhere correct differentiation for a language that supports conditionals on real numbers and primitives that can have points of non-differentiability. Their proof relies on operational semantic techniques. Huot et al. (2023) combine the ideas of Lee et al. (2020) with those of Vákár (2020) to give a denotational proof of almost everywhere correct AD for PCF, by using sheaves of logical relations. Section 11 of the present paper shows how their arguments can be reproduced without any sheaf-theoretic machinery, essentially by choosing a different lifting of the partiality monad to logical relations.

Barthe et al. (2020) have previously used (open) logical relations over the syntax, rather than semantics, to prove correctness of AD on total languages. It would be interesting to see whether and how their techniques could be adapted to languages with partial features. We suspect that the choice between logical relations over the syntax or semantics is mostly a matter of taste but that the extra (co)completeness properties that the semantics has can help, particularly when proving things about recursion and recursive types.

There is an independent line of inquiry into differential λ-calculus (Ehrhard and Regnier 2003) and differential categories (Blute et al. 2020; Cockett et al. 2020). A conceptual distinction with the work on AD is that differentiation tends to be a first-class construct (part of the language) in differential λ-calculus, rather than a code transformation in a metalanguage. Further, there is a stronger emphasis on the axioms that derivatives need to satisfy and less of a focus on recipes for computing derivatives. In this setting, differential restriction categories Cockett et al. (2012) gives a more abstract semantic study of the interaction between (forward) differentiation and partiality. We found that for our purposes, a concrete semantics in terms of ω-cpos sufficed, however.

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 data structures that we find in realistic ML-family languages.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Such AD for languages with expressive features such as recursion and user-defined data types has been called for by the machine learning community (Jeong et al. 2018; van Merrienboer et al. 2018). Previously, the subtlety of the interaction of AD and real conditionals had first been observed by Beck and Fischer (1994).

Our work gives a relatively simple denotational semantics for recursive types, which can be considered as an important special case of bilimit compact categories (Levy 2012). Bilimit compact categories are themselves, again, an important special case of the very general semantics of recursive types in terms of algebraically compact categories (Freyd 1991). 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 (Pitts 1996), which is quite general but very technical to use, in our experience, and step-indexed logical relations (Ahmed 2006), 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 PL literature on AD and recursion (and recursive types). In particular, we believe that it provides a simple, principled denotational explanation of how AD and expressive partial language features should interact. We plan to use it to generalize and prove correct the more advanced AD technique CHAD (Elliott 2018;Vákár 2021; Vákár and Smeding 2022; Lucatelli Nunes 2022; Kerjean and Pédrot 2022) when applied to languages with partial features.

Acknowledgments. 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 program under the Marie Skłodowska-Curie grant agreement no. 895827.

This research was supported through the program "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.

We are grateful to the anonymous reviewers for their helpful comments on this manuscript.

Notes

1 For this particular case of an iterative algorithm, it is actually possible and better (but more laborious!) to implement a custom derivative rather than differentiating through the while-loop (Margossian and Betancourt 2021).

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

3 Note that, in practice, Smeding and Vákár (2023) actually implement **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 Brunel et al. 2020) to achieve the correct computational complexity of reverse AD.

4 See Dubuc (1970, p. 60) for the classical enriched case. For the general case of monads in 2-categories, see Street (1972, p. 150) or, for instance, Lucatelli Nunes (2016, Section 3).

5 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 modeling of call-by-value languages in Levy et al. (2003)

6 In fact, it is locally presentable, hence complete and cocomplete, and its internal hom $X \Rightarrow Y$ is given by using the order $f \leq_{X\Rightarrow Y} g$ defined as $\forall x \in X. f(x) \leq_Y g(x)$ on the homset $\boldsymbol{\omega}\mathbf{Cpo}(X, Y)$. Its products $\sqcap_{i\in I} X_i$ carry the lexicographic order, and its coproducts $\bigsqcup_{i\in I} X_i$ have the disjoint union of the orders of all X_i, making elements in different components incomparable. See, for example, Vákár, Kammar, and Staton (2019) for more details.

7 $\mathbb{R}^\infty$ 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.

8 Goubault-Larrecq et al. (2002) develop some general methods for obtaining monad liftings to the scone and subscone. However, there are many choices for such monad liftings, and they need to be chosen depending on the specific application.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

9 That is, this **ωCpo**-functor is up to equivalence, the forgetful **ωCpo**-functor from the **ωCpo**-Eilenberg–Moore category of a (co)monad on $\mathcal{D} \times \mathcal{B}$. Dubuc (1970) gives a good introduction to the theory of enriched (co)monads.
10 In this work, we are only concerned with *conical* **ωCpo**-limits and colimits of functors $J : \mathcal{E} \to \mathcal{C}$ in the sense of $\mathcal{C}$-objects such that we have natural **ωCpo**-isomorphisms

$$\mathcal{C}(C, \lim J) \cong \mathbf{Cat}(\mathcal{E}, \mathcal{C})(\Delta_C, J) \qquad \text{and} \qquad \mathcal{C}(\operatorname{colim} J, C) \cong \mathbf{Cat}(\mathcal{E}, \mathcal{C})(J, \Delta_C),$$

where we write Δ_C for the constantly C functor. For the more general theory of (weighted) $\mathcal{V}$-(co)limits, we refer the reader to Kelly (1982).
11 See Dubuc (1968) for the original adjoint triangle theorem and Lucatelli Nunes (2016, Section 1) for the enriched version.
12 That is, all $\mathbf{Sub}\left(\mathcal{D} \downarrow G\right)\left((D, C, j), (D', C', h)\right) \to (\mathcal{D} \downarrow G)((D, C, j), (D', C', h))$ are isomorphisms of ω-cpos.
13 That is, $\mathbf{Sub}\left(\mathcal{D} \downarrow G\right) \to \mathcal{D} \downarrow G$ has an **ωCpo**-left adjoint.
14 That is, $\mathbf{Sub}\left(\mathcal{D} \downarrow G\right)$ is closed under isomorphisms in $\mathcal{D} \downarrow G$.
15 See Levy (2012, 4.2.2) or Vákár (2020, Section 8) for the general setting of bilimit compact expansions.
16 See, for instance, Kelly and Street (1972, Section 2) or Lucatelli Nunes (2016, 3.10) for adjunctions in 2-categories.
17 Recall that a *zero object* is an object that is both initial and terminal.
18 Because $\mathcal{V}$ is cartesian closed, any colimit in $\mathcal{V}$ is a conical $\mathcal{V}$-colimit (Kelly 1982). Because $\mathcal{V}$ is **ωCpo**-cartesian closed, any conical $\mathcal{V}$-colimit in $\mathcal{V}$ is, in particular, a conical **ωCpo**-colimit.
19 That is to say, it is just a reorganization of the involved coproducts and products.
20 This is a type theory for the Freyd category given by the Kleisli functor of the partiality monad. In the presence of the connectives we consider (in particular, function types), it is equivalent to Moggi's monadic metalanguage (Moggi 1991).

References

Abadi, M. and Plotkin, G. (2020). A simple differentiable programming language. In: *Proc. POPL 2020*, ACM.

Ahmed, A. J. (2006). Step-indexed syntactic logical relations for recursive and quantified types. In: Sestoft, P.(ed.) *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*, Springer, vol. **3924**, 69–83, Lecture Notes in Computer Science.

Barthe, G., Crubillé, R., Lago, U. D. and Gavazzo, F. (to appear). On the versatility of open logical relations: Continuity, automatic differentiation, and a containment theorem. In: *Proc. ESOP 2020*, Springer.

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

Betancourt, M. (2019). Double-pareto lognormal distribution in stan.

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

Blute, R., Cockett, J. R. B., Lemay, J. P. and Seely, R. A. G. (2020). Differential categories revisited. *Applied Categorical Structures* **28** (2) 171–235.

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

Carpenter, B., Hoffman, M., Brubaker, M., Lee, D., Li, P. and Betancourt, M. (2015). The stan math library: reverse-mode automatic differentiation in C++. arXiv preprint arXiv: 1509.07164.

Clementino, M. and Lucatelli Nunes, F. (2024). Lax comma 2-categories and admissible 2-functors. *Theory and Applications of Categories* **40** 180–226.

Cockett, J. R. B., Cruttwell, G. S. and Gallagher, J. D. (2012). Differential restriction categories. arXiv preprint arXiv: 1208.4068.

Cockett, J. R. B., Cruttwell, G. S. H., Gallagher, J., Lemay, J. P., MacAdam, B., Plotkin, G. D. and Pronk, D. (2020). Reverse derivative categories. In: *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, Fernández, M. and Muscholl, A. (eds.), vol. **152**.

Dubuc, E. (1968). Adjoint triangles. In: *Reports of the Midwest Category Seminar, II*, Berlin, Springer, 69–91.

Dubuc, E. (1970). *Kan Extensions in Enriched Category Theory*, Lecture Notes in Mathematics, vol. **145**, Berlin-New York, Springer-Verlag.

Ehrhard, T. and Regnier, L. (2003). The differential lambda-calculus. *Theoretical Computer Science* **309** (1-3) 1–41.

Elliott, C. (2018). The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages* **2** (ICFP) 70.

Fiore, M. and Plotkin, G. (1994). An axiomatisation of computationally adequate domain theoretic models of fpc. In: *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*, IEEE, 92–102.

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

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

Freyd, P. (1991). Algebraically complete categories. In: *Category Theory*, Springer, 95–104.
Goncharov, S., Rauch, C. and Schröder, L. (2015). Unguarded recursion on coinductive resumptions. *Electronic Notes in Theoretical Computer Science* **319** 183–198.
Goodrich, B. (2017). Conway-maxwell-poisson distribution.
Goubault-Larrecq, J., Lasota, S. and Nowak, D. (2002). Logical relations for monadic types. In: *International Workshop on Computer Science Logic*, Springer, 553–568.
Griewank, A. and Walther, A. (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, vol. **105**, SIAM.
Huot, M., Lew, A. K., Mansinghka, V. K. and Staton, S. (2023). ωpap spaces: Reasoning denotationally about higher-order, recursive probabilistic and differentiable programs. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, IEEE, 1–14.
Huot, M., Staton, S. and Vákár, M. (2020). Correctness of automatic differentiation via diffeologies and categorical gluing. Full version. arxiv:2001.02209.
Huot, M., Staton, S. and Vákár, M. (2021). Higher order automatic differentiation of higher order functions. CoRR, abs/2101.06757.
Jeong, E., Jeong, J. S., Kim, S., Yu, G.-I. and Chun, B.-G. (2018). Improving the expressiveness of deep learning frameworks with recursion. In: *Proceedings of the Thirteenth EuroSys Conference*, 1–13.
Kelly, G. M. (1982). *Basic Concepts of Enriched Category Theory*, vol. **64**, CUP Archive.
Kelly, G. M. and Street, R. (1974). Review of the elements of 2-categories. In: *Lecture Notes in Mathematics*, *Category Seminar (Proc. Sem., Sydney, 1972/1973)*, **420**, Springer, vol. 75–103.
Kerjean, M. and Pédrot, P.-M. (2022). ∂ is for Dialectica. working paper or preprint.
Krawiec, F., Jones, S. P., Krishnaswami, N., Ellis, T., Eisenberg, R. A. and Fitzgibbon, A. W. (2022). Provably correct, asymptotically efficient, higher-order reverse-mode automatic differentiation. *Proceedings of the ACM on Programming Languages* **6** (POPL) 1–30.
Lee, W., Yu, H., Rival, X. and Yang, H. (2020). On correctness of automatic differentiation for non-differentiable functions. In: *Advances in Neural Information Processing Systems, 33*, 6719–6730.
Levy, P. (2012). *Call-by-Push-Value: A Functional/Imperative Synthesis*, vol. **2**, Springer Science & Business Media.
Levy, P., Power, J. and Thielecke, H. (2003). Modelling environments in call-by-value programming languages. *Information and Computation* **185** (2) 182–210.
Lucatelli Nunes, F. (2016). On biadjoint triangles. *Theory and Applications of Categories* **31** 217–256. Paper No. 9.
Lucatelli Nunes, F. (2022). Semantic Factorization and Descent. *Applied Categorical Structures* **30** (6), 1393–1433.
Lucatelli Nunes, F. and Vákár, M. (2023). CHAD for expressive total languages. *Mathematical Structures in Computer Science* **33** (4-5) 311–426.
Mac Lane, S. (2013). *Categories for the Working Mathematician*, Vol. **5**, Springer Science & Business Media.
Margossian, C. C. and Betancourt, M. (2021). Efficient automatic differentiation of implicit functions. arXiv preprint arXiv: 2112.14217.
Mazza, D. and Pagani, M. (2021). Automatic differentiation in PCF. *Proceedings of the ACM on Programming Languages* **5** (POPL) 1–27.
Meijer, E. (2018). 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*, 1–1.
Mitchell, J. C. and Scedrov, A. (1992). Notes on sconing and relators. In: *International Workshop on Computer Science Logic*, Springer, pp. 352–378.
Moggi, E. (1989). Computational lambda-calculus and monads. In: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, *June 5-8, 1989*, *Pacific Grove, California, USA*, IEEE Computer Society, 14–23.
Moggi, E. (1991). Notions of computation and monads. *Information and Computation* **93** (1) 55–92.
Pearlmutter, B. and Siskind, J. (2008). Reverse-mode AD in a functional framework: lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)* **30** (2) 7–36.
Pitts, A. (1996). Relational properties of domains. *Information and Computation* **127** (2) 66–90.
Plotkin, G. (2018). Some principles of differential programming languages. In: *Invited talk, POPL, 2028*.
Shaikhha, A., Fitzgibbon, A., Vytiniotis, D. and Peyton Jones, S. (2019). Efficient differentiable programming in a functional array-processing language. *Proceedings of the ACM On Programming Languages* **3** (ICFP) 97–30.
Shalev-Shwartz, S., et al. (2012). Online learning and online convex optimization. *Foundations and Trends® in Machine Learning* **4** (2) 107–194.
Smeding, T. and Vákár, M. (2023). Efficient dual-numbers reverse AD via well-known program transformations. *Proceedings of the ACM on Programming Languages* **7** (POPL) 1573–1600.
Smeding, T. J. and Vákár, M. I. (2024). Parallel dual-numbers reverse ad. arXiv preprint arXiv: 2207.03418v3.
Smyth, M. and Plotkin, G. (1982). The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing* **11** (4) 761–783.

Socher, R., Lin, C. C., Manning, C. and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*, 129–136.
Street, R. (1972). The formal theory of monads. *Journal of Pure and Applied Algebra* **2** (2) 149–168.
Tai, K. S., Socher, R. and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv: 1503.00075.
Tsiros, P., Bois, F. Y., Dokoumetzidis, A., Tsiliki, G. and Sarimveis, H. (2019). Population pharmacokinetic reanalysis of a diazepam pbpk model: a comparison of stan and gnu mcsim. *Journal of Pharmacokinetics and Pharmacodynamics* **46** (2) 173–192.
van Merrienboer, B., Breuleux, O., Bergeron, A. and Lamblin, P. (2018). Automatic differentiation in ml: where we are and where we should be going. In: *Advances in Neural Information Processing Systems*, 8757–8767.
Vákár, M. (2020). Denotational correctness of forward-mode automatic differentiation for iteration and recursion. arXiv preprint arXiv: 2007.05282.
Vákár, M. (2021). Reverse AD at higher types: pure, principled and denotationally correct. In: *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*, Springer. 607–634.Proceedings,
Vákár, M., Kammar, O. and Staton, S. (2019). A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages* **3** (POPL) 36:1–36:29.
Vákár, M. and Smeding, T. (2022). CHAD: combinatory homomorphic automatic differentiation. *ACM Transactions on Programming Languages and Systems* **44** (3) 20:1–20:49.
Zhang, X., Lu, L. and Lapata, M. (2016). Top-down tree long short-term memory networks. In: *Proceedings of NAACL-HLT*, 310–320.

A. Fine grain call-by-value and AD

In Section 6, we have discussed a standard coarse-grain CBV language, also known as the λ_C-calculus, computational λ-calculus (Moggi 1989), or, plainly, CBV. In this appendix, we discuss an alternative presentation in terms of fine-grain CBV[20] (Levy et al. 2003; Levy 2012). 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.

$$
\begin{array}{llllll}
\tau,\sigma,\rho ::= & & \text{types} & | \ \mathbf{1} \mid \tau_1 \times \tau_2 & \text{products} \\
\quad | & \mathbf{real} & \text{numbers} & | \ \tau \to \sigma & \text{function} \\
\quad | & \mathbf{0} \mid \tau + \sigma & \text{sums} & & \\
\\
v,y,u ::= & & \text{values} & | \ \mathbf{case}\ v\ \mathbf{of}\{\begin{array}{l}\mathbf{inl}\ x \to w \\ \mid \mathbf{inr}\ y \to u\end{array}\} & \text{sum match} \\
\quad | & x,y,z & \text{variables} & | \ \langle\,\rangle \mid \langle v, w\rangle & \text{tuples} \\
\quad | & \underline{c} & \text{constant} & | \ \mathbf{case}\ v\ \mathbf{of}\ \langle x, y\rangle \to w & \text{product match} \\
\quad | & \mathbf{case}\ v\ \mathbf{of}\{\,\} & \text{sum match} & | \ \lambda x.t & \text{abstractions} \\
\quad | & \mathbf{inl}\ v \mid \mathbf{inr}\ v & \text{inclusions} & | \ \mu x.v & \text{term recursion} \\
\\
t,s,r ::= & & \text{computations} & & \\
\quad | & t\ \mathbf{to}\ x.\ s & \text{sequencing} & | \ \mathbf{case}\ v\ \mathbf{of}\ \{\begin{array}{l}\mathbf{inl}\ x \to t \\ \mid \mathbf{inr}\ y \to s\end{array}\} & \text{sum match}
\end{array}
$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

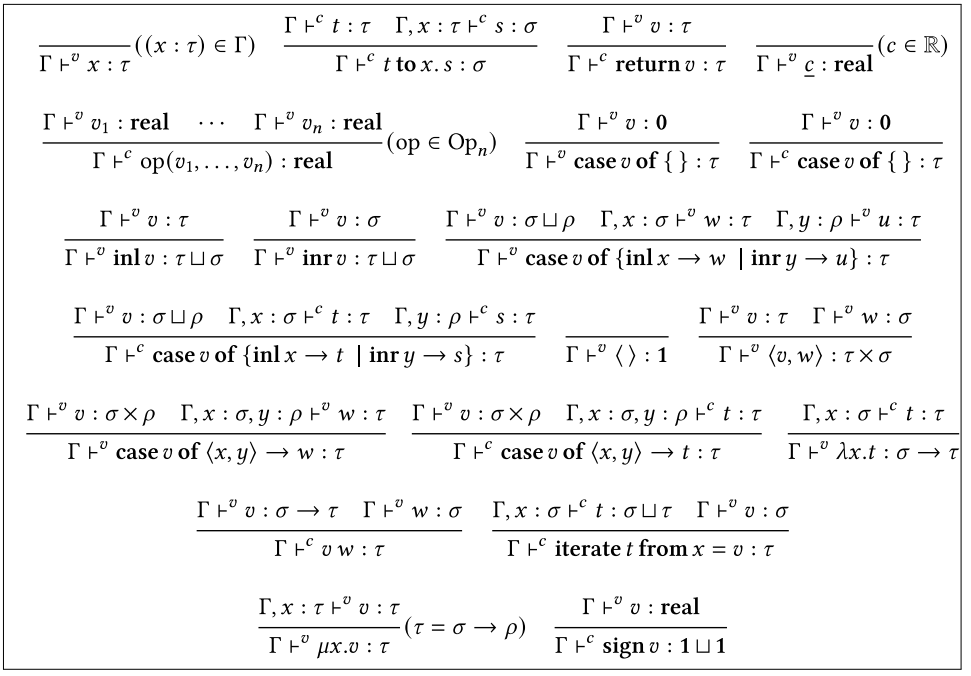


Figure A1. Typing rules for the our fine-grain CBV language with iteration and real conditionals. We use a typing judgement $\vdash^v$ for values and $\vdash^c$ for computations.

$\mid \mathbf{return}\, v$	pure comp.	$\mid$	$\mathbf{case}\, v\, \mathbf{of}\, \langle x, y\rangle \to t$	product match
$\mid \mathrm{op}(v_1, \ldots, v_n)$	operation	$\mid$	$v\ w$	function app.
$\mid \mathbf{case}\, v\, \mathbf{of}\{\,\}$	sum match	$\mid$	$\mathbf{iterate}\, t\, \mathbf{from}\, x = v$	iteration
$\mid$			$\mathbf{sign}\, v$	sign function

We will use sugar

$$\mathbf{if}\, v\, \mathbf{then}\, t\, \mathbf{else}\, s \stackrel{\text{def}}{=} \mathbf{sign}\,(v)\, \mathbf{to}\, x.\, \mathbf{case}\, x\, \mathbf{of}\, \{\_ \to s \mid \_ \to r\}$$
$$\mathbf{fst}\, v \stackrel{\text{def}}{=} \mathbf{case}\, v\, \mathbf{of}\, \langle x, \_\rangle \to x$$
$$\mathbf{snd}\, v \stackrel{\text{def}}{=} \mathbf{case}\, v\, \mathbf{of}\, \langle \_, x\rangle \to x$$
$$\mathbf{let\ rec}\, f(x) = t\, \mathbf{in}\, s \stackrel{\text{def}}{=} (\mu f.\, \mathbf{return}\, (\lambda x.t))\, \mathbf{to}\, f.\, s.$$

We could also define iteration as syntactic sugar: $\mathbf{iterate}\, t\, \mathbf{from}\, x = v \stackrel{\text{def}}{=} \left(\mu z.\lambda x.t\, \mathbf{to}\, y.\, \mathbf{case}\, y\, \mathbf{of}\{\mathbf{inl}\, x' \to z\, x' \mid \mathbf{inr}\, x'' \to \mathbf{return}\, x''\}\right)\, v$.

The typing rules are in Fig. A1.

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. A2.

Under the translation of coarse-grain CBV into fine-grain CBV, this equational theory induces precisely that of Section 6.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

Figure A2. Standard $\beta\eta$-laws for fine-grain CBV. We write $\#x_1, \ldots, x_n\equiv$ 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.

A.3 The CBV model $(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_\mu, \mathbf{Syn}_{\mathrm{it}})$

Our fine grain call-by-value language corresponds with a *CBV* model (see Definition 4.4).

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

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

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 Definition 4.1. Moreover, considering the free recursion and free iteration

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

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

Proposition 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*

$$H(\mathbf{real}) \in \mathrm{ob}\,\mathcal{V} \tag{A1}$$

$$H(\underline{c}) \in \mathcal{V}\,(1, H(\mathbf{real})) \tag{A2}$$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$H(\mathrm{op}) \in C\left(H(\mathbf{real})^n, H(\mathbf{real})\right) = \mathcal{V}\left(H(\mathbf{real})^n, TH(\mathbf{real})\right), \text{ for each } \mathrm{op} \in \mathrm{Op}_n \tag{A3}$$

$$H(\mathbf{sign}\ ) \in C\left(H(\mathbf{real}), 1 \sqcup 1\right) = \mathcal{V}\left(H(\mathbf{real}), T\left(1 \sqcup 1\right)\right) \tag{A4}$$

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

Proposition 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*

$$H(\mathbf{real}) \in \mathsf{ob}\,\mathcal{V} \tag{A5}$$

$$H(\underline{c}) \in \mathcal{V}\left(1, H(\mathbf{real})\right) \tag{A6}$$

$$H(\mathrm{op}) \in C\left(H(\mathbf{real})^n, H(\mathbf{real})\right) = \mathcal{V}\left(H(\mathbf{real})^n, TH(\mathbf{real})\right), \text{ for each } \mathrm{op} \in \mathrm{Op}_n \tag{A7}$$

$$H(\mathbf{sign}\ ) \in C\left(H(\mathbf{real}), 1 \sqcup 1\right) = \mathcal{V}\left(H(\mathbf{real}), T\left(1 \sqcup 1\right)\right) \tag{A8}$$

there is a unique CBV model morphism H between $\left(\mathbf{Syn}_V, \mathbf{Syn}_{\mathcal{S}}, \mathbf{Syn}_{\mu}, \mathbf{Syn}_{\mathsf{it}}\right)$ 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	$\mathbf{return}\ x$
$\mathbf{let}\ x = t\ \mathbf{in}\ s$	$t^\dagger\ \mathbf{to}\ x.s^\dagger$
$\underline{c}$	$\mathbf{return}\ \underline{c}$
$\mathbf{inl}\ t$	$t^\dagger\ \mathbf{to}\ x.\ \mathbf{return}\ \mathbf{inl}\ x$
$\mathbf{inr}\ t$	$t^\dagger\ \mathbf{to}\ x.\ \mathbf{return}\ \mathbf{inr}\ x$
$\langle\rangle$	$\mathbf{return}\ \langle\rangle$
$\langle t, s\rangle$	$t^\dagger\ \mathbf{to}\ x.s^\dagger\ \mathbf{to}\ y.\ \mathbf{return}\ \langle x, y\rangle$
$\lambda x.t$	$\mathbf{return}\ \lambda x.t^\dagger$
$\mathrm{op}(t_1, \ldots, t_n)$	$t_1^\dagger\ \mathbf{to}\ x_1 .. \ldots\ t_n^\dagger\ \mathbf{to}\ x_n.\ \mathrm{op}(x_1, \ldots, x_n)$
$\mathbf{case}\ t\ \mathbf{of}\ \{\ \}$	$t^\dagger\ \mathbf{to}\ x.\mathbf{case}\ x\ \mathbf{of}\{\ \}$
$\mathbf{case}\ t\ \mathbf{of}\ \{\mathbf{inl}\ x \to s\ \mid \mathbf{inr}\ y \to r\}$	$t^\dagger\ \mathbf{to}\ z.\mathbf{case}\ z\ \mathbf{of}\ \{\mathbf{inl}\ x \to s^\dagger\ \mid \mathbf{inr}\ y \to r^\dagger\}$
$\mathbf{case}\ t\ \mathbf{of}\ \langle x, y\rangle \to s$	$t^\dagger\ \mathbf{to}\ z.\mathbf{case}\ z\ \mathbf{of}\ \langle x, y\rangle \to s^\dagger$
$t\ s$	$t^\dagger\ \mathbf{to}\ x.s^\dagger\ \mathbf{to}\ y.x\ y$
$\mathbf{iterate}\ t\ \mathbf{from}\ x = s$	$s^\dagger\ \mathbf{to}\ y.\mathbf{iterate}\ t^\dagger\ \mathbf{from}\ x = y$
$\mathbf{sign}\ \ t$	$t^\dagger\ \mathbf{to}\ x.\mathbf{sign}\ \ x$
$\mu z.t$	$\mu z.\lambda x.t^\dagger\ \mathbf{to}\ y.\ yx$

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

$$\mathcal{D}(\mathbf{real}) \stackrel{\text{def}}{=} \mathbf{real} \times \mathbf{vect} \qquad \mathcal{D}(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0} \qquad \mathcal{D}(\tau \sqcup \sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau) \sqcup \mathcal{D}(\sigma)$$
$$\mathcal{D}(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1} \qquad \mathcal{D}(\tau \to \sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau) \to \mathcal{D}(\sigma) \qquad \mathcal{D}(\tau \times \sigma) \stackrel{\text{def}}{=} \mathcal{D}(\tau) \times \mathcal{D}(\sigma)$$

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

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

Figure A3. A forward-mode AD macro defined on types as $\mathcal{D}(-)$, values as $\mathcal{D}_{\mathcal{V}}(-)$, and computations as $\mathcal{D}_C(-)$. All newly introduced variables are chosen to be fresh.

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. A3, 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: $\mathcal{D}_C(\mathbf{if}\, v\, \mathbf{then}\, t\, \mathbf{else}\, s) = \mathbf{case}\, \mathcal{D}_{\mathcal{V}}(v)\, \mathbf{of} \langle x, \_\rangle \to \mathbf{if}\, x\, \mathbf{then}\, \mathcal{D}_C(t)\, \mathbf{else}\, \mathcal{D}_C(s)$.

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 Section 6.

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

B. A more efficient derivative for sign

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

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

and

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

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.

C. Enriched scone

We present straightforward generalizations (enriched versions) of the results presented in Lucatelli Nunes and Vákár (2023, Section 9) below.

Considering the **ωCpo**-category 2 with two objects and only one nontrivial morphism between them, the **ωCpo**-category $2 \pitchfork \mathcal{D}$ of morphisms of $\mathcal{D}$ can be described as the **ωCpo**-category **ωCpo-Cat** $[2, \mathcal{D}]$ of **ωCpo**-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 (**ωCpo**-)natural transformation. Finally, the **ωCpo**-structure is defined by $(\alpha_0, \alpha_1) \leq (\beta_0, \beta_1)$ if $\alpha_0 \leq \beta_0$ and $\alpha_1 \leq \beta_1$ in $\mathcal{D}$.

Given an **ωCpo**-functor $G : \mathcal{C} \to \mathcal{D}$, the comma category $\mathcal{D} \downarrow G$ of the identity on $\mathcal{D}$ along G in **ωCpo-Cat** is also known as the **ωCpo**-*scone* or *Artin glueing* of G. It can be described as the pullback (C1) in **ωCpo-Cat**, in which $\mathrm{codom} : 2 \pitchfork \mathcal{D} \to \mathcal{D}$, defined by $\big(\alpha = (\alpha_0, \alpha_1) : f \to g\big) \mapsto \alpha_1$, is the codomain **ωCpo**-functor.

$$\begin{array}{ccc}
\mathcal{D} \downarrow G & \xrightarrow{\mathrm{proj}_{2 \pitchfork \mathcal{D}}} & 2 \quad \mathcal{D} \\
{\scriptstyle \mathrm{proj}_C}\downarrow & & \downarrow{\scriptstyle \mathrm{codom}} \\
\mathcal{C} & \xrightarrow[G]{} & \mathcal{D}
\end{array} \tag{C1}$$

Since codom is an isofibration, the pullback (C1) is equivalent to the pseudo-pullback of codom along G, which is the **ωCpo**-category defined as follows. The objects of the pseudo-pullback are

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

triples

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

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

Lemma C.1. *The forgetful* $\boldsymbol{\omega}$**Cpo**-*functor* $\mathcal{L}\,:\,\mathcal{D}\downarrow G\to\mathcal{D}\times\mathcal{C}$, *defined in* (37), *creates all absolute (weighted) limits and colimits.*

Proof Clearly, the $\boldsymbol{\omega}$**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 $\boldsymbol{\omega}$**Cpo**-functor. Since $\mathcal{D}\downarrow G$ is the pullback (C1), 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 $(co)\mathsf{lim}\,(W, \mathrm{dom}D_0)\cong\pi_{\mathcal{D}}\,((co)\mathsf{lim}\,(W, \mathcal{L}\circ D))$ and $(co)\mathsf{lim}\,(W, \mathrm{codom}\circ D_0)\cong G\circ\pi_{\mathcal{C}}\,((co)\mathsf{lim}\,(W, \mathcal{L}\circ D))$. 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 $(co)\mathsf{lim}\,(W, D_1)\cong\pi_{\mathcal{C}}\,((co)\mathsf{lim}\,(W, \mathcal{L}\circ D))$.

Therefore, the isomorphism ξ given by

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

together with the pair $\big((co)\mathsf{lim}\,(W, D_0)\,, (co)\mathsf{lim}\,(W, D_1)\big)$ 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:

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

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

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

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

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

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

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

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

D. Some Haskell Code for a 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
```

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

```
51
52 -- The implementation of the RNN
53 -- version 1, without caching
54 modelH ((w, b, wScore), (adjMat, globalScore )) =
55   let getNode (Leaf i) = a !! i
56    in let getNode (Node l r) = f (w `mult` conc (getNode l) (getNode r)) `add` b
57       in let parentsScores =
58                map
59                  (\ i -> (i, innerprod wScore (getNode (uncurry Node i))))
60                  adjMat -- compute scores for all parent nodes of neighbours;
61                  -- super inefficient without caching getNode, but conceptually cleaner
62          in let ((bp1, bp2), bestScore) =
63                   foldl
64                     (\( i, s) (i', s') ->
65                        if s > s'
66                          then (i, s)
67                          else (i', s'))
68                     (head parentsScores)
69                     parentsScores -- find the neighbours that have the higest score
70             in let globalScore2 = globalScore + bestScore
71             -- add the local contribution of our chosen neighbour pair to the global score
72                in let bestPar = Node bp1 bp2
73                   -- actually compute our favourite parent;
74                   -- I guess we'd already done this before but it's cheap to redo
75                      in let mergeParH i
76                               | i == bp1 || i == bp2 = bestPar
77                         in let mergeParH i
78                                  | otherwise = i
79                            in let mergePar (i, j) =
80                                     (mergeParH i, mergeParH j)
81                               in let adjMat2 =
82                                        filter
83                                          (/= (bestPar, bestPar))
84                                          [ mergePar (i, j)
85                                          | (i, j) <- adjMat
86                                          ]
87                                          -- replace bp1 and bp2 with bestPar in adjacencyMatrix,
88                                          -- but we have a convention that nodes are not neighbours
89                                          -- of themselves
90                                  in if null adjMat2
91                                       then Right globalScore2
92                                       else Left (adjMat, globalScore2)
93                                       -- if we run out of neighbours that can be merged, we are done;
94                                       -- otherwise iterate with new adjacency matrix and score
95
96 it :: ((c, a) -> Either a b) -> (c, a) -> b -- functional iteration
97 it f (c, a) =
98   case f (c, a) of
99     Left a' -> it f (c, a')
100    Right b -> b
101
102 model :: ((Matrix, Vector, Vector), (AdjacencyMatrix, Scalar)) -> Scalar
```

https://doi.org/10.1017/S0960129524000215 Published online by Cambridge University Press

```
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 ))
```

Cite this article: Lucatelli Nunes F and Vákár M (2024). Automatic differentiation for ML-family languages: Correctness via logical relations. *Mathematical Structures in Computer Science*. https://doi.org/10.1017/S0960129524000215